Computing Bridges, Articulations, and 2-Connected Components in Wireless Sensor Networks

Volker Turau

Hamburg University of Technology, Institute of Telematics
Schwarzenbergstraße 95, 21073 Hamburg, Germany
turau@tuhh.de

Abstract. This paper presents a simple distributed algorithm to determine the bridges, articulation points, and 2-connected components in asynchronous networks with an *at least once* message delivery semantics in time $O(n)$ using at most $4m$ messages of length $O(\lg n)$. The algorithm does not assume a FIFO rule for message delivery. Previously known algorithms either use longer messages or need more time. The algorithm meets the requirements of wireless senor networks and can be applied in several areas relevant to this field such as topology control, clustering, localization and virtual backbone calculations.

1 Introduction

Sensor networks - networks of small, resource-constrained wireless devices embedded in the physical environment - present new challenges to the design and implementation of distributed algorithms. Energy efficiency is the key to prolonging the network life time and is thus of primary importance. Communication is the main consumer of energy and the consumption grows with the lengths of the messages. If energy consumption is not equally distributed over all nodes in the network, hot spots will emerge. This will lead to an early failure of these nodes, which may result in a disconnected network, that is no longer able to fulfill its task. Hence, it is important to identify hot spots and to use alternative routing paths to equally spread the load of communication.

The topology of a wireless sensor network is represented by an undirected graph $G = (V, E)$ where V is the set of nodes and $E \subseteq V \times V$ the set of edges describing the available communication links: (u, v) belongs to E means that u can send messages to v and vice versa. The topology depends on uncontrollable factors such as node mobility, weather interference noise, multi-path fading as well as on controllable parameters such as transmit power. Articulation points of G are likely to become hot spots, if they fail or run out of energy the network becomes disconnected. Hence, the number of articulation points of G reveals the number of weak points within the network topology. A high degree of fault-tolerance is achieved for networks that are k-connected, and this

property increases with increasing k. These networks provide multiple-path redundancy between every pair of nodes enabling techniques such as long sleeping periods and load balancing. Using random geometric graphs Bettstetter proved in [1] that the probability that a network with $n \gg 1$ nodes, each node with a transmission range r_0, and a homogeneous node density ρ is k-connected is $P(G \text{ is } k-\text{connected}) \approx P(\delta \geq k)$ for $P(\delta \geq k)$ almost one (δ denotes the minimum node degree of G) and this probability rises with ρ and r_0. Increasing the node density comes with higher costs and is very often not a valid option. The value of r_0 is of crucial importance for the functionality of the network. If the transmission power is too low, connectivity of all nodes is not guaranteed. If the power is too high, there may be too much interference, i. e., multi-user interference may not allow for efficient use of bandwidth. Furthermore, increasing the transmission range means a higher consumption of energy.

The topology of a wireless sensor network can be controlled by some tunable parameters such as transmitting power and antenna directions. This process is called topology control and extensive research has been done in this field in recent years [2–6]. The goal is to allow each node in the network to adjust its transmitting power individually (i. e., to determine its neighbors) so that a *good* network topology can be formed. In [2] Ramanathan and Hain propose a simple heuristic to achieve biconnectivity through the control of transmission power. If the network is connected but not biconnected each node attempts to do biconnectivity augmentation as follows. Every node sets a timer with a value t that is randomized around an exponential function of the distance from the next articulation point. If after time t the network is still not biconnected the node increases its power to the maximum possible. Nodes closer to an articulation point are more likely to remove the articulation and therefore given priority using timers. To determine the biconnected components a centralized algorithm is used, this is not a realistic option in non-static networks. The topology control algorithm presented in [6] extends the work of Ramanathan et al., but computing 2-connected components is still not distributed.

This paper contributes towards a solution for the topology control problem: A simple distributed algorithm to determine the bridges, articulation points, and 2-connected components in an asynchronous network using at most $4m$ messages with a length of $O(\lg n)$. Compared with the currently best distributed depth-first algorithms, this is only an increase of $n - 1$ messages, these are needed to inform each node about the 2-connected components they belong to. The assumptions for this algorithm meet the requirements of wireless sensor networks: The semantics of message delivery is *at least once* and the FIFO rule for message delivery is not assumed (i. e., messages transmitted over a link in the same direction can arrive at the other end of the link in any order). The footprint of the algorithm is small: On the average a node needs to store $2m/n+2$ identifiers in addition to a few local variables. The output is available in a distributed manner: Each link knows whether it is a bridge, each node knows whether it is an articulation and is aware of the 2-connected components it

belongs to. Previously known algorithms either use longer messages or need more time.

