Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2013, Article ID 867903, 15 pages
http://dx.doi.org/10.1155/2013/867903

Research Article

Energy-Efficient Bridge Detection Algorithms for Wireless Sensor Networks

Orhan Dagdeviren and Vahid Khalilpour Akram

International Computer Institute Bornova, Ege University, 35100 Izmir, Turkey

Correspondence should be addressed to Orhan Dagdeviren; orhan.dagdeviren@ege.edu.tr

Received 27 January 2013; Accepted 8 April 2013

Academic Editor: Hongju Cheng

Copyright © 2013 O. Dagdeviren and V. K. Akram. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

A bridge is a critical edge whose fault disables the data delivery of a WSN component. Because of this, it is important to detect bridges and take preventions before they are corrupted. Since WSNs are battery powered, protocols running on WSN should be energy efficient. In this paper, we propose two distributed energy-efficient bridge detection algorithms for WSNs. The first algorithm is the improved version of Pritchard's algorithm where two phases are merged into a single phase and radio broadcast communication is used instead of unicast in order to remove a downcast operation and remove extra message headers. The second algorithm runs proposed rules on 2-hop neighborhoods of each node and tries to detect all bridges in a breadth-first search (BFS) execution session using $O(N)$ messages with $O(\Delta(\log_2(N)))$ bits where N is the node count and Δ is the maximum node degree. Since BFS is a natural routing algorithm for WSNs, the second algorithm achieves both routing and bridge detections. If the second proposed algorithm is not able to to classify all edges within the BFS phase, improved version of Turau's algorithm is executed as the second phase. We show the operation of the algorithms, analyze them, and provide extensive simulation results on TOSSIM environment. We compare our proposed algorithms with the other bridge detection algorithms and show that our proposed algorithms provide less resource consumption. The energy saving of our algorithms is up to 4.3 times, while it takes less time in most of the situations.

1. Introduction

Rapid developments in the last decade in wireless and hardware technologies have created low-cost, low-power multifunctional miniature wireless devices. These devices have enabled the use of wireless sensor networks (WSNs) [1]. WSNs do not have any fixed infrastructure where hundreds even thousands of sensor nodes cooperate to implement a distributed application. WSNs can be used in various applications including habitat monitoring [1], military [2], and smart home applications [3]. Energy consumption of a WSN should be reduced to maximize the application lifetime since sensor nodes are battery powered. The radio component of a sensor node is the dominant energy consumer part.

WSNs are increasingly being used in challenged environments such as underground mines, tunnels, oceans, and the outer space. Wireless communication in challenged environments have channel (edge) failures, mainly as a consequence of direct impact of physical world. In addition to energy constraints and wireless communication problems, tiny sensor motes are prone to failures. In both type of these failures, sensor network can continue its operation without a serious bad effect. On the other side, some edges can have critical tasks in routing operation. These edges are called bridges (cut edges) which its removal breaks connectivity of the network. Bridge detection is an important research area for different types of networks [4–9]. After bridges are detected, various solutions can be applied [10] in order to neutralize bridges.

A WSN can be modeled with an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. BFS and depth-first search (DFS) are fundamental graph traversal algorithms. DFS starts from sink node, and searches deeper in the graph if possible [11]. Like DFS, BFS starts from sink node and search proceeds in a breadth-first manner [11]. The edges chosen for BFS are called tree edges, and other edges are called cross edges. In BFS, tree level of sink node is 0, the levels of neighbors of sink node are 1, and levels of other

nodes are their shortest distance to the sink node. From this property, BFS can be used to construct shortest path trees rooted at the sink node. BFS is used widely in sensor network for various purposes such routing and localization [12–16]. Besides, BFS can be modified to detect bridges. In this study, we proposed bridge detection algorithms that use BFS as the basis algorithm.

Distributed bridge detection algorithms proposed so far have some important disadvantages. Although distributed DFS based algorithms [8, 17–20] are simple and efficient for bridge detection, DFS based applications for WSN are rare in practice. Since then, DFS should be implemented as a separate service where this would be an extra load for battery-powered sensor nodes. Although BFS provides an efficient routing infrastructure for sensor networks, BFS based bridge detection algorithms lack some important design criterions. The message size of the Milic's BFS based algorithm [9] can be as large as $O(E \log_2(N))$. Pritchard's BFS based algorithm has two extra phases [21]. Because of these reasons, these algorithms may consume significant energy. Regarding these deficiencies, we propose two distributed localization-free and energy-efficient bridge detection algorithms for sensor networks. The contributions of this paper are listed below.

(i) We propose an improved version of Pritchard's algorithm (I-PRITCHARD). In I-PRITCHARD algorithm, radio broadcast communication is used instead of unicast communication, and a downcast operation in Pritchard's algorithm is removed. I-PRITCHARD can be completed within 2 phases; on the other hand, Pritchard's algorithm needs 3 phases. Because of these reasons, I-PRITCHARD consumes less energy than the Pritchard's algorithm.

(ii) We propose the energy-efficient bridge detection algorithm (ENBRIDGE) by using 2-hop neighborhood knowledge and radio broadcast communication. The algorithm uses $O(N)$ messages with $O(\Delta \log_2(N))$ bits, where N is the node count and Δ is the maximum node degree. Besides, the algorithm runs just one phase, that is, integrated with the BFS at the best case. This is a significant improvement over Milic's algorithm. In the worst case, the algorithm runs an improved DFS algorithm where message complexity and message size are asymptotically same with the first phase. ENBRIDGE outperforms its counterparts in the simulations.

The rest of this paper is organized as follows. In Section 2, the network model and bridge detection problem are described, and the related work is surveyed in Section 3. The proposed algorithms are described in Section 4. The detailed analysis of the algorithms are given in Section 5, and the results of performance tests are presented in Section 6. Conclusions are given in Section 7.

2. Background

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

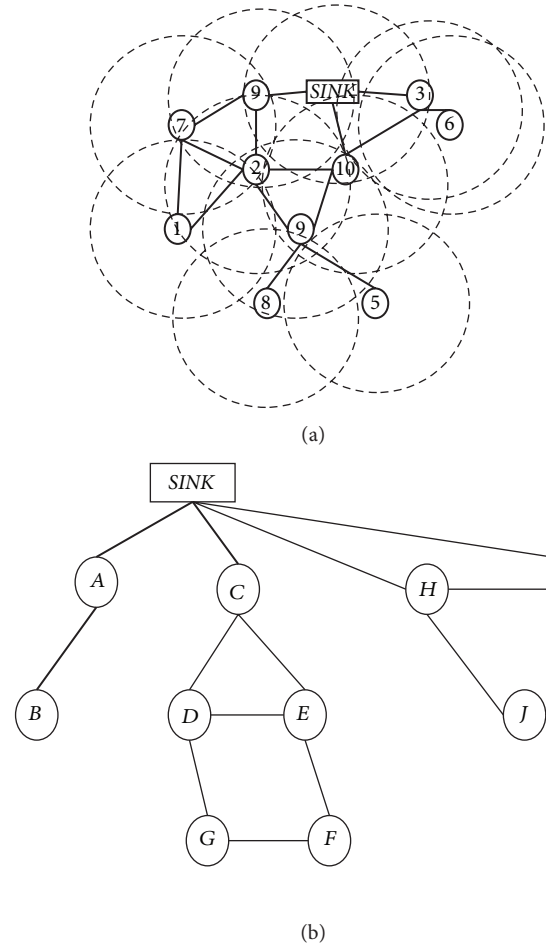

Figure 1: (a) Undirected graph model. (b) The bridge problem.

2.1. Network Model. The following assumptions are made about the network as in [8, 22].

(i) Each node has distinct *node_id*.

(ii) The nodes are stationary.

(iii) Links between nodes are symmetric. Thus, if there is a link from u to v, there exists a reverse link from v to u.

(iv) Nodes do not know their positions. They are not equipped with a position tracker like a GPS receiver.

(v) All nodes are equal in terms of processing capabilities, radio, battery, and memory.

(vi) Each node can send radio broadcast messages to its immediate neighbors in its transmission range.

(vii) Nodes are time synchronized. This can be achieved by implementing a time synchronization protocol as proposed in [23].

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 of undirected graph model is depicted in Figure 1(a), where the

transmission ranges of the nodes are shown with dashed circles.

2.2. The Bridge Detection Problem. Bridges can connect any two nodes on the network. A bridge can connect a leaf node to its parent or connect a whole component to lower layers. An example of sensor network is depicted in Figure 1(b). There are 10 nodes, where node ids are written inside of each node, and the communication edges are shown with the solid lines in this Figure. The edges (*A*, *B*), (*A*, *SINK*), and (*C*, *SINK*) are bridges which are depicted with bold lines. If the edge (*A*, *B*) fails then leaf node *B* cannot transmit its packet to the lower layers. If the edge (*A*, *SINK*) fails then both nodes *A* and node *B* cannot relay their data packets where 20% of the network can not transmit its data to the sink node. Node *C* is the parent of a component consisting of the node *D*, the node *E*, the node *G*, and the node *F*. If the edge (*C*, *SINK*) fails, 50% of the total nodes cannot send their data to the sink node. In this paper, our focus is energy-efficient bridge detection for sensor networks, so our objectives are listed below.

(1) Since message transmission is the dominant factor of energy consumption, the bridge detection algorithm should be efficient in terms of message complexity and message size.

(2) Routing is a fundamental operation for sensor networks. It is crucial for data delivery and data aggregation [24]. If the bridge detection can be integrated with the routing operation, it can introduce a less total cost to the network.

(3) Sink node may initiate the bridge detection algorithm locally. Hence, these operations should be distributed.

(4) Bridge detection algorithm should be independent from the underlying protocols as much as possible to interface to various MAC and physical layer standards such as in [25–30].

