An Energy-Efficient Distributed Cut Vertex Detection Algorithm for Wireless Sensor Networks

Orhan Dagdeviren, Vahid Khalilpour Akram

Ege University,
International Computer Institute, Bornova, Izmir 35100, Turkey
Tel: +90 232 311 32 35, Fax: +90 232 388 72 30
Email: orhan.dagdeviren@ege.edu.tr

Maintaining connectivity is a very important objective of wireless sensor networks (WSNs) in successfully achieving data collection for applications. A cut vertex (node) is defined as a critical vertex whose removal disconnects a network component and partially disables data delivery. Hence, it is crucial that cut vertices be detected and treated with caution. In this paper, we propose an energy-efficient cut vertex detection algorithm for WSNs. Our algorithm uses a depth-first search approach and is completely distributed. It benefits from the radio multicast capabilities of sensor nodes and is the first algorithm with O(N) time complexity and O(N) sent message complexity, in which each message is O($log_2(N)$) bits. We show the operation of the algorithm, analyze it in detail, provide testbed experiments and extensive simulations. We compare our proposed algorithm with the other cut vertex detection algorithms and show that our algorithm saves up to 6.8 times more energy in less time.

Keywords: Wireless sensor networks; cut vertex detection; distributed algorithm; energy-efficient algorithm design

Received 00 Month 2012; revised 00 Month 2012

1. INTRODUCTION

Recent technological advancements in wireless network areas and related hardware have enabled the rapid development of low-cost, low-power, multifunctional miniature devices [1]. These small, inexpensive sensor devices have limited processing capabilities. WSNs are ad hoc networks without any infrastructure in which thousands of nodes perform sensing and communicating operations. WSNs can be used in applications such as health care, military surveillance, habitat monitoring, disaster relief and target tracking. In a sensor network application, a node transmits its sensed data to a target node and the data are relayed and aggregated to the sink node. To achieve robust data collection from the environment, network connectivity should be maintained. From a routing perspective, some nodes can be redundant because alternative paths to the sink node can be constructed. However, a network may include some critical nodes that have crucial tasks in the routing operation. These nodes are called cut vertices (nodes) and their removal breaks the connectivity of the network.

The detection of cut vertices is an important research area for various types of networks [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]. Cut vertex detection is crucial in various application scenarios. For example, in a periodic report delivery and reactive critical event delivery application such as is used for military border control, an entire sensing area can be monitored using mobile sensor nodes. The nodes should at least periodically report that they are alive and monitoring all spots within the sensing region [10]. If an intruder is detected by the sensor nodes, the event should be delivered to the sink node within a reasonable amount time. Cut vertex detection and neutralization are important services for this type of application to maintain continuous connectivity in cases of single node failure to ensure report and event deliveries. Detecting cut vertices can be vital in heterogeneous sensor networks in which actor nodes with extra capabilities are used in combination with ordinary sensor nodes [10, 17, 18]. In wireless sensor and actor network (WSAN) applications, actor nodes may be capable of actively affecting the physical environment, and they may communicate to make a global plan. For example, in a forest fire monitoring application, actor nodes may cooperate to extinguish the fire. During this cooperation, the counting of actor

nodes and their distance from fire spots can produce important parameters. To make the most efficient plan, all commissioned actors should participate the decision-making operation, which requires continuous connectivity between the nodes. When a cut actor is corrupted, connectivity restoration algorithms in WSAN may be reactively executed to reconstruct topology [10, 17, 18]. This restoration operation requires that cut actors be detected priori to classify the type of node failure and decide whether it is necessary to execute a connectivity restoration algorithm. Moreover, it is interesting that cut vertex information can be used as a connectivity restoration strategy in WSANs. As an example, the partition detection and connectivity restoration approach assigns a backup node to a cut actor node, with backup nodes tested so that a choice can be made among non-cut nodes to reduce incurred overhead [17].

WSN is an example of a distributed system in which unpredictable events may occur at any time and any place in the network without the control of a central device. From this point of view, algorithms should be distributed. Nodes are battery powered in WSNs; thus, energy consumption should be reduced to maximize the network lifetime. In a sensor node, the radio component consumes a significant amount of energy. Therefore, the amount of message transmission in a network should be reduced. Energy consumption grows with the message length; thus, message length should be as small as possible. These constraints must be considered during algorithm design in sensor networks.

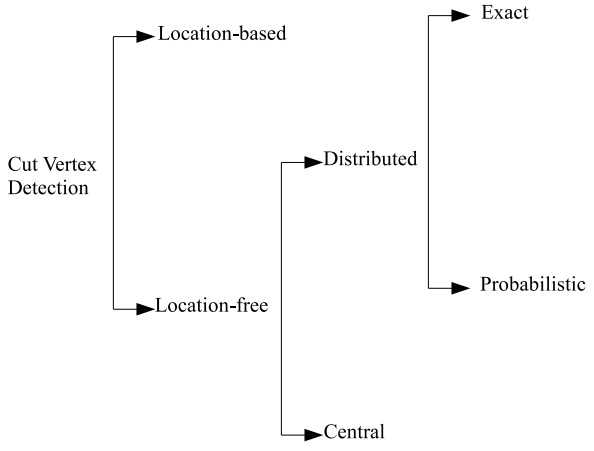

FIGURE 1. Main Research Lines of the Cut Vertex Detection Problem in Sensor Networks.

The main research lines of the cut vertex detection problem in WSNs are given in Fig. 1. Research approaches can be divided into two categories: Location-based and location-free algorithms. Because location-based [13, 15] algorithms require that coordinates be defined priori or localization services be executed, they are out of our scope. Location-free algorithms can be further divided into two groups: central and distributed. To execute central algorithms [2], neighborhood lists of all nodes should be collected by the sink node which may not be an energy-efficient operation as shown in this work. Distributed algorithms can be either exact [3, 4, 5, 6, 7, 9, 12] or probabilistic [8, 10, 11, 14, 16]. Because our concern in this paper is exactly detecting cut vertices exactly, we omit other distributed algorithms that can only detect within a probability interval. The exact algorithm proposed by Swaminathan [5] assumes that each connected component has a coordinator that should have a global view of its connected component. This assumption is unrealistic for WSNs as mentioned in [9]. Thurimella proposed a sub-linear time algorithm [6] by ignoring communication costs. The message complexity of the algorithm is as high as $O(k(M(D+N^{0.614})+N^{1.5}))$ for k-connected networks, where M is the edge count, D is the network diameter and N is the node count. This complexity bound can not be accepted for battery-powered sensor nodes. The exact distributed algorithms of Ahuja [3], Hohberg [4] and Turau [9] have $O(M)$ sent message complexity and an $O(\log_2(N))$ bits message size, where Δ is the maximum node degree. Although the algorithm of Chaudhuri[7] has $O(N)$ sent message complexity, it transmits long messages with an $O(N)$ bits size. The most recent study of exact localization free approaches is Xiong's algorithm [12], which has $O(\Delta N)$ sent message complexity with an $O(\log_2(N))$ message size. In this paper, we propose the distributed energy-efficient cut vertex detection algorithm (DENCUT), which is based on depth-first search and aims to reduce energy consumption by minimizing transmitted byte count during cut vertex detection. The contributions of this paper are as follows:

- By using the radio multicast communication capabilities of sensor networks, we propose the first depth-first-based exact cut vertex detection algorithm with $O(N)$ sent messages, each having $O(log_2(N))$ bits. In addition to this, time complexity remains $O(N)$ and space complexity is $O(\Delta)$. To the best of our knowledge, the DENCUT algorithm is theoretically (from complexity analysis) the best algorithm for energy-efficient WSNs.
- Using the overhearing technique, a leaf node (a node that has only one neighbor) may send just one message and terminates its local execution while its neighbor continues to execute the algorithm. This approach aims to reduce energy and time consumptions. In addition, this approach provides constant time complexity for star networks. To the best of our knowledge, the DENCUT is the first algorithm with constant time complexity at the best case.
- DENCUT is implemented on real sensor nodes and simulated using various sensor network setups in comparison with the previous work, and it

is shown that the proposed algorithm consumes much less energy. The time consumption of the algorithm are very low against various conditions. These improvements show that DENCUT has both practical (from testbed experiments and simulations) and theoretical importance.
- DENCUT is fully distributed and does not require localization, which makes it suitable for large-scale self-organizing sensor network applications.

The remainder of this paper is organized as follows. In Section 2, the network model and cut vertex detection problem are described. The related work is reviewed in Section 3. The proposed algorithm is described in Section 4. The detailed analyses of the algorithms are given in Section 5. Section 6 introduces the results of testbed experiments on real sensor nodes and simulations. Conclusions are given in Section 7.

2. BACKGROUND

In this section, we introduce the network model and the cut vertex detection problem.

2.1. Network Model

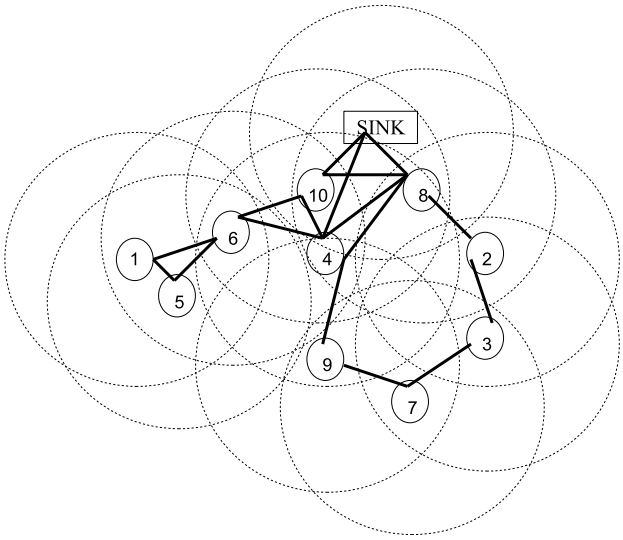

FIGURE 2. Undirected Graph Model.

The following assumptions are made about the network, as in [9, 19]:

- Each node has distinct *node_id*.
- Links between nodes are symmetric. Thus, if there is a link from u to v, there exists a reverse link from v to u.
- Nodes do not know their positions. They are not equipped with a position tracker such as with a GPS receiver.
- All nodes are equal in terms of processing capabilities, radio, battery and memory.
- Each node knows its 1-hop neighbors.
- Each node can send radio multicast messages to its immediate neighbors in its transmission range.
- Nodes are static and their transmission ranges are constant during the execution of the cut vertex algorithm. By applying this assumption, neighbor lists of nodes do not change and a consistent state of the network is provided as an input for the algorithm.