(5) The algorithm should not be dependent on localization information.

3. Related Work

DFS algorithm can be centrally executed by the sink node in order to detect bridges [4, 9]. Since collecting the whole network information is expensive and not always possible, various distributed implementations are proposed for DFS algorithm [17, 20, 31–36]. Most of these algorithms can be modified to detect bridge in sensor networks by using the rules proposed by Tarjan [4]. Cidon [17], Hohberg [18], Chaudhuri [19], Tsin [20], and Turau [8] proposed distributed DFS algorithms for bridge and cut vertex (a vertex whose removal breaks the connectivity of a graph) detection algorithms. Turau's algorithm [8] is an extended and sensor network adopted version of Cidon's [17] and Tsin's [20] algorithms. At the worst case, Turau's algorithm transmits $4E$ messages with $O(\log_2(N))$ size, where E is the number of edges and N is the total node count. Since unicast messages are used in Turau's algorithm, if the medium access control (MAC) layer does not provide an edge based sleep schedule, the upper bound for the received and overhead messages is $O(\Delta E)$. Our algorithms have $\Theta(N)$ sent message complexity, and $O(\Delta N)$ received and overheard message complexity. Also we show in this paper with extensive simulations that our algorithms are practically favorable.

Like DFS, BFS can be centrally executed to detect bridges [9]. Although this algorithm is simple to implement, execution of central BFS is an expensive operation in terms of energy consumption caused by message transfers, and it is not suitable for large scale self-organizing distributed sensor networks. Because of these reasons, distributed BFS algorithms are proposed [12–16]. For synchronous networks, a well-known greedy algorithm is applied to construct BFS [14]. This algorithm consumes $O(N)$ messages and $O(D)$ time, where D is the diameter of the network. Although this algorithm is very effective for constructing routing infrastructure, it is not adequate to find bridges without any extension. Liang proposed a BFS algorithm for biconnectivity testing algorithm which runs on permutation graphs and which cannot be generalized [5]. Thurimella proposed a BFS biconnectivity testing algorithm in which each processor is assumed to know the whole topology [6]. Because of this property, algorithm is not suitable for WSNs.

Milic proposed a bridge detection for wireless ad hoc networks [9]. The algorithm uses broadcast communication of wireless nodes, and it is integrated with the BFS operation. The forward phase of the algorithm is nearly the same with the standard BFS operation. In the backward phase of the algorithm, the nodes store a list of cross edges that they found or received, append cross edges to the messages, and send to their parents. Although the algorithm completes its operation within a BFS execution interval, the message size can be very large since it is dependent on the number of cross edges. The message size can be as large as $O(E \log_2(N))$. On the other side, our proposed algorithms' message size are $O(\Delta \log_2(N))$. Besides, in this paper, we simulate Milic's algorithm against various network topologies and show that our proposed algorithms are favorable.

Pritchard proposed a three-phased algorithm for the distributed bridge detection [21]. In the first step, the algorithm finds a spanning tree by implementing distributed greedy BFS tree algorithm. In the second step, the algorithm computes subtree sizes and preorder labels. In the last step, bridges are detected. The time complexity of the algorithm is $O(D)$, the message complexity is $O(E)$, and the message size is $O(\log_2(N))$ bits. In this paper, we first propose an improved version of this algorithm. Secondly, we propose an energy-efficient bridge detection algorithm which can be integrated with the BFS operation and can finish within the BFS execution. The algorithms covered so far exactly find bridges without localization. In this study, we omit localization-based bridge detection algorithms such as [16].

4. Proposed Algorithms

4.1. Improved PRITCHARD Algorithm. PRITCHARD algorithm effectively detects bridges of an undirected graph

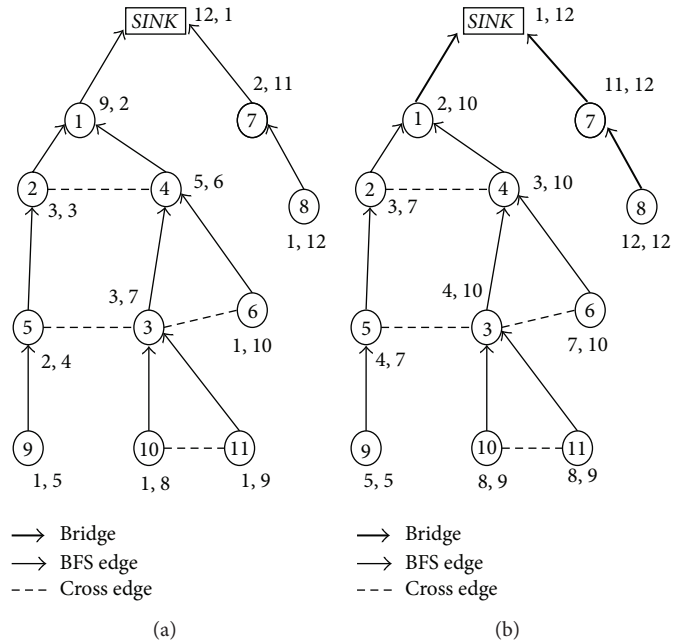

Figure 2: (a) I-PRITCHARD Phase 1 (b) I-PRITCHARD Phase 2.

by applying a 3 phase method. Although the algorithm is well designed, it can be further improved for battery-powered sensor nodes. To achieve this, we propose the I-PRITCHARD algorithm which includes the following list of modifications in order to reduce transmitted message counts and transmitted bit counts.

(1) Phase 1 and Phase 2 can be merged into a single phase. When the nodes are executing backward operation during BFS execution in Phase 1, each node may calculate its subtree size (*#desc*), and the convergecast operation in Phase 1 can be accomplished. To achieve this, each message in backward phase should include *#desc* field which is $O(\log_2(N))$ bits size. After backward operation is finished, Phase 2 is not executed, so that $O(N)$ messages which are flooded by the initiator node are saved by applying this improvement.

(2) Preorder messages in Phase 2 and announcement messages in Phase 3 are sent as broadcast messages instead of unicast. In this case, although the message size increases to $O(\Delta\log_2(N))$ bits, header fields for separate messages are not transmitted. So total transmitted bit counts are reduced.

An example operation is given in Figure 2. Since BFS execution is integrated with the preorder labeling in I-PRITCHARD, the algorithm is executed in 2 phases. The first phase is depicted in Figure 2(a). In this figure, ID of the nodes is written inside of each node, *#desc* value and preorder label are written near to each node. The second phase is depicted in Figure 2(b). In this figure, low and high values are written near to each node. The edges (1, *SINK*), (7, *SINK*), (5, 9), and (7, 8) are the bridges.

4.2. ENBRIDGE Algorithm. The algorithms covered so far have below listed deficiencies which motivate us to design ENBRIDGE algorithm.

(i) Even though the design of an energy-efficient DFS based bridge detection algorithm is possible, DFS applications are rare in real-world applications. Because of that, the DFS based bridge detection module may not be integrated to the other applications.

(ii) Although MILIC may be easily integrated to the BFS, transmitted bit count is proportional to the network diameter and cross edge count.

(iii) PRITCHARD and I-PRITCHARD are BFS based algorithms, transmitted bit counts are proportional to the average neighbor degree. Although these algorithms are energy-efficient, extra phases are executed after the BFS algorithm.

We propose ENBRIDGE algorithm for detecting bridges in WSN in an energy-efficient manner. The algorithm has two phases. In the first phase, an extended BFS algorithm is executed. The forward phase of the BFS algorithm is the same. At the backward phase of the BFS algorithm, each node broadcasts its edge states to its neighbors where a state of an edge can be one of *CHILD*, *CROSS*, and *BRIDGE*. Hence, each node knows the edge states of its neighbors. By using these 2-hop information, each node $n \in V$ runs the ENBRIDGE_Classify given in Algorithm 1 to check whether its edge connecting to the parent node p is bridge. Each rule given in Algorithm 1 is executed sequentially by the nodes since they are ordered by considering their computational complexity. The computational complexity of the first three rules is $O(\Delta)$, and the computational complexity of the last rule is $O(\Delta^3)$.

(1) // Algorithm inputs 2-hop neighbors of node n.
if the node n has only 1 incident edge **then** the edge (n, p) is a bridge (*Rule 1*).
(2) **else if** the node n has at least one cross edge **then** the edge (n, p) is not a bridge (*Rule 2*).
(3) **else if** the edge (n, p) is the only edge connecting node n to lower levels and all other edges are bridges **then** the edge (n, p) is a bridge (*Rule 3*).
(4) **else if** one of node n's children has a cross edge connecting to node x where node x's parent is not n and id of one of node n's children **then** the edge (n, p) is not a bridge (*Rule 4*).
(5) **else if** all neighbors of node n's children are also children of node n **then** the edge (n, p) is a bridge (*Rule 5*).
(6) **end if**.
(7) **if** one of these rules are applied **then** return true.
(8) **else** return false.
(9) **end if**.

Algorithm 1: ENBRIDGE BFS edge classification algorithm (ENBRIDGE_Classify).

After executing these rules at the backwards stage of the first phase, each node notifies whether it is able to classify its parent link. To achieve this notification, a 1-bit *classified* field is transmitted during convergecast operation in backwards stage. If the node n or one of descendants cannot classify its parent link, *classified* field gets 0, otherwise it gets 1. Inductively, sink node finds whether one of the BFS edges is left unclassified. If the node n sends classified as false, then its parent node p does not execute ENBRIDGE_Classify as given in Algorithm 2 in order to save CPU power. If all BFS edges are classified, then sink node stops the execution of the algorithm. Otherwise, the sink node starts the second phase of the ENBRIDGE algorithm.

Although the second phase of the ENBRIDGE algorithm is not always executed, it should be energy-efficient as the first phase. For the second phase, we propose an improved version of the TURAU's DFS based algorithm (I-TURAU). In this improved version, *SEARCH* messages are sent as broadcast messages instead of unicast messages so *VISITED* messages no longer need to be transmitted. With this improvement, $O(N)$ messages with $O(\log_2(N))$ bits are transmitted during the second phase of the ENBRIDGE algorithm. The second phase can classify all edges in all situations where the first phase cannot. From this fact, one can claim that the execution of the first phase is redundant. Although this claim can be true for sensor networks which do not use BFS like routing infrastructures, this claim is false for sensor networks where BFS is the dominant routing protocol. The detailed ENBRIDGE algorithm is given in Algorithm 2.

Example operations of ENBRIDGE are depicted in Figure 3. In the first example shown in Figure 3(a), all edges can be classified by the first phase of the algorithm. Edges (10, 12) and (6, 11) are classified as bridges by executing Rule 1. Edges (1, 3), (2, 4), (4, 6), (5, 7), (5, 8), and (5, 9) are classified as nonbridges by executing Rule 2. Edge (6, 10) is classified as bridge by executing Rule 3. Edges (1, *SINK*) and (2, *SINK*) are classified as nonbridges by executing Rule 4. Edge (3, 5) is classified as bridge by executing Rule 5. Since all edges can be classified, the second phase is not executed, and the ENBRIDGE algorithm is finished. In the second example given in Figure 3(b), although edges (1, 4), (2, 5), (4, 6), and (5, 10) can be classified as nonbridges, (1, 2) and (1, *SINK*) can not be classified. Thus, the second phase is executed. In the second phase, (1, 2) is classified as nonbridge, and (1, *SINK*) is classified as bridge edge.

5. Analysis

In this section, we will analyze proof of correctness, message, time, space, and computational complexities of the I-PRITCHARD and ENBRIDGE algorithms.

5.1. Proof of Correctness

Theorem 1. *Nodes executing I-PRITCHARD detect bridges and terminate the execution correctly.*

Proof. Correctness of merging Phase 1 and Phase 2 into a single phase can be proved by induction. Since the BFS execution is synchronous, each leaf node can calculate its *#desc* as 1 during backward phase of BFS as the base case of the induction. Each nonleaf node can calculate its *#desc* by aggregating its children's *#desc*, where this operation is continued until the sink node's execution inductively. The proof of correctness of using broadcast instead of unicast is trivial since the same information is received by all nodes. □

Theorem 2. *Each node detects its parent link state correctly after executing the ENBRIDGE algorithm.*

Proof. We first prove the correctness of the ENBRIDGE_Classify algorithm. To prove the correctness of Rule 1, we assume that the node n has only 1 incident edge. In this case, the node n is a leaf; thus, (n, p) is a bridge. To prove the correctness of Rule 2, we assume that the node n has a cross edge e, and then e can be one of followings.

(i) The edge e equals (n, x) where the node x's level (level_x) is smaller then or equal to the level_n. In both of this cases, the edge (n, p) is not a bridge since the node n has an incident edge other than (n, p) which connects the node n to lower layers.

(ii) The edge e equals (n, x) where the level_x is greater than level_n. Since the node x is not a child of the node

```
(1) message formats: FORWARD(source, level, parent), BACKWARD(source, destination, edges, classified)
(2) initially: id is the unique node identifier; parent ← ∞; children ← ∅; crosses ← ∅;
    neighbors is the set of collected sources of HELLO messages; forward_sent ← false;
    forward_received ← false; two_hop_edges ← ∞; classified ← true
(3) sink node multicasts FORWARD(0, 0, ∞) to neighbors
(4) upon a node receives FORWARD(source, p_level, p_parent) message
(5)   forward_received ← true
(6)   if parent = ∞ then
(7)     parent ← id; level ← p_level + 1; assign the edge (source, id) as a BFS edge
(8)   else if p_parent = id then
(9)     children ← children ∪ source; assign the edge (source, id) as a BFS edge
(10)  else assign the edge (source, id) as a cross edge; crosses ← crosses ∪ source
(11)  end if
(12) end upon
(13) upon a new period starts
(14)  if forward_sent = false ∧ forward_received = true then
(15)    forward_sent ← true; multicast FORWARD(id, level, parent) to neighbors
(16)    if neighbors = children ∪ crosses ∪ {parent} then
(17)      classified ← ENBRIDGE_Classify(assigned edges)
(18)      send BACKWARD(id, parent, assigned edges, classified) to parent
(19)    end if
(20)  else if ∀ BACKWARD messages from i ∈ children received then
(21)    two_hop_edges ← two_hop_edges ∪ assigned edges
(22)    if classified = true then
(23)      classified ← ENBRIDGE_Classify(two_hop_edges)
(24)    end if
(25)    if id = sink then
(26)      if classified = true then finish execution
(27)      else start I-TURAU execution
(28)      end if
(29)    else
(30)      send BACKWARD(id, parent, assigned edges, classified) to parent
(31)    end if
(32) end upon
(33) upon a node receives BACKWARD(source, destination, edges, p_classified)
(34)  two_hop_edges ← two_hop_edges ∪ edges; classified ← classified ∧ p_classified
(35) end upon
```

Algorithm 2: ENBRIDGE main algorithm.

n, then the $\text{level}_{\text{parent}_x}$ can be at most level_n. Thus, same as in previous case, (n, p) is not a bridge.

To prove the correctness of Rule 3, we assume that the node n does not have any cross edge; the edge (n, p) is the only edge connecting node n to lower layers, and all other edges are bridges. In this case, the node n does not have any alternative path connecting it to lower layers which excludes the edge (n, p), so the edge (n, p) is a bridge. To prove the correctness of Rule 4, we assume that one of the node n's children (node c) has a cross edge connecting to node x, where $parent_x$ is not equal to the n and one of node n's children. In this case, an alternative path can be found as (n, c), (c, x), and (x, m) as shown in Figure 4(a). To prove the correctness of Rule 5, we assume that Rule 2 and Rule 4 are not true and all neighbors of the node n's children are also children of the node n. In this case, since all edges other than the edge (n, p) cannot connect the node n to the lower layers, the edge (n, p) is a bridge. An instance of this case is depicted in Figure 4(b).

If all of these rules are not applicable, ENBRIDGE uses broadcast-based TURAU to find bridges. Thus, ENBRIDGE detects bridges, and execution of the ENBRIDGE terminates in all nodes. □

5.2. Message Complexity

Theorem 3. *The count of sent messages in I-PRITCHARD is 3N-1 at the best case and 4N-3 at the worst case.*

Proof. At the best case, the nodes are arranged as a star topology, where $2N$-1 messages are sent at the first phase, 1 message is sent by the center node at the beginning of the second phase, and N-1 messages are sent by the other nodes at the end of the second phase. Thus, $3N$-1 total messages are sent at the best case. At the worst case, $2N$-1 messages are sent at the first phase, N-1 announcement messages are sent at the beginning of the second phase, and N-1 at the end of the second phase; thus, $4N$-3 messages are sent. □

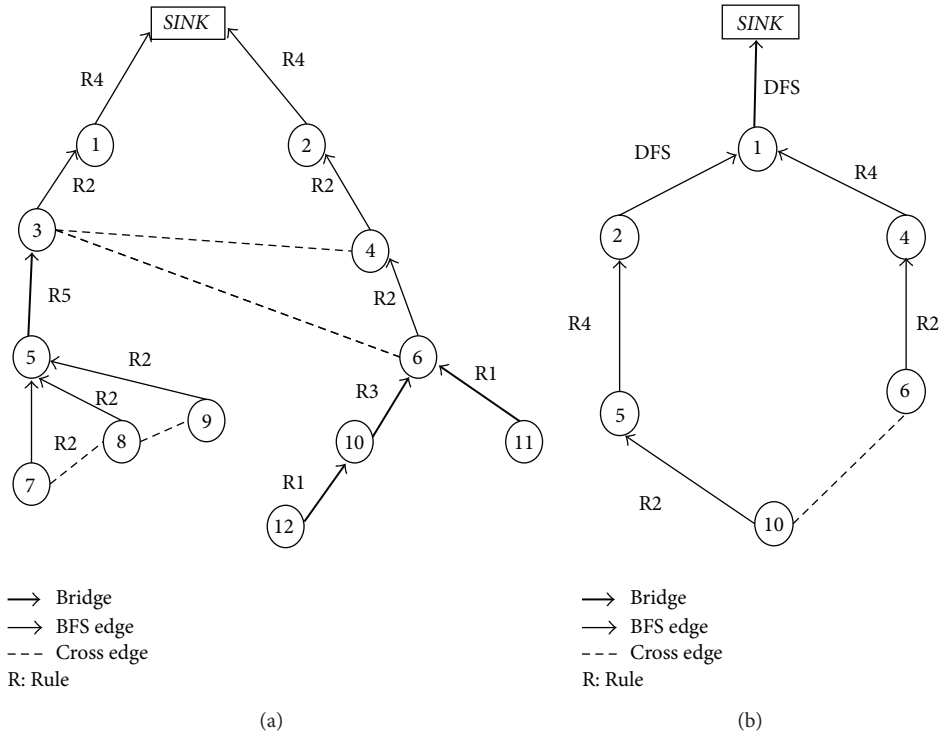

Figure 3: (a) ENBRIDGE Phase 1 classifies all edges and (b) ENBRIDGE Phase 2 is necessary.