Based on these assumptions, the network may be modeled as an undirected graph $G(V, E)$ where V is the set of vertices and E is the set of the edges. An example undirected graph model is depicted in Fig. 2, with the nodes's transmission ranges represented by dotted circles.

2.2. The Cut Vertex Detection Problem

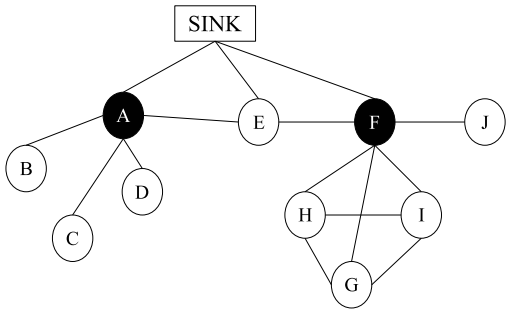

FIGURE 3. Cut Vertex Problem.

A path between u and v is a series of distinct vertices such as $<x_1,x_2,x_3,...,x_n>$ where $x_1=u$, $x_n=v$ and there is an edge between each x_i and x_{i+1}. A graph G is connected if there is a path between each pair (u,v):$u\neq v$, otherwise the graph G is disconnected. Vertex v is a cut vertex of G if and only if the G' obtained from G by removing v and its incident edges is disconnected. Cut vertices can occur in any location within the network. They can be the parent(s) of the leaf node(s) or the parent(s) of the component(s). An example sensor network is given in Fig. 3. In this figure, there are 10 nodes where node ids are written inside each node and the communication edges are indicated by solid lines. Node A and node F are cut vertices, filled in black. Node A is the parent of three leaf nodes (nodes B, C and D) and if it fails, 30% of the network cannot transmit its data to the sink node. Node F is the parent of a component consisting of nodes H, G, I and J. If node F fails, 40% of the total nodes cannot send their data to the sink node. This simple example shows that the cut vertices can be very critical to the data delivery operation in sensor networks. Thus, the cut vertices should be detected and precautions should be taken. After detecting cut vertices, neighbors of the cut vertices can extend their transmission range or move to other locations to create new paths. These considerations are beyond the scope

of this paper, because creating k-connected topology is another research thread [18]. We do not focus on detecting cut edges (bridges) in this paper, rather because bridge detection targets the prevention of edge failures rather than node failures [20, 21]. In this paper, our focus is the energy-efficient, distributed detection of cut vertices in the sensor network environment. Our objectives are listed below as follows:

1. The cut vertex detection algorithms should be efficient in terms of time, message, space and computational complexities to provide low resource consumption for sensor networks.
2. The sink node may initiate the cut vertex detection algorithm at any time locally. Hence, these operations should be distributed.
3. The cut vertex detection algorithm should be independent of the underlying protocols as much as possible to interface with various MAC and physical layer standards.
4. The algorithm should not be dependent on localization information.

3. RELATED WORK

Depth-first search is a fundamental algorithm in computer science with numerous applications [22]. Cut vertex detection using depth-first search was first introduced by Tarjan [2]. To detect cut vertices, Tarjan proposed the following list of rules.

- *Rule 1:* For each vertex a *low* value should be stored and $low[i]$ should be equal to min($depth[i]$, min(*low* values of children of i), min(*depth* values of back vertices of i)) where $depth[i]$ is the hop distance of i to root vertex in DFS tree and back vertices are neighbors other than parent and children.
- *Rule 2:* A non-root vertex i is a cut vertex if and only if one of its children j has no descendant of l with a back edge connecting it above i. Briefly, if $depth[i] \leq low[c]$ for any child c of vertex i then i is a cut vertex.
- *Rule 3:* The root vertex is a cut vertex if it is has two or more children.

This algorithm (CENTRAL) is completely central and can be implemented by the sink node after collecting lists of neighboring nodes. This operation starts by constructing a breadth-first search tree rooted at the sink node, then each node can send its neighbor list by using this tree. The sent message complexity of this operation is $O(DN)$ and the message size is $O(\Delta log_2(N))$ bits. In this paper, we show with testbed experiments and simulations that our algorithm outperforms this algorithm.

Generally, depth-first search-based distributed algorithms use Tarjan's rules to detect cut vertices. Ahuja proposed a DFS-based distributed cut vertex detection algorithm (AHUJA) with $4M$-N messages and $2N$-2 time units. The message size of Ahuja's algorithm is $O(log_2(N))$ bits. The message size of Hohberg's algorithm (HOHBERG) [4] is the same and it consumes $2M$-N-1 messages and time units. Chaudhuri [7] proposed a distributed algorithm (TOKEN) for cut detection with $O(N)$ messages and time units. A token is used in each message and each discovered node inserts its id into this token. Although this algorithm is optimal in terms of message and time complexity, the message size of the algorithm is $O(N)$ bits. Thus, the received byte count and the sent byte count of the algorithm are as large as $O(N^2)$. Swaminathan and Goldman [5] proposed an algorithm for computing 2-connected components. This algorithm works for dynamic graphs. It takes $O(b+c)$ messages with $O(b(b+e))$ to recompute the 2-connected components after each edge update where b, e and c are the node count, edge count and 2-connected component count, respectively. The algorithm is not suitable for sensor networks as mentioned in [9]. Thurimella's algorithm is designed for time efficiency and its time complexity is $O(D+g(N)+\sqrt{N})$, where $g(N)$ is the time complexity of computing a $\sqrt{N}$-dominating set. Its message complexity is $O(k(M(D+N^{0.614})+N^{1.5}))$ which is not suitable for battery-powered sensor networks. Turau proposed an algorithm (TURAU) [9] that transmits $4M$ messages with $O(log_2(N))$ bits. This algorithm takes $O(N)$ time to complete. We show in this paper that our algorithm outperforms AHUJA, HOHBERG, TOKEN and TURAU algorithms both theoretically and practically.

Xiong and Li [12] proposed a coded spanning tree-based approach to detect cut vertices in wireless sensor networks. The cut vertex detection algorithm (CVD) scans the nodes of a network in parallel to utilize edge color mechanism on the interval coded spanning tree. The algorithm consists of three phases, a tree building phase, a tree coding phase and a cut detection phase. In the tree building phase, a distributed spanning tree algorithm is executed. In the tree coding phase, interval codes are assigned to each node. In the cut detection phase, cut vertices are identified. The message and time complexities of the algorithm are $O(\Delta N)$ and each message has a length of $O(log_2(N))$ bits. Our algorithm is theoretically better than this algorithm because its complexity of sent messages and its time complexity are $O(N)$. We also show by testbed experiments and simulations that our proposed algorithm is practically better than the CVD algorithm. The algorithms covered thus far exactly find cut vertices without localization. In this study, we omit the probabilistic cut vertex detection algorithms given in [8, 10, 11, 14, 16] and localization-based cut vertex detection algorithms given in [13, 15].

A summary and theoretical comparison of DENCUT, CVD, TURAU, TOKEN, HOHBERG, AHUJA and CENTRAL algorithms are given in Table 1. A radio multicast and overhearing based MAC layer is assumed

TABLE 1. Analytical Comparison of Cut Vertex Detection Algorithms for WSNs.

Algorithm/ Complexity	Sent Messages	Received Messages	Message Size	Time	Space (per node)	Computational (per node)
DENCUT	O(N)	O(ΔN)	O($log_2(N)$)	O(N)	O(ΔN)	O(Δ)
CVD	O(ΔN)	O($\Delta^2 N$)	O($log_2(N)$)	O(ΔN)	O(ΔN)	O(Δ^2)
TURAU	O(M)	O(ΔM)	O($log_2(N)$)	O(N)	O(ΔN)	O(Δ)
TOKEN	O(N)	O(ΔN)	O(N)	O($Nlog_2(N)$)	O(ΔN)	O(Δ)
HOHBERG	O(M)	O(ΔM)	O($log_2(N)$)	O(N)	O(ΔN)	O(Δ)
AHUJA	O(M)	O(ΔM)	O($log_2(N)$)	O(N)	O(ΔN)	O(Δ)
CENTRAL	O(DN)	O(ΔDN)	O($\Delta log_2(N)$)	O(N)	O(ΔN+M)	O(N+M)

in this comparison. As seen in Table 1, DENCUT is equal to or better than all of the algorithms for sent message complexity, received message complexity, message size, time complexity, space complexity and computational complexity metrics. To validate and strengthen our theoretical findings, we provide testbed experiment measurements and extensive simulation results in the following sections.

4. DISTRIBUTED CUT VERTEX DETECTION ALGORITHM

In this section, we provide the concept, description and an example operation of the proposed algorithm.

4.1. Our Approach

We propose the distributed energy-efficient cut vertex detection algorithm (DENCUT) for WSNs. It detects cut vertices using a depth-first search-based approach and its main goal is the energy-efficient detection of cut vertices consuming reasonable time and memory space. The algorithm is fully distributed and triggered by the sink node. Thus, it is completely suitable for operating in a self-organized sensor network environment. The algorithm has two phases of execution: forward and backward. The node executes forward phase searches for an undiscovered neighbor and includes it in its children list. When a node does not have any undiscovered neighbors, it enters the backtrack phase and waits for its children, then informs its parent and terminates the execution. To decrease send message count, our algorithm benefits from the radio multicast capability of sensor nodes. The overhearing technique is applied by sensor nodes during algorithm execution. Although this is a well known technique for wireless ad hoc networks, especially used in topology control algorithms such as dominating set algorithms [23, 24, 25, 26, 27], DENCUT is the first algorithm that applies this technique among other cut detection approaches. In contrast to previous approaches, a new discovered node in forward phase informs its neighbors by multicasting a single message. The same operation is applied during the backtrack phase. These extensions provide an asymptotical decrease in the complexity of sent messages with O($\log_2(N)$) bits from O($N\Delta$) to O(N) in the worst case. Another extension of DENCUT is designed for the leaf nodes. When a leaf node overhears a discovery message, it informs its parent and suddenly terminates the algorithm. By applying this method, time complexity becomes constant for star networks in the best case. This is our second asymptotic improvement over previous work. Moreover, this fast-quit method aims to decrease the count of overheard messages.

4.2. Description

The local algorithm consists of sending messages over adjoining links, waiting for incoming messages and processing messages. The pseudo-code of the algorithm is given in Alg. 1.