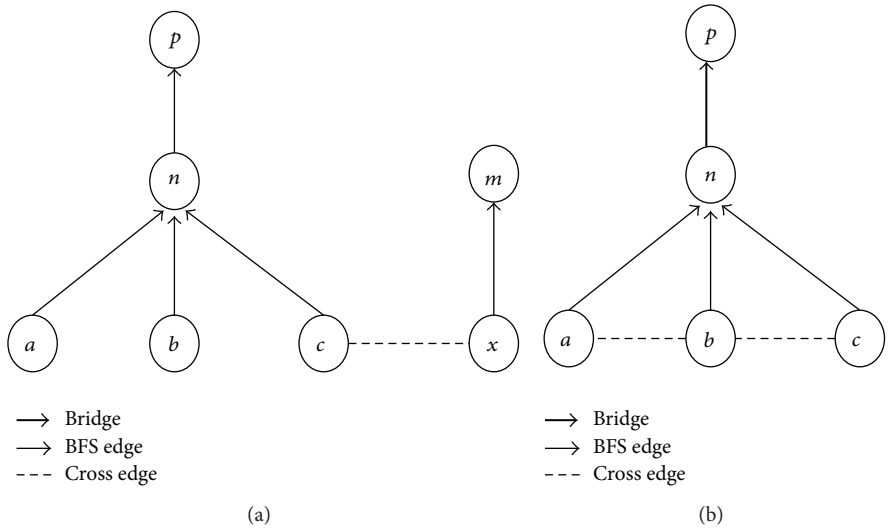

Figure 4: (a) Example for ENBRIDGE Rule 4. (b) Example for ENBRIDGE Rule 5.

Theorem 4. *The count of sent messages in ENBRIDGE is 2N-1 at the best case and 4N-3 at the worst case.*

Proof. At the best case, only BFS is executed on the star topology, so $2N$-1 messages are sent. At the worst case, an extra DFS is executed, where each node uses broadcast instead of unicast, so $2N$-2 messages are sent. Hence, $4N$-3 total messages are sent at the worst case. □

Theorem 5. *The count of received and overheard messages of I-PRITCHARD is O(ΔN).*

Proof. At the worst case, each node receives and overhears Δ messages at the first and second phases. Thus, total count for N nodes is O(ΔN). □

Theorem 6. *The count of received and overheard messages of ENBRIDGE is O(ΔN).*

Table 1: Analytical comparison of algorithms.

Algorithm/complexity	Sent messages	Received messages	Message size	Time	Space (per node)	Computational (per node)
CENTRAL	$\Theta(N^2)$	$O(\Delta N^2)$	$O(\Delta \log_2(N))$	$O(D)$	$O(E)$	$O(E)$
MILIC	$\Theta(N)$	$O(\Delta N)$	$O(E \log_2(N))$	$O(D)$	$O(E)$	$O(E)$
TURAU	$\Theta(E)$	$\Omega(\Delta N)$, $O(\Delta^2 N)$	$O(\log_2(N))$	$O(N)$	$O(\Delta)$	$O(\Delta^2)$
PRITCHARD	$\Theta(E)$	$\Omega(\Delta N)$, $O(\Delta^2 N)$	$O(\log_2(N))$	$O(D)$	$O(\Delta)$	$O(\Delta)$
I-PRITCHARD	$\Theta(N)$, lower:$3N-1$, upper:$4N-3$	$O(\Delta N)$	$O(\Delta \log_2(N))$	$\Omega(D)$, $O(N)$	$O(\Delta)$	$O(\Delta)$
ENBRIDGE	$\Theta(N)$, lower:$2N-1$, upper:$4N-3$	$O(\Delta N)$	$O(\Delta \log_2(N))$	$\Omega(D)$, $O(N)$	$O(\Delta^2)$	$\Omega(\Delta)$, $O(\Delta^3)$

Proof. Each node receives and overhears Δ *FORWARD* and *BACKWARD* messages during BFS operation at the worst case. This bound is the same for the DFS operation, so that total count of received and overheard messages of ENBRIDGE for N nodes is $O(\Delta N)$. □

Theorem 7. *The message size of I-PRITCHARD is $O((\Delta \log_2(N))$ bits.*

Proof. In I-PRITCHARD, each parent node announces pre-order label of its children by broadcasting a single message. Thus, the message size of the I-PRITCHARD is $O(\Delta \log_2(N))$ bits. □

Theorem 8. *The message size of ENBRIDGE is $O(\Delta \log_2(N))$ bits.*

Proof. At the backwards stage of ENBRIDGE, each node broadcasts its incident edge states to its neighbors. Because of this, the message size of the ENBRIDGE is $O(\Delta \log_2(N))$ bits. □

5.3. Time, Space, and Computational Complexities

Theorem 9. *The time complexity of I-PRITCHARD is $O(D)$.*

Proof. Since proposed improvements do no effect on time complexity, time complexity of I-PRITCHARD algorithm depends on the network diameter. Because of this, the time complexity of I-PRITCHARD is $O(D)$. □

Theorem 10. *ENBRIDGE takes $\Omega(D)$ time at the best case and $O(N)$ time at the worst case.*

Proof. At the best case, only Phase 1 is executed, so that the time complexity of the ENBRIDGE algorithm is equal to the time complexity of the BFS operation, so that the lower bound of the time complexity is $\Omega(D)$. At the worst case, Phase 2 is executed with phase 1. In this case, the time complexity is equal to the worst case time of the DFS operation, so that the upper bound of the time complexity is $O(N)$. □

Theorem 11. *The space and computational complexities of the I-PRITCHARD algorithm is $O(\Delta)$.*

Proof. Each node should store its neighbor's state where this table can be at most $O(\Delta)$. The algorithm executes on this table, so computational complexity is $O(\Delta)$. □

Theorem 12. *The space complexity of ENBRIDGE is $O(\Delta^2)$.*

Proof. Each node should store its 2-hop neighbor's state, so that this table can be at most $O(\Delta^2)$. □

Theorem 13. *The computational complexity of the ENBRIDGE algorithm is $O(\Delta^3)$. The lower bound for the computational complexity is $\Omega(\Delta)$.*

Proof. At the best case, one of Rule 1, Rule 2, and Rule 3 is executed which results in the $\Omega(\Delta)$ computational complexity. In order find the computational complexity of Rule 4, we assume that the node n has c cross edges which are represented as (a, b) and which are not incident to it but incident to one of its children (lets call it a). In order to find whether a is not equal to n or one of n's children, $c(\Delta^2 - c)$ computations should be made. If we maximize this equation, then $c = \Delta/\sqrt{2}$, and the computational complexity of Rule 4 becomes $O(\Delta^{5/2})$. Execution of Rule 5 can be $O(\Delta^3)$ at the worst case since a node may have Δ^2 2-hop neighbor nodes, and these 2-hop neighbor nodes are searched in the list of Δ 1-hop neighbor nodes. □

A summary and analytical comparison of central algorithm (CENTRAL), Milic's Algorithm (MILIC), Turau's Algorithm (TURAU), Pritchard's Algorithm (PRITCHARD), I-PRITCHARD, and ENBRIDGE algorithms are given in Table 1. MILIC, I-PRITCHARD, and ENBRIDGE are asymptotically better algorithms in terms of sent and received messages. The message sizes of I-PRITCHARD and ENBRIDGE are $O(\Delta \log_2(N))$; on the other side, message size of MILIC is $O(E \log_2(N))$. Although TURAU and PRITCHARD algorithm's message sizes are $O(\log_2(N))$ bits, the sent and received message counts are higher than other algorithms. Since ENBRIDGE algorithm's lower bound of sent message count is $2N$-1, it is analytically the best algorithm among other algorithms in terms of energy consumption caused by sent and received messages. ENBRIDGE is favorable in terms of energy consumption, but its time and computational complexities are worse than I-PRITCHARD as

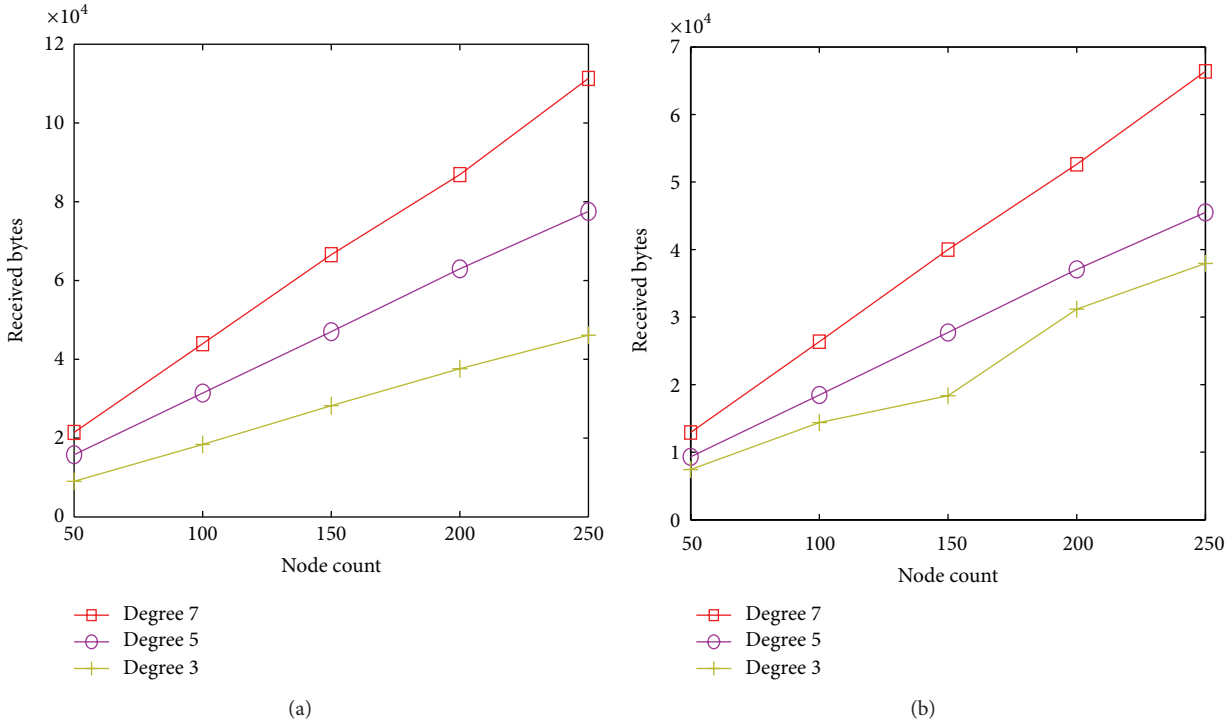

Figure 5: (a) Received bytes of I-PRITCHARD against node degree and node count. (b) received bytes of ENBRIDGE against node degree and node count.

shown in Table 1. Although the worst case time complexity of ENBRIDGE is worse than I-PRITCHARD, its best case time complexity is equal to I-PRITCHARD's time complexity. Besides, ENBRIDGE may terminate just after BFS execution, where I-PRITCHARD has an extra phase.