Initially, each node stores a Γ set that includes its one-hop neighbors. A *low* variable and a *depth* variable are used. Each node also stores the parent's id in *parent* variable, a *root* boolean variable indicates whether it is root or not and a *cut_vertex* boolean variable is true only if the node is a cut vertex. A *states* array includes states for each node in Γ. Possible values in each entry of the states array are $UNVISITED$, $PARENT$, $BACK\_NODE$, $CHILD$ and $BACK\_CHILD$. The descriptions of these states in this array are as follows:

- $UNVISITED$: Initially all entries for each node are set to $UNVISITED$.
- $PARENT$: A node sets its parent's state to $PARENT$.
- $CHILD$: A node sets its states of children to $CHILD$.
- $BACK\_CHILD$: A node that receives a $BACKTRACK$ message from its child i sets i's state to $BACK\_CHILD$.
- $BACK\_NODE$: If a node overhears a $BACKTRACK$ message from its neighbor i that is not in one of its $CHILD$ and $PARENT$ states, it sets i's state to $BACK\_NODE$.

The message types used in the proposed algorithm are $START$ (*destination*), $DISCOVER$ (*source*,

Algorithm 1 DENCUT Algorithm for node$_i$

1: **initially**
2: Γ is the set of neighbors; states[j] $\leftarrow$ UNVISITED $\forall$ j $\in \Gamma$
3: low $\leftarrow$ parent $\leftarrow \infty$; root $\leftarrow$ cut_vertex $\leftarrow$ false
4: i is the unique id; sink sends START(v) to any node to trigger algorithm
5: **upon** receiving START(v)
6: **if** *v=i* **then** *// root node i starts algorithm.*
7: low $\leftarrow$ depth $\leftarrow$ 0; root $\leftarrow$ true
8: call discoverNeighbor()
9: **end if**
10: **end upon**
11: **upon** receiving DISCOVER(u,v,p_depth)
12: **if** v=i and parent=∞ and $|\Gamma| \neq 1$ **then** *// non-leaf node i is discovered by node u.*
13: parent $\leftarrow$ u; depth $\leftarrow$ p_depth
14: low $\leftarrow$ min(low,depth); states[u] $\leftarrow$ PARENT
15: call discoverNeighbor()
16: **else if** $|\Gamma| \neq 1$ **then** *// non-leaf node i classifies node u as BACK_NODE.*
17: low $\leftarrow$ min(low,p_depth-1); states[u] $\leftarrow$ BACK_NODE
18: **else if** $|\Gamma|=1$ and root=false **then** *// leaf node i is discovered by node u.*
19: parent $\leftarrow$ u; depth $\leftarrow$ p_depth; low $\leftarrow$ min(low,p_depth-1)
20: send FINISH(v) to v; terminate the execution of the algorithm
21: **end if**
22: **end upon**
23: **upon** receiving BACKTRACK(u,i,p_low)
24: **if** v=i **then** *// node i receives BACKTRACK from its child node u.*
25: states[u] $\leftarrow$ BACK_CHILD; low $\leftarrow$ min(low,p_low)
26: **if** depth $\leq$ p_low and root=false **then**
27: cut_vertex $\leftarrow$ true
28: **else if** $\exists$ j_1,j_2 s.t. ($j_1 \neq j_2$ and (states[$j_{1,2}$]=CHILD or states[$j_{1,2}$]=BACK_CHILD) and root=true **then**
29: cut_vertex $\leftarrow$ true
30: **end if**
31: call discoverNeighbor()
32: **else** *// node i classifies node u as BACK_NODE.*
33: states[u] $\leftarrow$ BACK_NODE
34: **end if**
35: **end upon**
36: **upon** receiving FINISH(u)
37: **if** states[u]=CHILD **then** *// DISCOVER(i,u,depth) and FINISH(u) arrive at the same time.*
38: states[u] $\leftarrow$ BACK_CHILD
39: **call** discoverNeighbor()
40: **end if**
41: states[u] $\leftarrow$ BACK_CHILD; cut_vertex $\leftarrow$ true
42: **end upon**
43: **procedure** discoverNeighbor()
44: **if** $\exists$ j s.t. states[j]=UNVISITED **then** *// node i discovers node j.*
45: states[j] $\leftarrow$ CHILD
46: send DISCOVER(i,j,depth+1) to Γ
47: **else if** $\nexists$ j s.t. states[j]=CHILD **then** *// node i has no undiscovered neighbor, starts backtrack phase.*
48: **if** root=false **then**
49: send BACKTRACK(i,parent,low) to parent
50: **else** *// only root node reports the termination.*
51: report the termination of execution to the sink
52: **end if**
53: terminate the execution of the algorithm *// all nodes terminate the execution.*
54: **end if**
55: **end procedure**

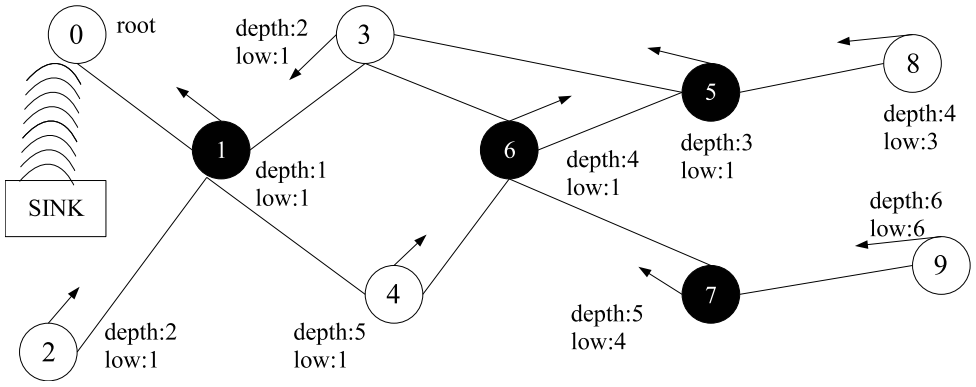

FIGURE 4. An Example Operation of DENCUT.

destination, *depth_value*), $FINISH$ (*source*) and $BACKTRACK$ (*source*, *destination*, *low*). These message types are described as follows:

- $START$: Sink triggers the algorithm by sending a $START$ message to one of its neighbors. If a node u receives a $START$ message, it initiates its local variables and starts neighbor discovery.
- $DISCOVER$: When a node u discovers a new child v for DFS tree, it sends a $DISCOVER$ (u, v, *depth*+1) to its neighbors. When a node v receives a $DISCOVER$ (u, v, *depth*+1), it sets its *parent* to v and starts a neighbor discovery operation. When a non-leaf node x overhears a $DISCOVER$ message, it sets its *low* value.
- $FINISH$: When a leaf node y overhears a $DISCOVER$ message from a node u, it sends a $FINISH$ (u). Node u sets *cut_vertex* to true after receiving the $FINISH$ message from node y.
- $BACKTRACK$: When a node s has no $CHILD$ link, it executes backward phase by sending a $BACKTRACK$ (x, $parent_x$, *low*) message to $parent_x$. When the root node receives $BACKTRACK$ messages from all of its neighbors in $CHILD$ state, it reports the termination of execution to the sink node.

4.3. An Example Operation

Assume a WSN with 10 nodes in which the id of the nodes and the edges connecting them are given, as shown in Fig. 4. Each node's *depth* and *low* values are written near it. The *parent* of each node is shown with a small arc. Cut vertices detected after the execution are filled in black in Fig. 4. The diagram of the message transfers are shown in Fig. 5. Multiple arrows originating from a node show a neighbor multicast message in this figure. The tracing of an example DENCUT operation is as follows. The sink node sends a $START$ message to node 0 to initiate the algorithm. This action is depicted by the uppermost arrow in Fig. 5. When node 0 receives the $START$ message it becomes the root node, sets its *depth* and *low* values to 0 and starts neighbor discovery operation by sending a $DISCOVER$(0,1,1) message to node 1 as shown in Fig. 5. After node 1 receives $DISCOVER$(0,1,1), it sets its *parent* value to 0, *depth* value to 1, *low* value to 1 and multicasts a $DISCOVER$(1,3,2) message to its neighbors as depicted by the multiple arrows originating from node 1 in Fig. 5. When node 3 receives this $DISCOVER$(1,3,2) message, it sets its *parent* to 1, its *depth* value to 2, its *low* value to 2 and executes neighbor discovery. Concurrently, nodes 2 and 4 overhear the $DISCOVER$(1,3,2) message. Node 2 is a leaf node so it sends a $FINISH$(2) message to node 1 and terminates its execution of the algorithm. Node 4 sets its *low* value to 1. When node 1 receives the $FINISH$(2) message, it becomes a cut vertex by setting *cut_vertex* to *true*. For all message transfers please see Fig. 5.

5. ANALYSIS

In this section, we will analyze the proof of correctness, message, time, space and computational complexities of the DENCUT algorithm.

5.1. Proof of Correctness

Our distributed algorithm can be represented with an extended finite state machine as a 6-tuple, (S, S_0, I, L, K, T) where S is the set of states, S_0 is the set of start states ($S \subseteq S_0$), I is the input messages, L is the local variables, K is the compound boolean statements of L and T is the transition function ($S\times$ ($I \wedge K$) $\rightarrow$ S). To prove the correctness of the DENCUT algorithm, we provide a finite state machine representation of the algorithm in Fig. 6. In this figure, S=$\{IDLE$, $WAIT\_BACKTRACK$, END, $ROOT$, $ROOT\_END\}$, S_0=$\{IDLE\}$, I=$\{START$, $DISCOVER(x$, y=i, *p_depth*), $DISCOVER(x$, $y \neq i$, *p_depth*), ...$\}$, L=$\{N$, $\Gamma\}$, K=$\{$ ($|\Gamma|$=1 $\vee$ $|N|$=0), ($|N \setminus \{x\}|$=0 $\vee$ $|C \setminus \{x\}|$=0), ($|N \setminus \{x\}| \neq 0 \vee |C \setminus \{x\}| \neq 0$), ...$\}$, T=$\{$*Transition 1*, *Transition 2*, ...*Transition 14* $\}$ and *Transition 1*=(send $DISCOVER(i,j{:}j \in N$, *depth*+1) to Γ, $N \leftarrow N\ \{j\}$, $C \leftarrow C \cup \{j\}$). For the