6. Performance Evaluations

We implemented I-PRITCHARD and ENBRIDGE algorithms in TOSSIM simulator [37] to evaluate their performance. TOSSIM simulator is developed by the researchers from University of California Berkeley, Intel Research Berkeley, and Harvard University. TOSSIM inherits the TinyOS's structure and provides a simulation environment that is close to the real world. The nesC compiler which is also currently used for the TinyOS applications is modified in order to use the same compiler for both simulation and implementation. A discrete event queue is used in the execution model. By using this queue, all simulator events are timestamped and processed in order. The hardware parts of the real-world implementations are emulated in TOSSIM. Two radio models are simulated. In the first model, the developers use error-free transmission to test the correctness of their protocols. The second model provides the developers to test the multihop transmissions.

We implemented CENTRAL, MILIC, TURAU, and PRITCHARD algorithms in order to compare them with the proposed algorithms. We generated randomly connected networks varying from 50 to 250 nodes that are uniformly distributed over the sensing area. For the lower layers, we implemented a TDMA based MAC protocol, and we used the IEEE 802.15.4 physical layer. The transmission power is 3 dBm. The average degrees of the nodes in generated networks are varying between 3, 5, and 7. We measured total received bytes, total sent bytes, energy consumption, and wallclock times. Each measurement is the average of 10 repeated simulations. In each simulation, nodes are randomly placed in the area. Table 2 summarizes the simulation parameters.

Table 2: Simulation parameters.

Node distribution	Random
Sink position	Randomly placed in the area
Number of sensors	50–250
MAC	TDMA
Transmission power	3 dBm
Transmission range	50 m
Node degrees	3, 5, and 7

6.1. Received Byte Counts. Since radio transmitter is the dominant energy consumer of the component of the sensor node, received byte counts are important evaluation criterions. Until otherwise stated, the default node degree is 5, and the node count is 150. Total received byte counts of I-PRITCHARD and ENBRIDGE against node count and node degree are shown in Figures 5(a) and 5(b), respectively. When the node degree and node count are increased, total received byte count of both algorithms increase linearly.

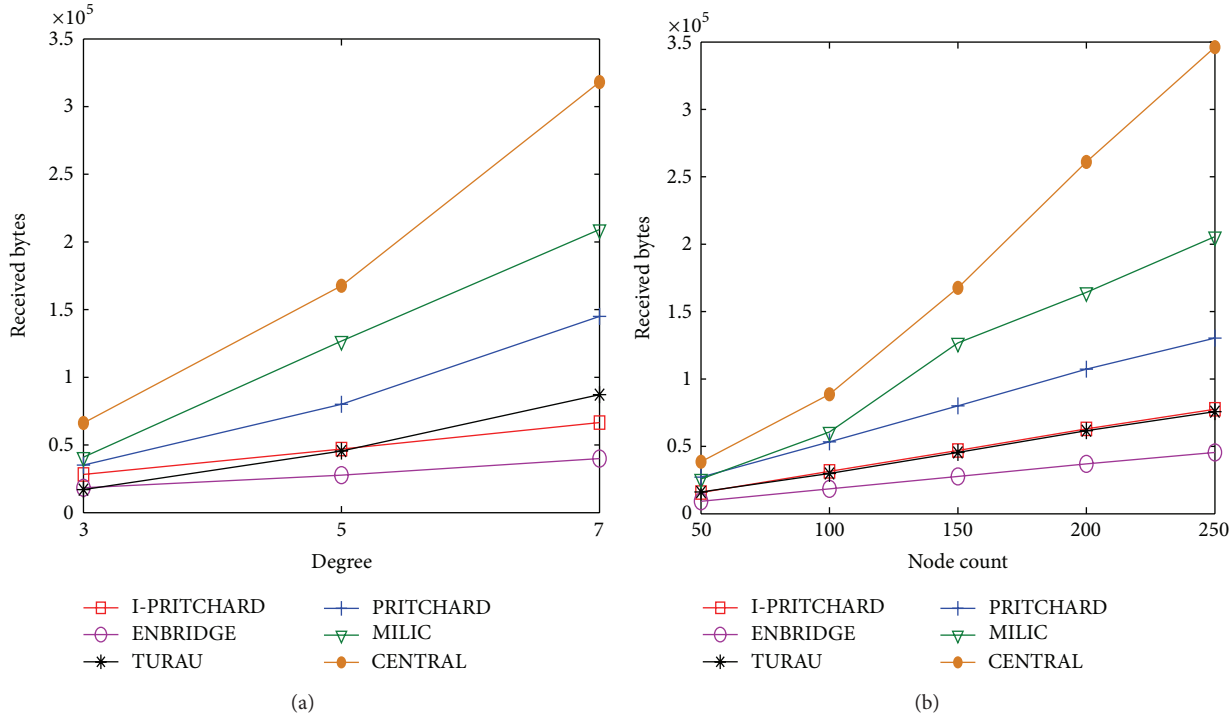

Figure 6: (a) Received bytes of algorithms against node degree. (b) Received bytes of algorithms against node count.

These results conform with theoretical analysis where the complexity of received and overheard messages is $O(\Delta N)$. These results show that I-PRITCHARD and ENBRIDGE are scalable against varying node degrees and node counts.

Performance comparisons of I-PRITCHARD, ENBRIDGE, and their counterparts are given in Figures 6(a) and 6(b). The received byte count of ENBRIDGE is the smallest, and I-PRITCHARD is the second smallest among the other algorithms. The received byte count performance order of algorithms is ENBRIDGE, I-PRITCHARD, TURAU, PRITCHARD, MILIC, and CENTRAL. These results show that BFS based approaches other than I-PRITCHARD and I-MILIC are worse than Turau's DFS based approach. I-PRITCHARD is approximately 1.7 times better than PRITCHARD on the average. ENBRIDGE is approximately 2 times better than TURAU, 3 times better than PRITCHARD, 4.5 times better than MILIC, and 6 times better than CENTRAL algorithm on the average. The reasons of this significant performance of proposed algorithms are using broadcast messages with at most $O(\Delta \log_2(N))$ bits.

6.2. Sent Byte Counts. Secondly, we measured the sent byte counts to evaluate the energy consumption of the algorithms. Total sent byte counts of I-PRITCHARD and ENBRIDGE against node count and node degree are shown in Figures 7(a) and 7(b), respectively. Total received byte count values fluctuate between similar values when the node degree and node count are increased. These results show that I-PRITCHARD and ENBRIDGE are stable and scalable against varying node degrees and node counts.

Total sent byte counts of I-PRITCHARD, ENBRIDGE, and their counterparts are given in Figures 8(a) and 8(b). The sent byte counts of ENBRIDGE are the smallest, and those of I-PRITCHARD are the second smallest among the other algorithms similar to the received byte count performance. The sent byte count performance order of algorithms is ENBRIDGE, I-PRITCHARD, TURAU, PRITCHARD, MILIC, and CENTRAL. The sent byte count of I-PRITCHARD approximately is 1.7 times smaller than PRITCHARD on the average. ENBRIDGE is approximately 1.8 times better than PRITCHARD, 3.6 times better than MILIC, 4.5 times better than MILIC, and 8 times better than CENTRAL algorithm on the average.

6.3. Energy Consumption. Energy efficiency is one of the most important objective for WSN. We measured the energy consumption of the distributed bridge detection algorithms. We assumed that the energy consumption occur mostly by message transfers. Energy consumptions of I-PRITCHARD and ENBRIDGE against node count and node degree are shown in Figures 9(a) and 9(b), respectively. Energy consumptions increase linearly when the node degree and node count are increased. These results show that the energy consumption of I-PRITCHARD and ENBRIDGE are stable and scalable against varying node degrees and node counts.

Performance comparisons of I-PRITCHARD, ENBRIDGE, and their counterparts are given in Figures 10(a) and 10(b). The ENBRIDGE algorithm consumes 3.4 mJ per node, and it has the best performance. The I-PRITCHARD algorithm consumes 5.3 mJ per node, and it has the second performance among the other algorithms. I-PRITCHARD

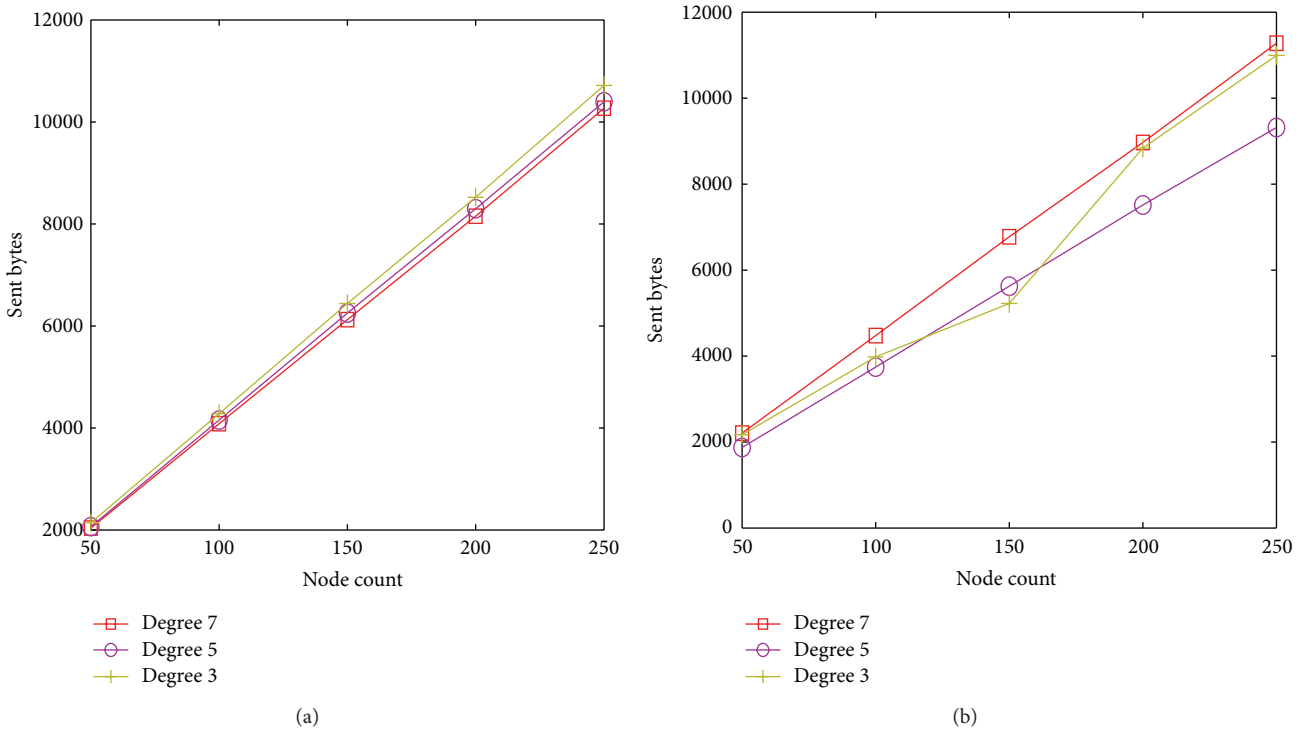

FIGURE 7: (a) Sent bytes of I-PRITCHARD against node degree and node count. (b) Sent bytes of ENBRIDGE against node degree and node count.

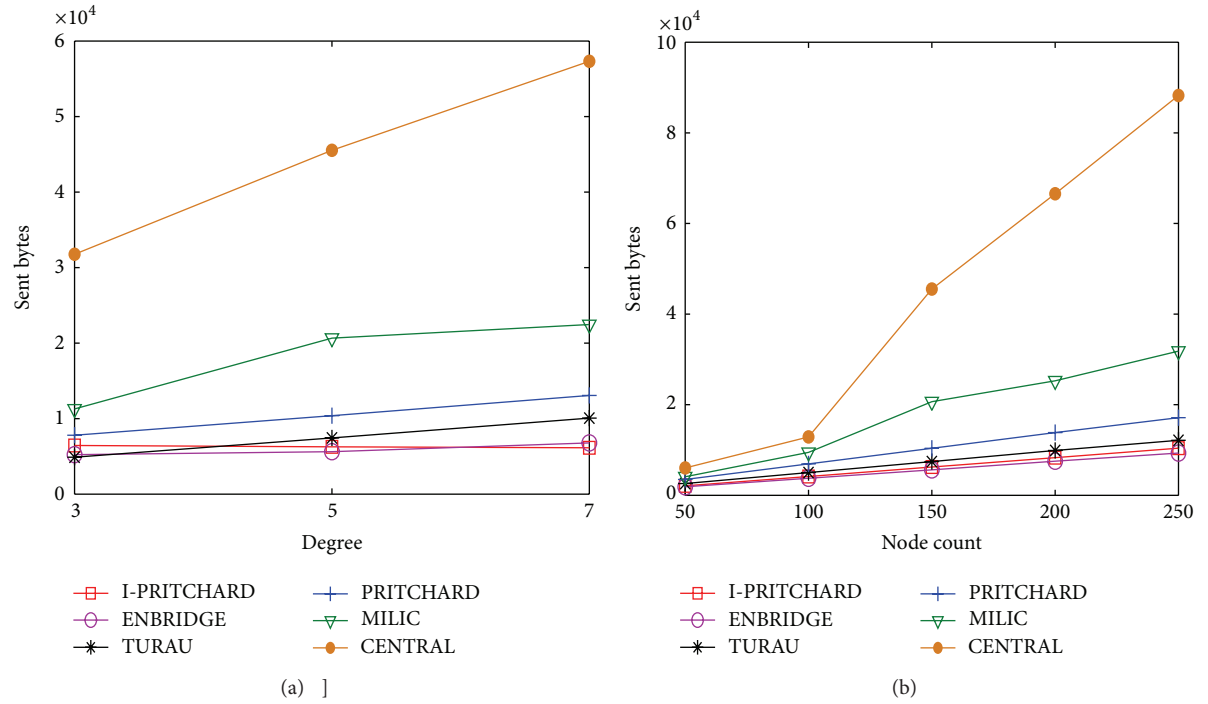

FIGURE 8: (a) Sent bytes of algorithms against node degree. (b) Sent bytes of algorithms against node count.

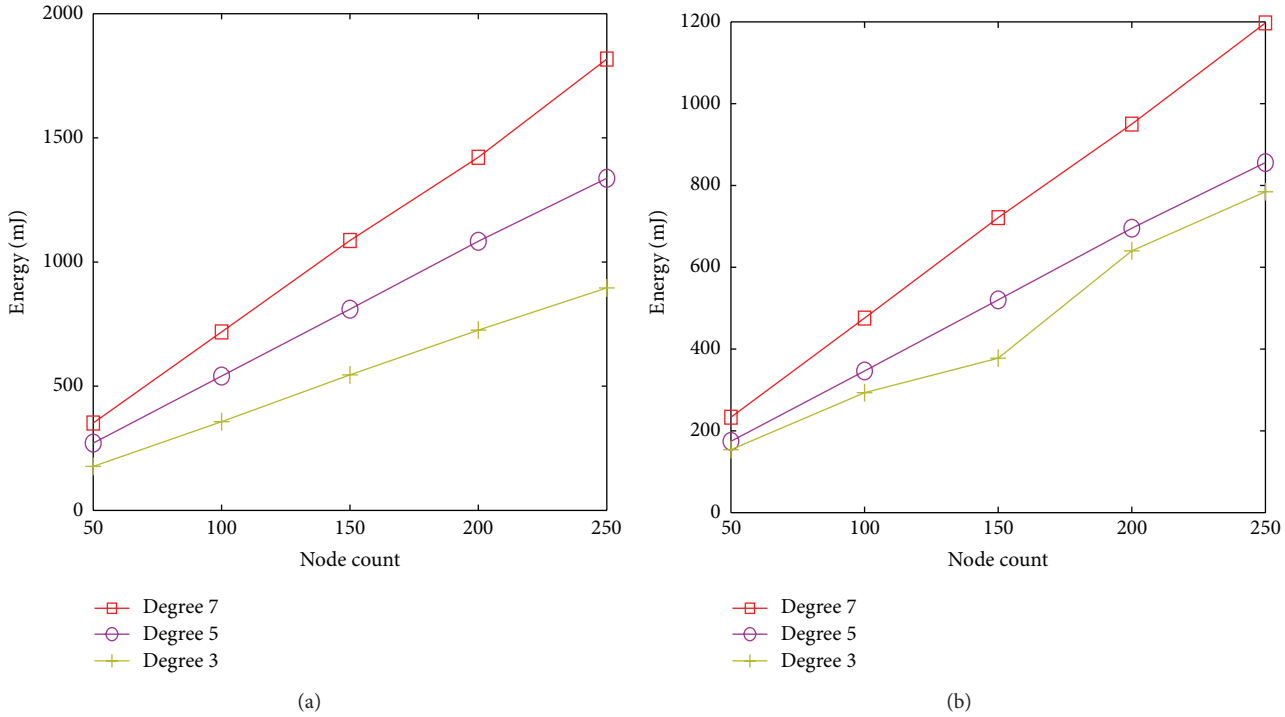

FIGURE 9: (a) Energy consumption of I-PRITCHARD against node degree and node count. (b) Energy consumption of ENBRIDGE against node degree and node count.