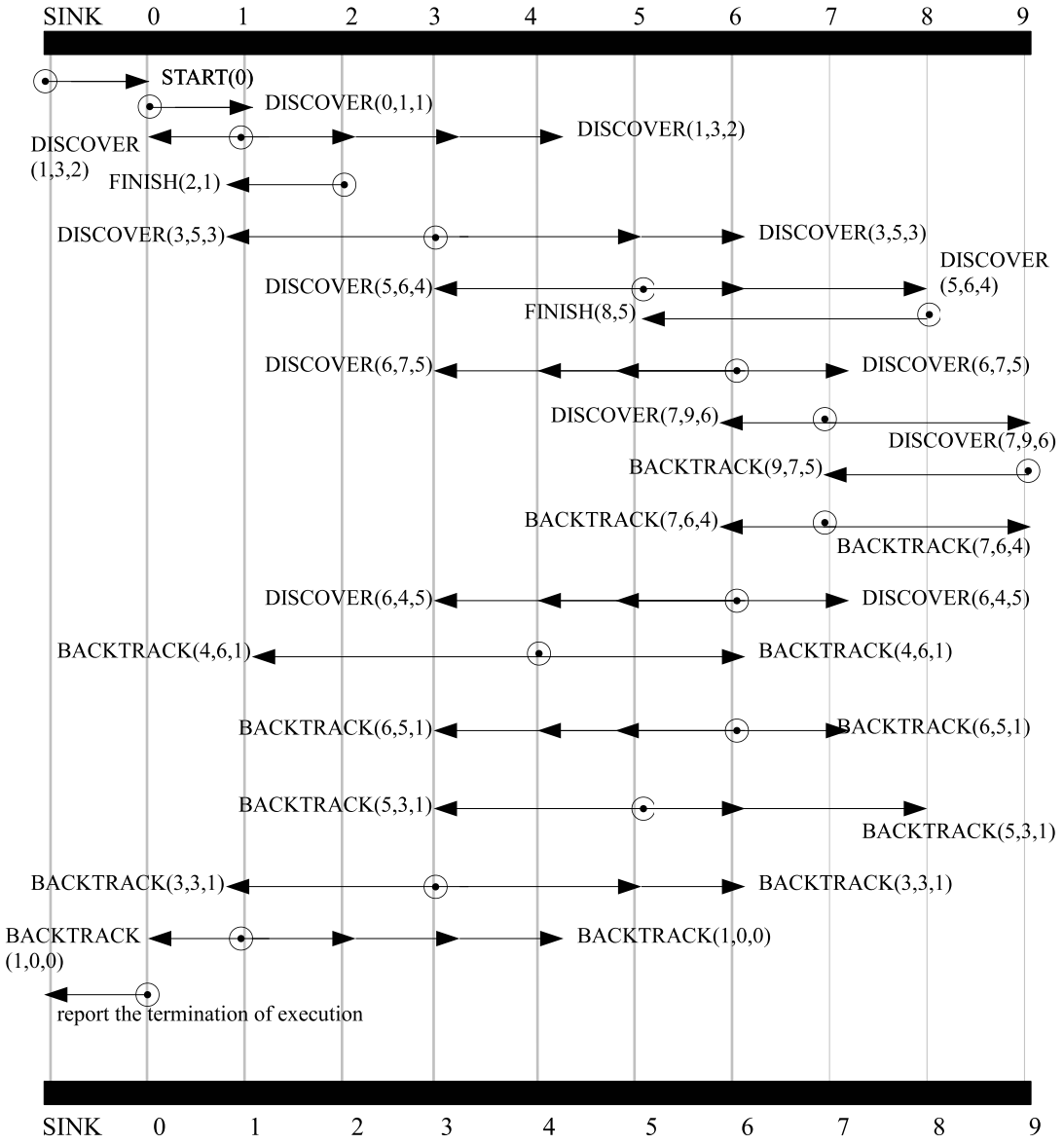

FIGURE 5. Time Diagram of the Example Operation.

full set of I, K and T please refer to Figure 6.

In our new representation, nodes start in the $IDLE$ state, depicted bye dashed lines. The $ROOT$ and $WAIT\_BACKTRACK$ states are intermediate states indicated by circles. Only the root node, which initiates the execution of the algorithm, can be in the $ROOT$ state. $WAIT\_BACKTRACK$ is the intermediate state for non-root nodes. The END and $ROOT\_END$ states are the end states and they are filled in black. The root node finishes its execution in the $ROOT\_END$ state, while other nodes finish in the END state. Initially $N{=}\Gamma$ and $C{=}\emptyset$.

Our proof of correctness can be divided into 2 parts: all nodes terminate the algorithm in end states and cut vertices are correctly detected. Both of these statements are proved in Theorem 5.1 by linking Lemmas 5.4 and 5.5. The correction of the cut vertex detection operation is proven in Lemma 5.5 by showing that Tarjan's rules are correctly applied. Lemma 5.4 proves that a node cannot terminate the algorithm in an intermediate state. Lemma 5.4 is linked by Lemmas 5.1, 5.2 and 5.3. The plan for proof of correctness is given in Fig. 7.

LEMMA 5.1. *A leaf node i overhears or receives a $DISCOVER$ message. A non-leaf node i receives a $DISCOVER$ message.*

Proof. The following cases provide a dissemination of the $DISCOVER$ messages to each node i other than the root node:

- *Case 1*: If the root node receives a $START$ message, it sends a $DISCOVER$ message to a

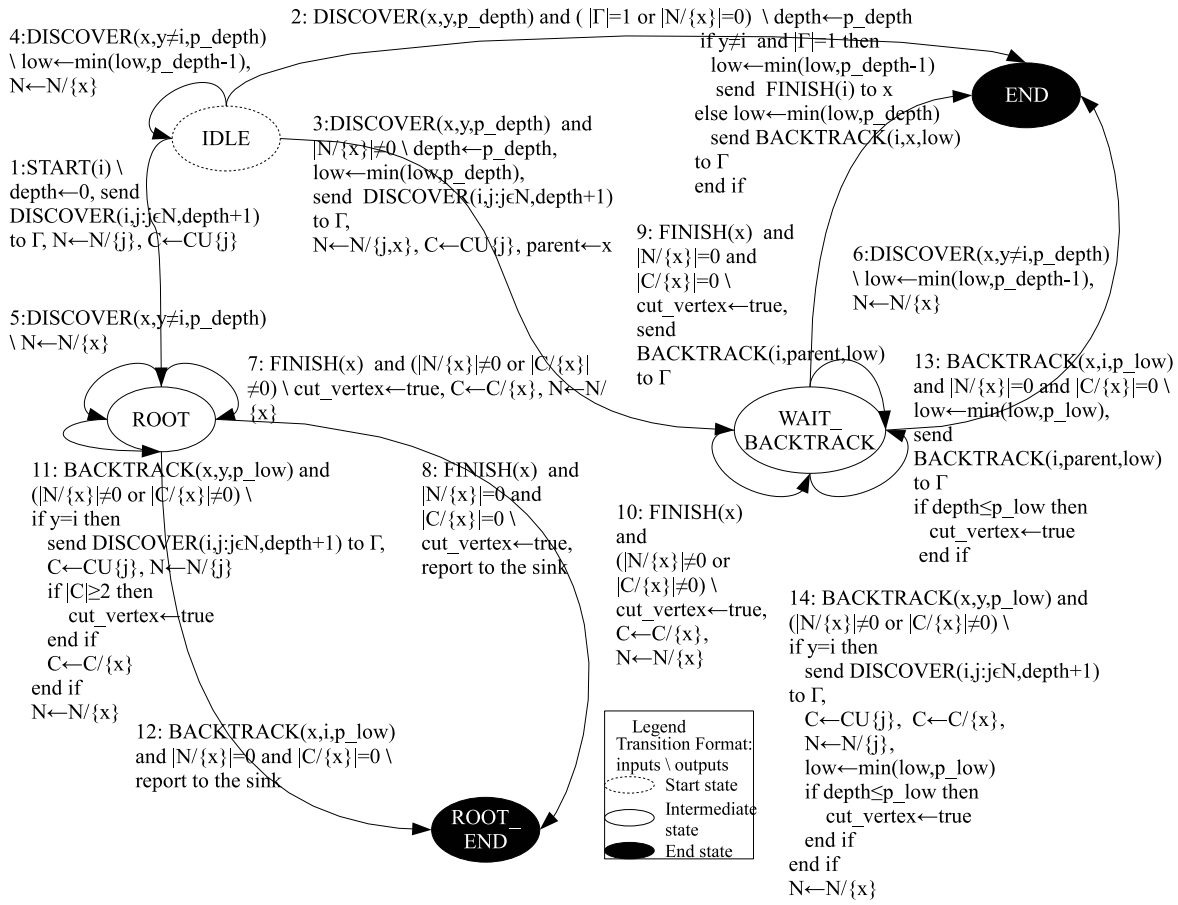

FIGURE 6. Finite State Machine of the DENCUT for node$_i$.

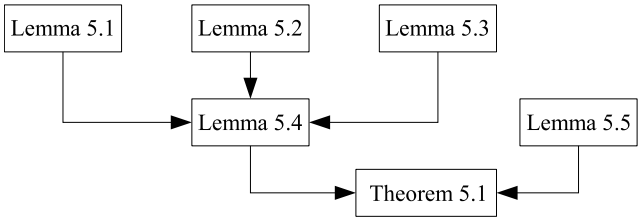

FIGURE 7. The Plan for Proof of Correctness.

node $i \in N$, as shown with Transition 1.

- *Case 2*: If leaf node i, which is in the $IDLE$ state, overhears or receives a $DISCOVER$ message then Transition 2 is applied.
- *Case 3*: If non-leaf node i, which is in the $IDLE$ state, receives a $DISCOVER$ message then Transition 3 is applied and node i sends a $DISCOVER$ message to a node $j \in N$. Thus, case 2 or case 3 can be applied for a node $j \in N$.
- *Case 4*: If the root node finds a undiscovered node i, then Transition 11 is applied, thus the root node sends a $DISCOVER$ message to a node i. Case 2 or case 3 can be applied for node i.
- *Case 5*: If a non-root node finds an undiscovered node i, then Transition 14 is applied, thus the root node sends a $DISCOVER$ message to a node i. Case 2 or case 3 can be applied for node i.

□

LEMMA 5.2. *A non-root node i sends a $BACKTRACK$ or $FINISH$ message to its parent.*

Proof. The following cases provide the transmission of $BACKTRACK$ and $FINISH$ messages to parent nodes:

- *Case 1*: If a leaf node receives or overhears a $DISCOVER$ message, it sends a $BACKTRACK$ or $FINISH$ message to its parent as given in Transition 2.
- *Case 2*: If a non-leaf and non-root node collects $BACKTRACK$, $FINISH$ and $DISCOVER$ messages from their children and other neighbors as given in Transitions 6, 9, 10, 13 and 14 then it sends a $BACKTRACK$ message to its parent as given in Transitions 9 and 13.

□

LEMMA 5.3. *The root node reports the termination of the execution to the sink.*

Proof. We assume the contrary. In this case, the root node must not receive or overhear $BACKTRACK$, $FINISH$ and $DISCOVER$ messages from its neighbors. However, executions of Transitions 5, 7, 8, 11 and 12 ensures the collection of these messages. The root node terminates the execution of the algorithm and reports to the sink node. □

Now, we can link Lemmas 5.1, 5.2 and 5.3 to prove nodes are in end states after the execution of the algorithm.