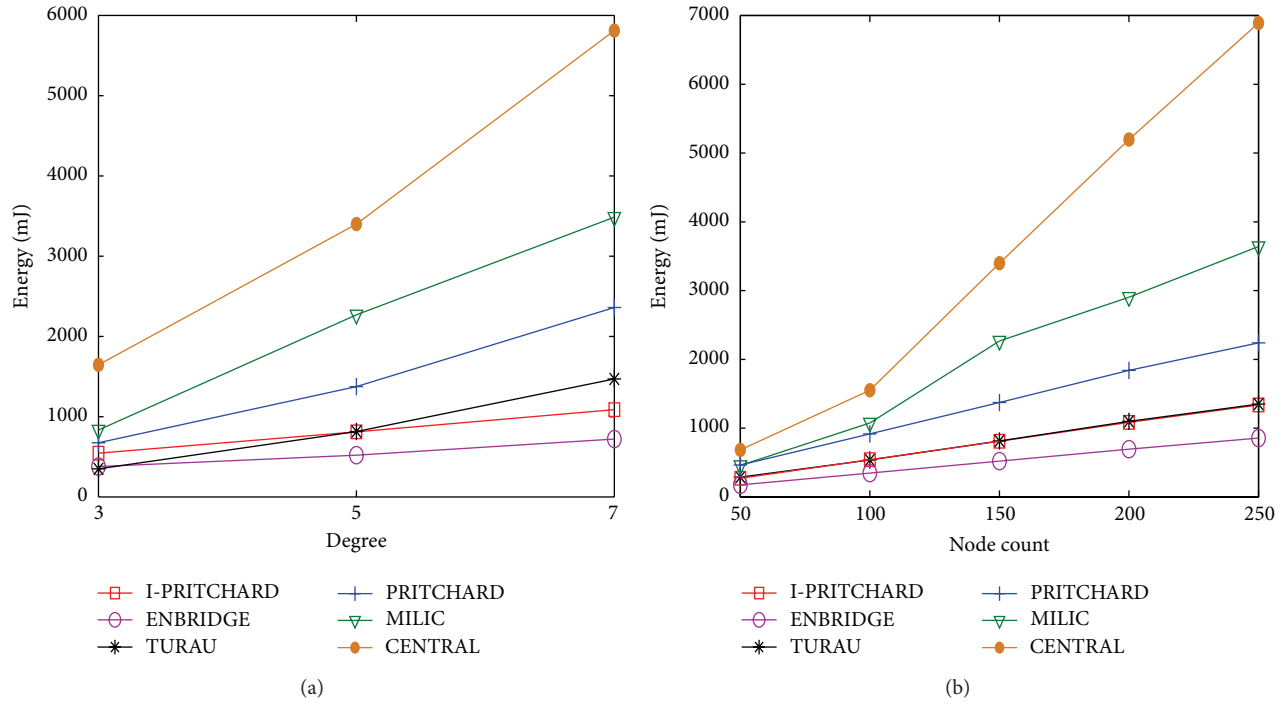

FIGURE 10: (a) Energy consumption of algorithms against node degree. (b) Energy consumption of algorithms against node count.

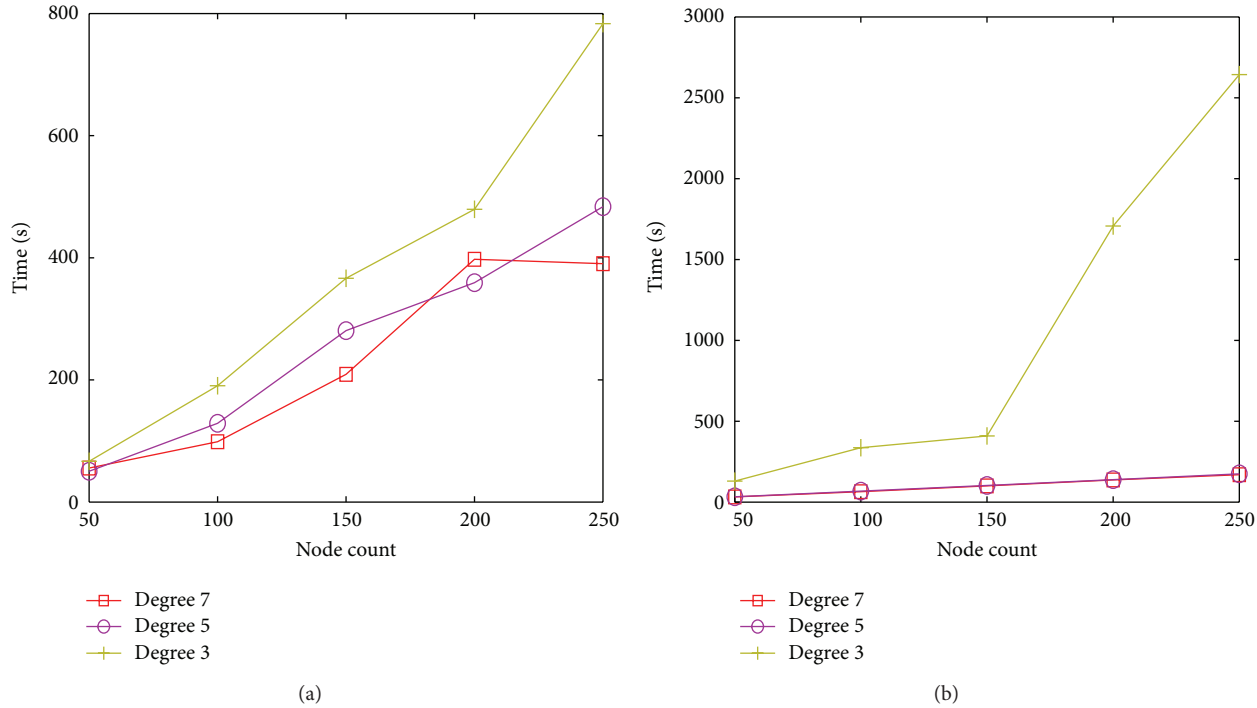

Figure 11: (a) Wallclock times of I-PRITCHARD against node degree and node count. (b) Wallclock times of ENBRIDGE against node degree and node count.

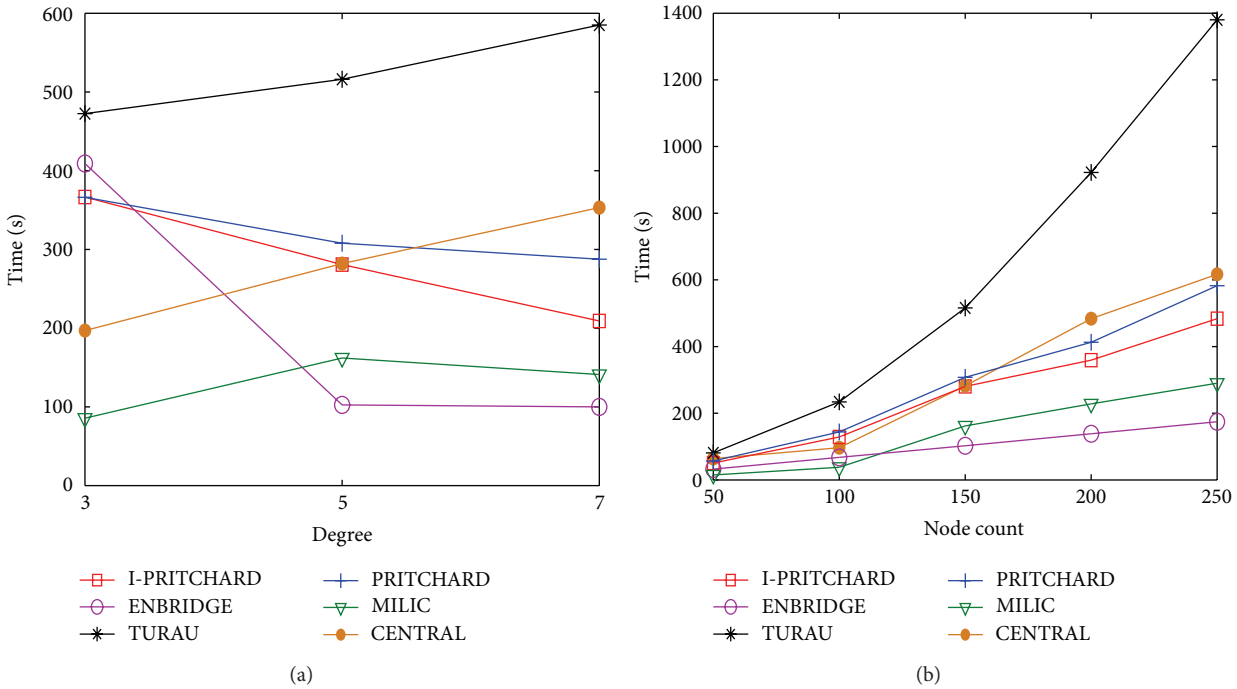

Figure 12: Wallclock times of algorithms against node degree. (b) Wallclock times of algorithms against node count.

consumes 1.7 times less energy than PRITCHARD algorithm. The energy consumption performance order of algorithms is ENBRIDGE, I-PRITCHARD, TURAU, PRITCHARD, MILIC, and CENTRAL. The energy consumption of ENBRIDGE is approximately 1.6 times better than TURAU, 2.6 times better than PRITCHARD, 4.3 times better than MILIC, and 6.5 times better than CENTRAL algorithm. This is a significant improvement over battery-powered sensor nodes in order to maximize the network lifetime.

6.4. Wallclock Times. Lastly, we measured the wallclock times of the distributed bridge detection algorithms. Firstly, we measured the wallclock times of I-PRITCHARD and ENBRIDGE algorithms against node count and node degree which are shown in Figures 11(a) and 11(b), respectively. Since the network diameter decreases when the degree increases, the wallclock times of both algorithms decrease. A sharp increase in the wallclock time of ENBRIDGE algorithm can be observed in Figure 11(b) when the node count is 150 and the degree is 3. The reason of this sharp increase is the fact that for especially sparse networks, ENBRIDGE runs both of the phases. Since the second phase is improved DFS based cut bridge detection, time consumption increases.

Wallclock times of I-PRITCHARD, ENBRIDGE, and their counterparts are given in Figures 12(a) and 12(a). The wallclock times of ENBRIDGE are the smallest, and those of I-PRITCHARD are the second smallest among the other algorithms. TURAU has the worst performance since it is DFS based and uses unicast for the message transmission. I-PRITCHARD is better than PRITCHARD in all cases since its phase count is 1 less. MILIC algorithm performs well for most of the cases. Although the performance of the ENBRIDGE is not good for some cases as explained in the previous paragraph, the algorithm performs best among other algorithms on the average since it completes its operation within a BFS session in most of the simulations.