LEMMA 5.4. *After the execution of DENCUT, each node is in the ROOT_END or LEADER_END state.*

Proof. We prove by contradiction. We assume to the contrary that a node is in the $IDLE$, $ROOT$ or $WAIT\_BACKTRACK$ state after the execution of the algorithm. We investigate each of the following cases:

- *Case 1*: The node is in the $IDLE$ state. If the node is a leaf, it must not overhear or receive a $DISCOVER$ message. If the node is a non-leaf node it must not receive a $DISCOVER$ message. If any one of these events occur, the node makes a state transition and never returns to $IDLE$ state. According to Lemma 5.1, these assumptions cannot be true. A node cannot be in the $IDLE$ state after an execution of the algorithm.
- *Case 2*: The node is in the $WAIT\_BACKTRACK$ state. In this case, the node must not collect $BACKTRACK$, $DISCOVER$ or $FINISH$ messages from its neighbors. If any of these events occur, the node makes a state transition to the END state. According to Lemmas 5.1 and 5.2, this assumption cannot be true. A node cannot be in the $WAIT\_BACKTRACK$ state after an execution of the algorithm.
- *Case 3*: The root node is in the $ROOT$ state. From Lemma 5.3, the root node reports the termination of the execution to the sink, thus this assumption cannot be true.

All nodes make a state transition to the END or $LEADER\_END$ state. As shown in the finite state machine given in Fig. 6, transitions from the END and $ROOT\_END$ state are not possible. We contradict with our assumption.

□

Lemma 5.5 proves that the state transitions of the DENCUT algorithm are correct. From now on we prove that DENCUT achieves cut vertex detection.

LEMMA 5.5. *After the execution of DENCUT, all and only cut nodes set the cut_vertex=true.*

Proof. We prove by showing each case of Tarjan's algorithm [2].

- *Case 1*: Tarjan's first rule states that *low* numbers should satisfy

  $low[i]$=min($depth[i]$, min(*low* values of children), min(*depth* values of back nodes))

  Each node finds its *depth* value by executing Transitions 1, 2 and 3. Each node finds and updates its *low* values by executing Transitions 4, 6, 13 and 14.
- *Case 2*: Tarjan's second rule states that a non-root vertex i is a cut vertex if and only if one of its children j has no descendant of l with a back edge connecting it above i. In our algorithm, we investigate this rule using into two sub cases:
  - *Case 2.1*: A non-root vertex may have a leaf child. In this sub case, Transitions 9 and 10 are applied.
  - *Case 2.2*: A non-root vertex may have a non-leaf child. In this sub case, Transitions 13 and 14 are applied.
- *Case 3*: Tarjan's third rule states that the root vertex i is a cut vertex if and only if it has two or more children. In our algorithm, we investigate this rule using two sub cases:
  - *Case 3.1*: The root vertex may have a leaf child. In this sub case, Transitions 7 and 8 are applied.
  - *Case 3.2*: The root vertex may have non-leaf children. In this sub case, Transition 11 is applied.

□

To prove the correctness of the algorithm we link Lemmas 5.4 and 5.5 to prove Theorem 5.1.

THEOREM 5.1. *Each node detects its cut vertex state correctly after executing the DENCUT algorithm without deadlock and starvation.*

Proof. We assume the contrary. First, we assume that the DENCUT algorithm is not free from deadlock and starvation. In this case, a node must wait in a start or intermediate state, but it is proven in Lemma 5.4 that this case is not possible. Second, we assume that DENCUT cannot correctly detect the cut vertex state. However, it is proven in Lemma 5.5, that this case is not possible. We contradict with our assumptions. The DENCUT algorithm works and terminates correctly. □

5.2. Message Complexity

In this section, we analyze the counts of sent, received and overheard messages in the worst and best cases. We provide complexity of message size to complete the analysis of transmitted bit counts.

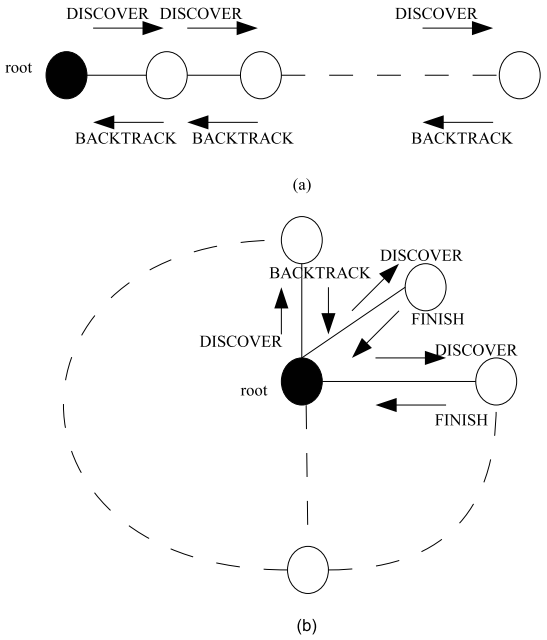

FIGURE 8. (a) Topology in Best Case (b) Topology in Worst Case.

THEOREM 5.2. *The count of sent messages in DENCUT is 2N-2 in the worst case, N in the best case.*

Proof. In the worst case, the nodes are arranged in a line topology, as shown in Fig. 8.a where N-1 node (all nodes excluding the last node in the topology) sends a $DISCOVER$ message and N-1 node (all nodes excluding the root node) sends a $BACKTRACK$ message. Thus, the total count of sent messages is $2N$-2. The best case, nodes are arranged in a star topology as shown in Figure 8.b where the root node is located in the center. In this case, the root node sends a $DISCOVER$ message to node i. At the same time, other nodes overhear a $DISCOVER$ message. Node i replies with a $BACKTRACK$ message and other nodes reply with a $FINISH$ message. Thus, the total count of sent messages is N in the best case. □

THEOREM 5.3. *The count of received and overheard messages is 2NΔ in the worst case, and 2N-1 in the best case.*

Proof. In the worst case, each node receives and overhears Δ $DISCOVER$ and Δ $BACKTRACK$ messages. Thus, the total count for N nodes is $2N\Delta$. In the best case when nodes are arranged in a star topology, each node receives and overhears only 1 $DISCOVER$ message while the root at the center receives total N-1 $BACKTRACK$ and $FINISH$ messages, such that the total count of received and overheard messages is $2N$-1 in the best case. □

THEOREM 5.4. *The message size of DENCUT is $O(log_2(N))$ bits.*

Proof. In DENCUT, the $DISCOVER$(*message_type*, *source*, *destination*, *depth*), $FINISH$(*message_type*, *source*) and $BACKTRACK$(*message_type*, *source*, *destination*, *low*) are transmitted. The longest message is $BACKTRACK$ and it includes 4 message fields represented with $log_2(N)$ bits. Thus, the message size of the DENCUT is O($log_2(N)$) bits. □

5.3. Time, Space and Computational Complexities

Finally, we work on time, space and computational complexity to fulfill the resource consumption analysis.

THEOREM 5.5. *DENCUT takes (2N-2)T time in the worst case, and 2T+S in the best case where T is the transfer time for a single message and S is the synchronization time for transmitting N-1 messages to the same destination concurrently.*

Proof. In the worst case, the nodes are arranged in a line topology, thus total operation takes $(2N\text{-}2)T$ time similar to Theorem 5.2. In the best case, the nodes are arranged in a star topology where the root node at the center sends a $DISCOVER$ message and other nodes concurrently reply. Thus, the total operation takes $2T+S$ time. □

THEOREM 5.6. *The space complexity of the DENCUT algorithm is O(c+Δ) where c is the number of variables used in the algorithm.*

Proof. Each node should store a constant number of variables (c) plus its neighbor's state where this table can be at most O($c+\Delta$). □

THEOREM 5.7. *The computational complexity per node of the DENCUT algorithm is O(Δ).*

Proof. In the worst case, a node may execute the neighbor discovery procedure for all of its neighbors, thus local computational complexity is bounded by O(Δ). □

6. PERFORMANCE EVALUATIONS

We provide extensive performance evaluations of the algorithms through testbed experiments and simulations in this section.

6.1. Experiments

First, we conducted experiments on our testbed to evaluate the performance of the proposed algorithm and its counterparts. We implemented the DENCUT, TURAU, CVD, TOKEN, HOHBERG, AHUJA and CENTRAL algorithms in nesC language on the TinyOS platform. In our testbed, we used 20 Crossbow-Memsic IRIS nodes with a 2.4 GHz IEEE 802.15.4 [28] compliant transceiver, 128 kB programmable flash memory, 8 kB RAM, 250 kpbs nominal data rate and 3 dBm transmission power. The features of our

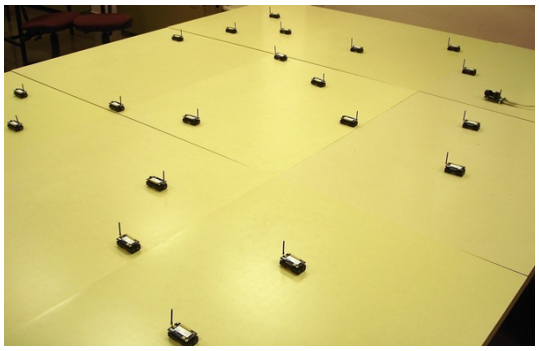

FIGURE 9. Sparse Topology.

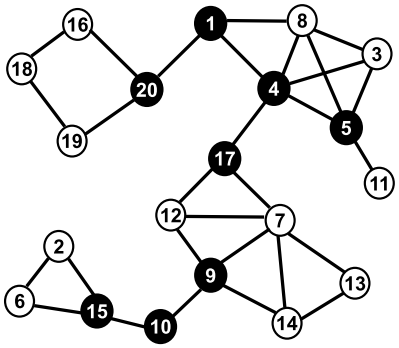

FIGURE 10. Graph of Sparse Topology.

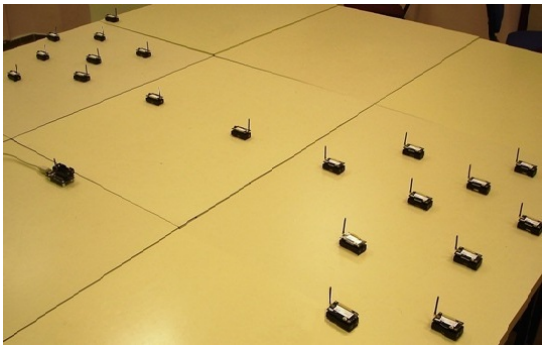

FIGURE 11. Dense Topology.

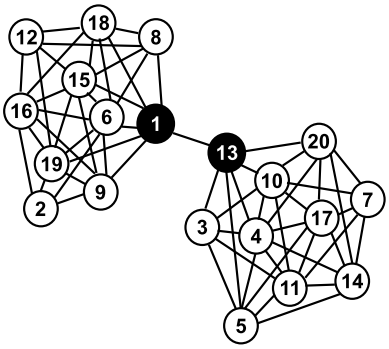

FIGURE 12. Graph of Dense Topology.

TABLE 2. Experiment Testbed Parameters.

Mote	IRIS
Platform	TinyOS
Transceiver	2.4 GHz, IEEE 802.15.4 compliant
Transmission Rate	250 kbps
Transmission Power	3 dbm
Memory	128 kB flash, 8 kB RAM
Node Count	20
Node Degree	3 and 7

testbed are summarized in Table 2. The transmission range of an IRIS mote could be adjusted by reducing transmission power and receiving sensitivity. A TDMA-based MAC protocol was implemented.

The algorithms were evaluated in sparse and dense topologies. In a sparse topology, each node had 3 neighbors and in a dense topology each node had 7 neighbors on average, which are shown with their corresponding topologies in Figs. 9, 10 11 and 12 respectively. The cut vertices in Figs. 10 and 12 are filled in black. We measured the sent byte counts, received byte counts and wallclock times of the algorithms.

The performance of the algorithms in the sparse deployment are shown in Table 3. These measurements show us that DENCUT outperformed all of the algorithms in terms of sent byte count, received byte count and wallclock time. The sent byte count of DENCUT was 24% smaller than TOKEN, 35% smaller than TURAU, 48% smaller than CVD, 51% smaller than HOHBERG, 54% smaller than CENTRAL and 55% smaller than AHUJA. The received byte count of DENCUT was 23% smaller than TOKEN, 50% smaller than TURAU, 52% smaller than CVD, 58% smaller than HOHBERG, 63% smaller than CENTRAL and 64% smaller than AHUJA. The AHUJA algorithm performs worst in terms of sent byte count, received byte count and wallclock time as it transmits a considerable number of unicast messages.

Table 4 shows the performances of the algorithms in the dense topology. As with the previous measurements, DENCUT had the best performance among all of the algorithms in terms of all metrics. The sent byte count of DENCUT was the same as in the sparse topology, approximately 1.3 times better than TOKEN, 2.9 times better than TURAU, 3.1 times better than CVD, 3.4 times better than HOHBERG, 3.5 times better than CENTRAL and 3.6 times better than AHUJA. DENCUT uses radio multicast communication and its sent byte count only depends on the node count. The received byte count of DENCUT was 17% smaller than TOKEN, 26% smaller than TURAU, 31% smaller than CVD, 39% smaller than HOHBERG, 41% smaller than AHUJA and 44% smaller than CENTRAL. CENTRAL had the worst case performance in terms of received byte count as it transmits long messages. The wallclock times of DENCUT, TOKEN, CVD and CENTRAL

TABLE 3. Performance of Algorithms in Sparse Topology.

Algorithm/ Performance	Sent bytes	Received bytes	Wallclock Times (s)
DENCUT	640	1820	35.7
TOKEN	846	2352	38.34
TURAU	990	2739	256.25
CVD	1249	3819	37.25
HOHBERG	1309	4316	271.48
CENTRAL	1391	4971	46.39
AHUJA	1416	5017	239.31

TABLE 4. Performance of Algorithms in Dense Topology.

Algorithm/ Performance	Sent bytes	Received bytes	Wallclock Times (s)
DENCUT	640	8880	33.44
TOKEN	846	10792	38.29
TURAU	1859	11986	514.418
CVD	1973	12852	35.21
HOHBERG	2192	14544	562.313
CENTRAL	2262	15819	41.82
AHUJA	2280	14982	529.17

TABLE 5. Simulation Parameters.

Node Distribution	Random
Sink Position	Randomly placed in the area
Number of Sensors	50 - 500
MAC	TDMA
Transmission Power	3 dBm
Transmission Range	50 m
Node degrees	3, 5 and 7
Leaf Percentage (%)	10 - 30

were stable against network density, while the wallclock times of TURAU, HOHBERG, AHUJA and CENTRAL significantly increased in the dense topology.

As the testbed experiments revealed, DENCUT had the best performance in terms of all metrics. The performance order in terms of sent and received byte count on average was: DENCUT, TOKEN, TURAU, CVD, HOHBERG, AHUJA and CENTRAL. When the wallclock time measurements were investigates the performance order became: DENCUT, CVD, TOKEN, CENTRAL, TURAU, AHUJA and HOHBERG. Although these measurements provide ideas about the performance of algorithms and their efficiency, the network size is limited with 20 nodes. Moreover, TinyOS does not provide an interface to measure energy consumption. To measure the performance of the algorithms for large scale networks and measure energy consumption, we use a simulator in the following section.

6.2. Simulations

We implemented the DENCUT algorithm in the TOSSIM simulator to test its performance with extensive simulations [29]. TOSSIM provides a simulation environment that is close to the real world by inheriting the TinyOS's structure. It was developed by researchers from University of California Berkeley, Intel Research Berkeley and Harvard University. To compare DENCUT with the existing work, we implemented TURAU, CVD, TOKEN, HOHBERG, AHUJA and CENTRAL algorithms.

We generated randomly connected networks with 50 to 500 nodes. The IEEE 802.15.4 radio standard was readily available in the TOSSIM simulator and chosen for the physical layer. A TDMA-based MAC protocol was implemented. The transmission power was set to 3 dBm. The average degrees of the nodes in generated networks were selected from 3, 5 to 7. We also varied leaf node percentages between 10% and 30% to show the effect of DENCUT's leaf node rule. We measured total received and total sent bytes, energy consumption and wallclock times in which each measurement was the average of 10 repeated simulations. In each simulation, nodes were randomly placed in the area. Table 5 summarizes the simulation parameters.

6.2.1. Received Bytes

First, we measured the algorithms' received byte counts. One of the most important criteria effecting energy consumption is the received byte count. Unless otherwise stated, the default node degree and count are 5 and 100, respectively.

In Fig. 13, the total received byte count of the DENCUT against the node degree and count is shown. When the node count is increased, the total received byte count of DENCUT increases linearly. These results show us that the DENCUT responds well to the increase in network connectivity and node count, as shown in Fig. 13. A comparison of the DENCUT, TURAU, CVD, TOKEN, HOHBERG, AHUJA and CENTRAL algorithms to the node count is given in Fig. 14. The total count of received bytes of the DENCUT is the smallest among the other algorithms. DENCUT is approximately 1.4 times better than TURAU, 1.7 times

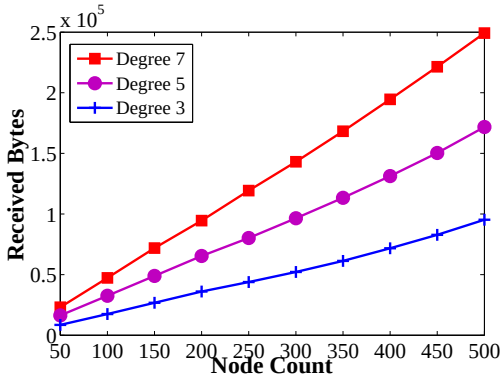

FIGURE 13. Received Bytes of DENCUT against Node Degree.

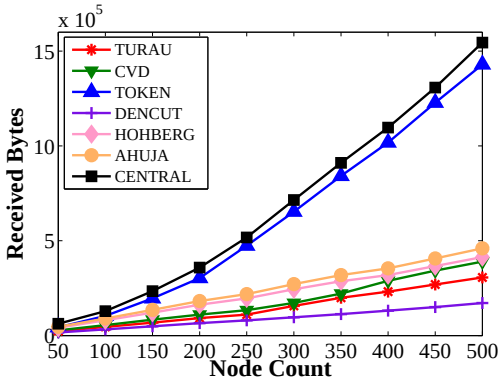

FIGURE 14. Received Bytes of Algorithms against Node Count.

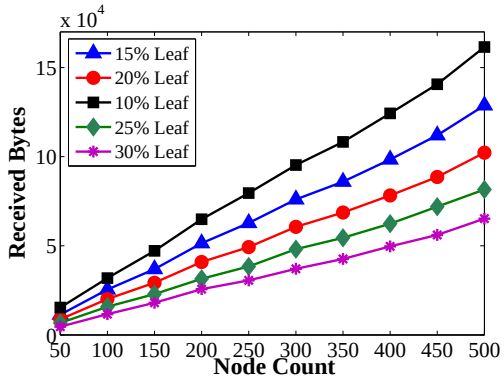

FIGURE 15. Received Bytes of DENCUT against Leaf Percentage.

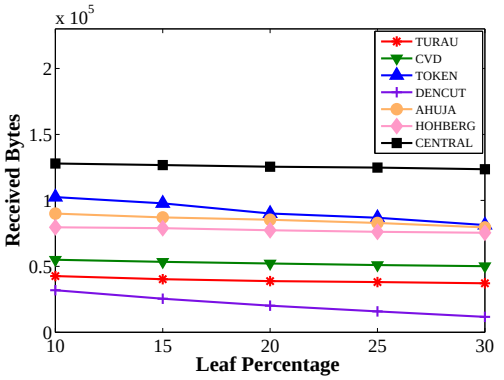

FIGURE 16. Received Bytes of Algorithms against Leaf Percentage.

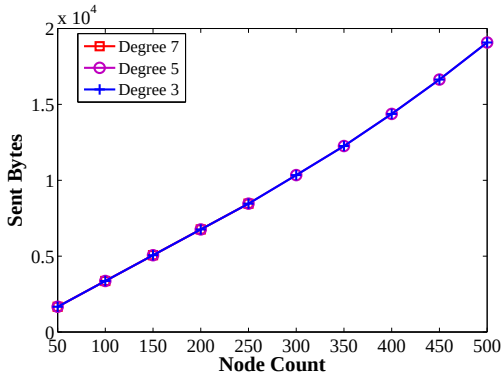

FIGURE 17. Sent Bytes of DENCUT against Node Degree.

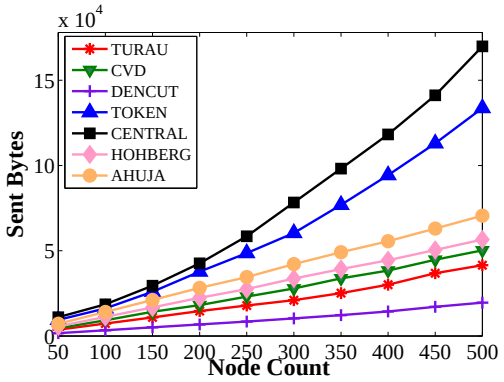

FIGURE 18. Sent Bytes of Algorithms against Node Count.

better than CVD, 2.5 times better than HOHBERG, 2.7 times better than AHUJA, 5.9 times better than TOKEN and CENTRAL algorithms on the average. It is also observed in Fig. 14 that the received byte counts of the TOKEN and CENTRAL algorithms do not increase linearly. The result of this fact is the TOKEN algorithm's message size can be as large as $O(N)$ bits and the received message count of CENTRAL algorithm is $O(\Delta DN)$.