In this section, we showed that our analytical results given in Section 5 conform with the simulations results that ENBRIDGE and I-PRITCHARD outperform the previously proposed approaches in terms of energy and time consumptions.

7. Conclusions

We proposed two distributed energy-efficient bridge detection algorithms I-PRITCHARD and ENBRIDGE. Our first algorithm, is the improved version of the Pritchard's algorithm. In this algorithm, we merged two phases into a single phase which leads to the removal of a downcast operation. Besides, we used radio broadcast communication instead of unicast message transfer that leads to the reduction of message header transmissions. The original idea of the second algorithm is to process proposed rules on 2-hop neighborhood information during a BFS session in order to detect bridges and classify all edges. With the help of these methods, our algorithms have $O(N)$ sent message complexity, $O(\Delta N)$ received and overheard message complexity where the largest message is $O(\Delta \log_2(N))$ bits.

We showed the detailed design of the proposed algorithms with example operations. We analyzed the proof of correctness, message complexity, time complexity, space complexity, and computational complexity and compared them with the previous work. We implemented the algorithm on TOSSIM environment with its counterparts. From our extensive simulations, we showed that the received and sent byte counts of I-PRITCHARD are always less than PRITCHARD. Besides, I-PRITCHARD consumes less energy and time than PRITCHARD in all simulation steps. These simulation results conform with the theoretical analysis and show that I-PRITCHARD improves PRITCHARD both theoretically and practically. Our second designed algorithm, ENBRIDGE, has the best simulation performance in terms of received byte count, sent byte count, energy consumption, and wallclock times. The energy savings of ENBRIDGE algorithm when compared to the other approaches are between 1.6 and 4.3. This is a significant improvement for energy-efficient bridge detection in sensor networks in order to detect weak points of the network and to prevent possible faults.

References

[1] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA '02)*, pp. 88–97, Atlanta, Ga, USA, September 2002.

[2] S. H. Lee, S. Lee, H. Song, and H. S. Lee, "Wireless sensor network design for tactical military applications: remote large-scale environments," in *Proceedings of the 28th IEEE Conference on Military Communications (MILCOM '09)*, pp. 911–917, Boston, Mass, USA, 2009.

[3] S. Hussain, S. Schaffner, and D. Moseychuck, "Applications ofwireless sensor networks and RFID in a smart home environment," in *Proceedings of the 7th Annual Communication Networks and Services Research Conference (CNSR '09)*, pp. 153–157, Moncton, Canada, May 2009.

[4] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM Journal on Computing*, vol. 1, pp. 146–160, 1972.

[5] Y. D. Liang and C. Rhee, "Optimal algorithm for finding biconnected components in permutation graphs," in *Proceedings of the ACM Computer Science Conference (CSC '95)*, pp. 104–108, Nashville, Tenn, USA, March 1995.

[6] R. Thurimella, "Sub-linear distributed algorithms for sparse certificates and biconnected components," in *Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing (PODC '95)*, pp. 28–37, Ontario, Canada, August 1995.

[7] B. Milic, N. Milanovic, and M. Malek, "Prediction of partitioning in location-aware mobile ad hoc networks," in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS '05)*, vol. 9, p. 306, Big Island, Hawaii, USA, January 2005.

[8] V. Turau, "Computing bridges, articulations, and 2-connected components in wireless sensor networks," in *Proceedings of the 2nd international conference on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS '06)*, pp. 164–175, Springer, Venice, Italy, 2006.

[9] B. Milic and M. Malek, "Adaptation of the breadth first search algorithm for cut-edge detection in wireless multihop networks," in *Proceedings of the 10th ACM Symposium on Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWiM '07)*, pp. 377–386, Chania, Greece, October 2007.

[10] S. Wang, X. Mao, S. J. Tang, X. Y. Li, J. Zhao, and G. Dai, "On movement-assisted connectivity restoration in wireless sensor and actor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 4, pp. 687–694, 2011.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, McGraw-Hill, Cambridge, UK, 2nd edition, 2001.

[12] K. Lu, L. Huang, Y. Wan, and H. Xu, "Energy-efficient data gathering in large wireless sensor networks," in *Proceedings of the 2nd International Conference on Embedded Software and Systems (ICESS '05)*, pp. 327–331, Xi'an, China, 2005.

[13] H. Haanpaa, A. Schumacher, T. Thaler, and P. Orponen, "Distributed computation of maximum lifetime spanning subgraphs in sensor networks," in *Proceedings of the 3rd International Conference on Mobile Ad Hoc and Sensor Networks (MSN '07)*, pp. 445–456, Springer, Beijing, China, 2007.

[14] K. Erciyes, D. Ozsoyeller, and O. Dagdeviren, "Distributed algorithms to form cluster based spanning trees in wireless sensor networks," in *Proceedings of the 8th International Conference on Computational Science (ICCS '08)*, pp. 519–528, Springer, Kraków, Poland, 2008.

[15] S. Chen, M. Coolbeth, H. Dinh, Y.-A. Kim, and B. Wang, "Data collection with multiple sinks in wireless sensor networks," in *Proceedings of the 4th International Conference on Wireless Algorithms, Systems, and Applications (WASA '09)*, pp. 284–294, Springer, Boston, Mass, USA, 2009.

[16] V. Savic, A. Población, S. Zazo, and M. García, "An experimental study of RSS-based indoor localization using nonparametric belief propagation based on spanning trees," in *Proceedings of the 4th International Conference on Sensor Technologies and Applications (SENSORCOMM '10)*, pp. 238–243, Venice, Italy, July 2010.

[17] I. Cidon, "Yet another distributed depth-first-search algorithm," *Information Processing Letters*, vol. 26, no. 6, pp. 301–305, 1988.

[18] W. Hohberg, "How to find biconnected components in distributed networks," *Journal of Parallel and Distributed Computing*, vol. 9, no. 4, pp. 374–386, 1990.

[19] P. Chaudhuri, "An optimal distributed algorithm for computing bridge-connected components," *Computer Journal*, vol. 40, pp. 200–207, 1997.

[20] Y. H. Tsin, "Some remarks on distributed depth-first search," *Information Processing Letters*, vol. 82, pp. 173–178, 2002.

[21] D. Pritchard, "An optimal distributed bridge-finding algorithm," in *Proceedings of the 25th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC '06)*, Denver, Colo, USA, July 2006.

[22] O. Dagdeviren and K. Erciyes, "Graph matching-based distributed clustering and backbone formation algorithms for sensor networks," *Computer Journal*, vol. 53, no. 10, pp. 1553–1575, 2010.

[23] P. Sommer and R. Wattenhofer, "Gradient clock synchronization in wireless sensor networks," in *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN '09)*, pp. 37–48, San Francisco, Calif, USA, April 2009.

[24] H. Cheng, Q. Liu, and X. Jia, "Heuristic algorithms for real-time data aggregation in wireless sensor networks," in *Proceedings of the International Wireless Communications and Mobile Computing Conference (IWCMC '06)*, pp. 1123–1128, Vancouver, Canada, July 2006.

[25] "IEEE Standard for Information technology-Telecom. and information exchange between systems-Local and metropolitan area networks-Specific requirements-Part 15.4: Wireless Medium Access Control and Physical Layer Specifications for Low-Rate Wireless Personal Area Networks," 2003.

[26] "IEEE 802.11 Standard Working Group, Draft Standard for Information Technology—Telecom. and Information Exchange Between Systems—LAN/MAN Specific requirements Part 11: Wireless LAN Medium Access Control and Physical Layer Specifications," 2007.

[27] W. Ye, J. Heidemann, and D. Estrin, "Medium access control with coordinated adaptive sleeping for wireless sensor networks," *IEEE/ACM Transactions on Networking*, vol. 12, no. 3, pp. 493–506, 2004.

[28] L. Choi, S. H. Lee, and H. Choi, "M-mac: mobility-based link management protocol for mobile sensor networks," in *Proceedings of the Software Technologies for Future Dependable Distributed Systems (STF-SSD '09)*, pp. 210–214, Tokyo, Japan, 2009.

[29] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*, pp. 95–107, Baltimore, Md, USA, 2004.

[30] V. Rajendran, K. Obraczka, and J. J. Garcia-Luna-Aceves, "Energy-efficient, collision-free medium access control for wireless sensor networks," *Wireless Networks*, vol. 12, no. 1, pp. 63–78, 2006.

[31] B. Awerbuch, "A new distributed depth-first-search algorithm," *Information Processing Letters*, vol. 20, pp. 147–150, 1985.

[32] K. B. Lakshmanan, N. Meenakshi, and K. Thulasiraman, "A time-optimal message-efficient distributed algorithm for depth-first-search," *Information Processing Letters*, vol. 25, no. 2, pp. 103–109, 1987.

[33] M. B. Sharma and S. S. Iyengar, "An efficient distributed depth-first-search algorithm," *Information Processing Letters*, vol. 32, pp. 183–186, 1989.

[34] T.-Y. Cheung, "Graph traversal techniques and the maximum flow problem in distributed computation," *IEEE Transactions on Software Engineering*, vol. 9, pp. 504–512, 1983.

[35] D. Kumar, S. S. Iyengar, and M. B. Sharma, "Corrigenda: corrections to a distributed depth-first search algorithm," *Information Processing Letters*, vol. 35, pp. 55–56, 1990.

[36] S. A. M. Makki and G. Havas, "Distributed algorithms for depth-first search," *Information Processing Letters*, vol. 60, no. 1, pp. 7–12, 1996.

[37] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: accurate and scalable simulation of entire TinyOS applications," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys '03)*, pp. 126–137, Los Angeles, Calif, USA, November 2003.