Most topology control algorithms construct to reduce interference [30, 31]. After these types of operations are applied, the cut vertex and leaf counts may increase significantly. To investigate these cases and measure the performance of the DENCUT algorithm against various leaf percentages. A remarkable decrease in received byte count can be observed in Fig. 15, due to a leaf node transmitting a $FINISH$ message with 2 fields instead of a $BACKTRACK$ message with 4 fields. Leaf nodes shut down their radio component so that they will not overhear unrelated messages after the operation. Fig.

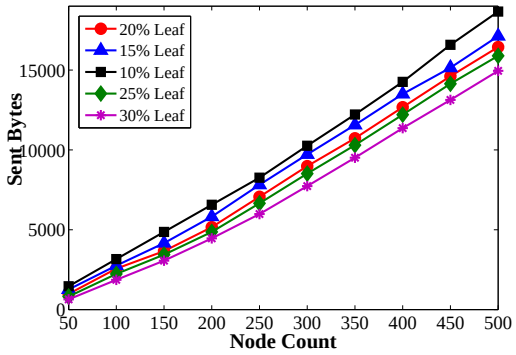

FIGURE 19. Sent Bytes of DENCUT against Leaf Percentage.

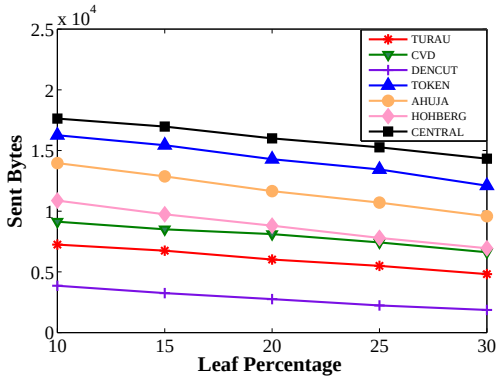

FIGURE 20. Sent Bytes of Algorithms against Leaf Percentage.

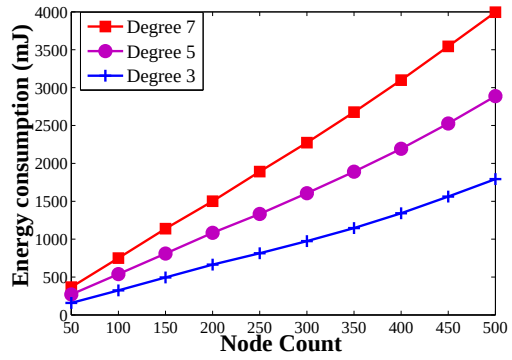

FIGURE 21. Energy Consumption of DENCUT against Node Degree.

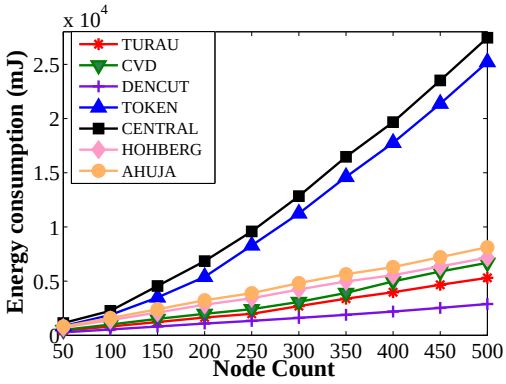

FIGURE 22. Energy Consumption of Algorithms against Node Count.

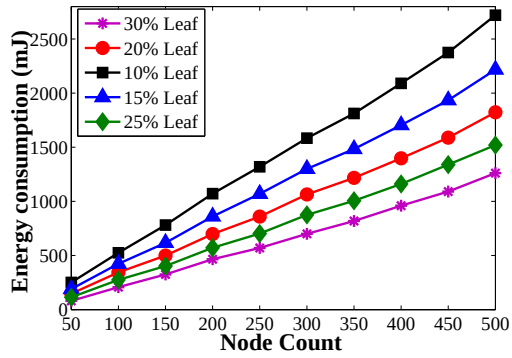

FIGURE 23. Energy Consumption of DENCUT against Leaf Percentage.

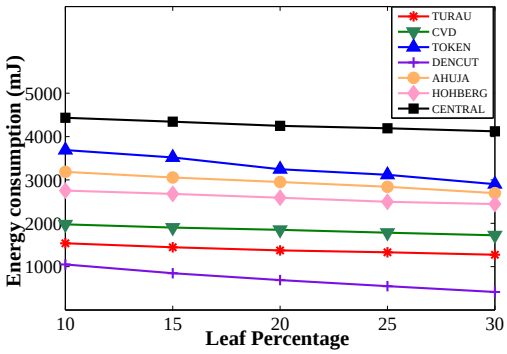

FIGURE 24. Energy Consumption of Algorithms against Leaf Percentage.

16 shows a comparison of the implemented algorithms against the leaf percentage. The total count of received bytes of the DENCUT against leaf percentage is the smallest among the other algorithms and the received byte count decreases when the leaf percentage increases. DENCUT is approximately 1.9 times better than TURAU, 2.6 times better than CVD, 3.8 times better than HOHBERG, 4.2 times better than AHUJA, 4.5 times better than TOKEN and 6.2 times better than CENTRAL. Although AHUJA, HOHBERG and TURAU are depth-first search-tree-based algorithms, DENCUT benefits from the additional rules given in this work and its performance increases with the leaf node percentage. Measurements given in this section show us that DENCUT is far better than its counterparts in terms of total received byte count.

6.2.2. Sent Bytes

Second, we measured the sent byte count of the algorithms. Similar to the received byte count, the

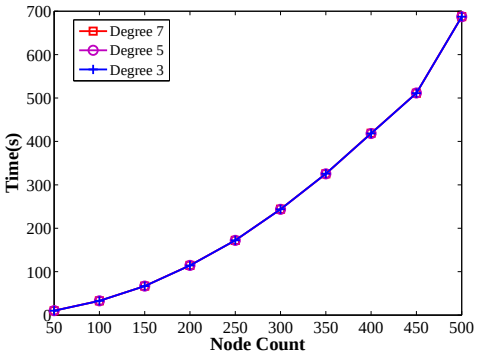

FIGURE 25. Wallclock Times of DENCUT against Node Degree.

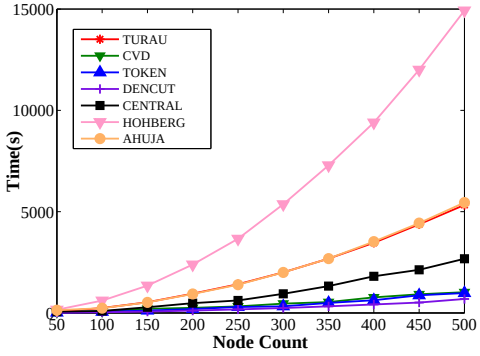

FIGURE 26. Wallclock Times of Algorithms against Node Count.

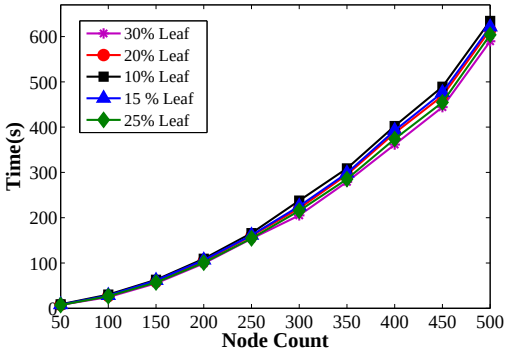

FIGURE 27. Wallclock Times of DENCUT against Leaf Percentage.

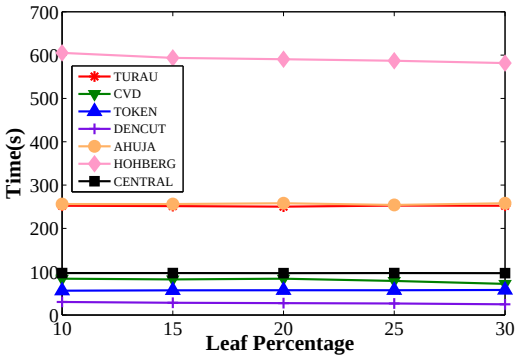

FIGURE 28. Wallclock Times of Algorithms against Leaf Percentage.

sent byte count is another important criteria for energy consumption. Our algorithm has $O(Nlog_2(N))$ sent byte complexity and it is theoretically better than other cut vertex detection algorithms.

The total sent byte count of the DENCUT against the node degree is shown in Fig. 17. The total sent byte count of the DENCUT increases linearly when the node count is increased. However, when the node degree is increased, the total sent byte count of DENCUT remains the same. This is due to the fact that node degree does not depend on sent byte complexity theoretically. A comparison of the algorithms against the node count is given in Fig. 18. CVD is better than all DFS-based algorithms except TURAU, because TURAU is a single-phased algorithm tuned for energy-efficient operation in ad hoc networks. DENCUT outperforms all algorithms, and is approximately 2.1 times better than TURAU, 2.7 times better than CVD, 3.3 times better than HOHBERG, 4.1 times better than AHUJA, 8.1 times better than TOKEN and 12 times better than CENTRAL.

The total sent byte count of the DENCUT algorithm against the leaf node percentage is shown in Fig. 19. A slight decrease in the sent byte count with the increasing leaf node percentage can be observed in Fig. 19. This decrease is due to the transmission of *FINISH* messages instead of *BACKTRACK* messages. Fig. 20 shows a comparison of the DENCUT, TURAU and CVD algorithms against the leaf percentage. The total count of sent bytes of the DENCUT against leaf percentage is the smallest among the other algorithms. DENCUT is approximately 2.4 times better than TURAU, 3.2 times better than CVD, 3.4 times better than HOHBERG, 4.6 times better than AHUJA, 6 times better than TOKEN and 6.1 times better than CENTRAL. The measurements given in this section show that DENCUT outperforms its counterparts in terms of total sent byte count.

6.2.3. Energy Consumption

Third, we measured the energy consumption of cut vertex detection algorithms. It is assumed that energy consumption occurs mostly by message transfer and depends on total count of sent and received bytes.

Fig. 21 shows the total energy consumption of the DENCUT against the node degree. The energy consumption of the DENCUT algorithm increases linearly against node degree and node count. Fig. 22 shows a comparison of the algorithms against the node count. On average, the DENCUT consumes 5.3

mJ, TURAU consumes 7.9 mJ, CVD consumes 9.6 mJ, HOHBERG consumes 13.7 mJ, AHUJA consumes 15.5 mJ, TOKEN consumes 33.1 mJ and CENTRAL consumes 35.9 mJ per node. This shows that the performance of DENCUT is at least approximately 1.5 times better than the existing work. This is a significant performance result for battery constraint networked sensor nodes to extend the lifetime of applications.

Fig. 23 shows the energy consumption of the DENCUT algorithm against the leaf node percentage. A remarkable decrease in energy consumption can be seen in Fig. 23, because both the sent and received byte counts decrease. A comparison of the algorithms against the leaf percentage is given in Fig. 24. New defined rules in DENCUT provide low energy consumption by decreasing the total count of received and sent bytes when the leaf node percentage increases. On average, DENCUT consumes 3.4 mJ, TURAU consumes 6.9 mJ, CVD consumes 9.2 mJ, HOHBERG consumes 13 mJ, AHUJA consumes 14.8 mJ, TOKEN consumes 16.2 mJ and CENTRAL consumes 21.2 mJ. The measurements given in this section show that DENCUT is far better than its counterparts in terms of energy consumption.

6.2.4. Wallclock Times

Finally, we measured the wallclock times of DENCUT and the other cut vertex detection algorithms. Fig. 25 shows the wallclock times of the DENCUT against the node degree. DENCUT is stable and scalable against the node count and degree because multicast messages are used. A comparison of the algorithms against the node count is given in Fig. 26. The performance order of the algorithms is: DENCUT, TOKEN, CVD, CENTRAL, TURAU, AHUJA and HOHBERG. DENCUT works at least 2.1 times and at most 21.5 times faster than its counterparts, on average.

The wallclock times of the DENCUT algorithm against the leaf node percentage is shown in Fig. 27. The results are stable against the leaf percentage. A comparison of the algorithms against the leaf percentage is given in Fig. 28. The performance order of the algorithms is same as in the Fig 26. In both cases, the DENCUT performs well and has a significant wallclock time performance. On average, DENCUT is 2.1 times faster than TOKEN, 3.1 times faster than CVD, 3.5 times faster than CENTRAL, 9.1 times faster than TURAU, 9.4 times faster than AHUJA and 21.5 times faster than HOHBERG.

In this section, we show that the analytical results given in Section 5 conform with the simulation results suggesting that DENCUT outperforms its counterparts in terms of energy and time consumption.

7. CONCLUSIONS

We proposed a depth-first-based energy-efficient cut vertex detection algorithm for WSNs. Our first original idea is to use the radio multicast capabilities of the sensor nodes and the overhearing method to discover links and asymptotically reduce the complexity of sent messages to $O(N)$. Our second idea is the fast-quit method of leaf nodes in which a leaf node informs its parent and suddenly terminates its local execution of the algorithm. By applying these methods, we analytically show that our algorithm is the first depth-first search based approach with with $O(N)$ time complexity and $O(N)$ sent message complexity where each message is $O(log_2(N))$ bits. To the best of our knowledge, our algorithm is the first with constant time complexity in the best case.

We showed the detailed design of the algorithm by giving its pseudocode and showed its operation by providing an example instance. We provide a finite state machine model and provide detailed proof of correctness of the DENCUT algorithm, which is implemented on real sensor nodes and in the TOSSIM simulation environment with its counterparts. We measured the received byte count, sent byte count, energy consumption and wallclock times of the algorithms against varying node degrees, node counts and node leaf percentages. In most cases, DENCUT performs well and has a significant wallclock time performance. From wallclock time measurements, DENCUT is approximately at least 2 times faster than its counterparts and provides at least 50% energy conservation as observed from energy consumption measurements. These results show that DENCUT is a significant contribution to resource-efficient cut vertex detection for fault prevention in sensor networking applications.

REFERENCES

[1] Tubaishat, M. and Madria, S. (2003) Sensor networks: an overview. Potentials, IEEE, **22**, 20–23.

[2] Tarjan, R. (1972) Depth-first search and linear graph algorithms. SIAM Journal on Computing, **1**, 146–160.

[3] Ahuja, M. and Zhu, Y. (1989) An efficient distributed algorithm for finding articulation points, bridges, and biconnected components in asynchronous networks. Proceedings of the Ninth Conference on Foundations of Software Technology and Theoretical Computer Science, London, UK, UK, pp. 99–108. Springer-Verlag.

[4] Hohberg, W. (1990) How to find biconnected components in distributed networks. J. Parallel Distrib. Comput., **9**, 374–386.

[5] Swaminathan, B. and Goldman, K. J. (1994) An incremental distributed algorithm for computing biconnected components (extended abstract). Proceedings of the 8th International Workshop on Distributed Algorithms, London, UK, UK WDAG '94, pp. 238–252. Springer-Verlag.

[6] Thurimella, R. (1995) Sub-linear distributed algorithms for sparse certificates and biconnected components. Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing, New York, NY, USA PODC '95, pp. 28–37. ACM.

[7] Chaudhuri, P. (1997) An optimal distributed algorithm for computing bridge-connected. Comput. J., **40**, 200–207.

[8] Jorgic, M., Hauspie, M., Simplot-Ryl, D., and Stojmenovic, I. (2004) Localized Algorithms for Detection of Critical Nodes and Links for Connectivity in Ad hoc Networks. Proceedings of the 3rd IFIP Mediterranean Ad Hoc Networking Workshop, Turquie 12.

[9] Turau, V. (2006) Computing bridges, articulations, and 2-connected components in wireless sensor networks. Proceedings of the Second international conference on Algorithmic Aspects of Wireless Sensor Networks, Berlin, Heidelberg ALGOSENSORS'06, pp. 164–175. Springer-Verlag.

[10] Akkaya, K., Senel, F., Thimmapuram, A., and Uludag, S. (2010) Distributed recovery from network partitioning in movable sensor/actor networks via controlled mobility. IEEE Transactions on Computers, **59**, 258–271.

[11] He, Y., Ren, H., Liu, Y., and Yang, B. (2009) On the reliability of large-scale distributed systems - a topological view. Comput. Netw., **53**, 2140–2152.

[12] Xiong, S. and Li, J. (2010) An efficient algorithm for cut vertex detection in wireless sensor networks. Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems, Washington, DC, USA ICDCS '10, pp. 368–377. IEEE Computer Society.

[13] Won, M. and Stoleru, R. (2011) Destination-based cut detection in wireless sensor networks. Embedded and Ubiquitous Computing, IEEE/IFIP International Conference on, **0**, 55–62.

[14] Stojmenovic, I., Simplot-Ryl, D., and Nayak, A. (2011) Toward scalable cut vertex and link detection with applications in wireless ad hoc networks. Netwrk. Mag. of Global Internetwkg., **25**, 44–48.

[15] Barooah, P., Chenji, H., Stoleru, R., and Kalmar-Nagy, T. (2012) Cut detection in wireless sensor networks. IEEE Trans. Parallel Distrib. Syst., **23**, 483–490.

[16] Liu, X., Xiao, L., and Kreling, A. (2012) A fully distributed method to detect and reduce cut vertices in large-scale overlay networks. IEEE Trans. Comput., **61**, 969–985.

[17] Imran, M., Younis, M., Said, A. M., and Hasbullah, H. (2010) Partitioning detection and connectivity restoration algorithm for wireless sensor and actor networks. Proceedings of the 2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, Washington, DC, USA EUC '10, pp. 200–207. IEEE Computer Society.

[18] Wang, S., Mao, X., Tang, S.-J., Li, X.-Y., Zhao, J., and Dai, G. (2011) On movement-assisted connectivity restoration in wireless sensor and actor networks. IEEE Transactions on Parallel and Distributed Systems, **22**, 687–694.

[19] Dagdeviren, O. and Erciyes, K. (2010) Graph matching-based distributed clustering and backbone formation algorithms for sensor networks. Comput. J., **53**, 1553–1575.

[20] Dagdeviren, O. and Khalilpour Akram, V. (2013) Energy-efficient bridge detection algorithms for wireless sensor networks. International Journal of Distributed Sensor Networks, **2013**.

[21] Akram, V. K. and Dagdeviren, O. (2013) Breadth-first search-based single-phase algorithms for bridge detection in wireless sensor networks. Sensors, **13**, 8786–8813.

[22] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001) Introduction to algorithms, 2. edition. McGraw-Hill Book Company, Cambridge, London.

[23] Stojmenovic, I., Seddigh, M., and Zunic, J. (2002) Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks. IEEE Trans. Parallel Distrib. Syst., **13**, 14–25.

[24] Dai, F. and Wu, J. (2005) On constructing k-connected k-dominating set in wireless networks. In Proceedings of the 19 th International Parallel and Distributed Processing Symposium (IPDPS).

[25] Cokuslu, D., Erciyes, K., and Dagdeviren, O. (2006) A dominating set based clustering algorithm for mobile ad hoc networks. in Proc. ICCS2006, LNCS 3991, pp. 571–578. Springer-Verlag.

[26] Yuanyuan, Z., Jia, X., and Yanxiang, H. (2006) Energy efficient distributed connected dominating sets construction in wireless sensor networks. Proceedings of the 2006 international conference on Wireless communications and mobile computing, New York, NY, USA IWCMC '06, pp. 797–802. ACM.

[27] Raei, H., Tabibzadeh, M., Ahmadipoor, B., and Saei, S. (2009) A self-stabilizing distributed algorithm for minimum connected dominating sets in wireless sensor networks with different transmission ranges. Proceedings of the 11th international conference on Advanced Communication Technology - Volume 1, Piscataway, NJ, USA ICACT'09, pp. 526–530. IEEE Press.

[28] Gutierrez, J. A., Callaway, E. H., and Barrett, R. L. (2007) Low-Rate Wireless Personal Area Networks: Enabling Wireless Sensors with IEEE 802.15.4. Standards Information Network IEEE Press, New York, NY, USA.

[29] Levis, P., Lee, N., Welsh, M., and Culler, D. (2003) Tossim: Accurate and scalable simulation of entire tinyos applications. Proceedings of the 1st int. conf. on Embedded networked sensor systems, New York, NY, USA SenSys '03, pp. 126–137. ACM.

[30] Rickenbach, P. V., Wattenhofer, R., and Zollinger, A. (2009) Algorithmic models of interference in wireless ad hoc and sensor networks. IEEE/ACM Trans. Netw., **17**, 172–185.

[31] Yilmaz, O., Dagdeviren, O., and Erciyes, K. (2011) Interference-aware dynamic algorithms for energy-efficient topology control in wireless ad hoc and sensor networks. Comput. J., **54**, 1398–1411.