Many algorithms developed for wireless sensor networks can benefit from the knowledge of articulations points. Connected dominating sets (CDS) have been proposed as virtual backbones of wireless ad hoc networks [7]. All articulation points are included in every minimum CDS. Hence, identifying articulations points can reduce the communication effort for determining a CDS: If $M_1, \ldots, M_s$ are CDSs of the 2-connected components and A is the set of articulations of G, then $\bigcup_{i=1}^{s} M_i \cup A$ is a CDS of G (not necessarily a minimum CDS). T. Hara proposed a replica allocation method in ad hoc networks were the biconnected components form groups of nodes maintaining replicas [8]. Span is a power saving technique for multi-hop ad hoc wireless networks that reduces energy consumption without significantly diminishing the capacity or connectivity of the network [9]. When a region of a network has a sufficient density of nodes, only a small number of them need be on at any time to forward traffic. Obviously, putting nodes that are articulations into sleep mode leads to a disconnected network. Hence, this work also benefits from the knowledge of the articulations and biconnected components. Further applications domains include clustering (a cluster should be fully contained in a 2-connected component), localization of nodes ([10]), and TDMA slot assignment algorithms.

This paper is organized as follows: Section 2 summarizes related work and Section 3 defines the computational model used for this work. The algorithm and its analysis is presented in Section 4. A short discussion of an implementation of the algorithm using a real wireless sensor network concludes the paper.

2 Related Work

The presented algorithm is the first algorithm for this problem with time complexity $O(n)$ using $O(\lg n)$-messages. The messages consist of an identifier (3 bits) and at most one additional integer. The algorithm transmits $4m$ messages in the worst and $2m + n - 1$ messages in the best case, m being the number of links in the network. If message delivery is guaranteed in a single unit of time, the algorithm terminates within $2n + d - 2$ time units in the worst case (d is the depth of the search tree). Hohberg [11] presents a distributed algorithm for the problem at hand using $2m + n - 1$ messages. The proposed algorithm needs $2m + n - 1$ units of time and is consequently considerably slower than our algorithm. While all messages of Hohberg's algorithm have length $O(\lg n)$, in our algorithm $m - n$ of the messages have only length $O(1)$. In [12] Chaudhuri presents a distributed algorithm for the restricted problem of finding the bridge-connected components. While the number of messages is $O(n)$, it uses messages of length $O(n)$ and it is based on the FIFO rule. An algorithm for synchronous networks is described in [13]. The communication costs are not analyzed in detail, but are high compared with our algorithm. Swaminathan and Goldman present in [14] an incremental distributed algorithm for computing the 2-connected components in a dynamically changing graph. After a new edge is

inserted it takes $O(b+c)$ messages of length at most $O(b(b+e))$ to recompute the 2-connected components, where c is the number of 2-connected components and b and e are the numbers of nodes and edges in the resulting 2-connected component. The algorithm is rather complex and an implementation would probably overstrain the main memory of many of the currently available sensor nodes.

3 Computational Model

This paper assumes an asynchronous model based on message passing. Let $G(V, E)$ represent a connected communication network, with V being the set of nodes and E the set of bidirectional communication links. The asynchronous network has the following properties:

1. No two nodes in the network share memory.
2. The semantics of message delivering is *at least once*, i. e., there is a guarantee that every message sent is also received, but a node may receive a message more than once. The most straightforward realization of this semantics is through ARQ: Upon receiving a message, a node sends an acknowledgment to the sender. If a sender does not receive an acknowledgment within a given unit of time, the message is sent again until the receiver confirms the receipt of the message.
3. Message delivery times vary and cannot be predicted or bounded.
4. Messages sent over a link are not corrupted. This can be achieved by using error-correcting codes in combination with ARQ.
5. Every message reaches its receiver after a finite amount of time.
6. Messages sent over a link may not necessarily arrive in the same order they were sent (i. e., no FIFO rule for message delivery is assumed). This can be a consequence of the way messages are handled at the MAC-layer.
7. Every node is aware of all its links. A node knows the identity of the link over which a message is received. Nodes have a unique identifier and have no global knowledge about the network.

This computational model meets the requirements of most current wireless sensor networks. The main weakness of the model is that it does not tolerate failures of links or nodes. In case a node fails during execution, the algorithm terminates without computing all articulations.

4 Algorithm

The proposed algorithm is based on the distributed depth-first search algorithm invented by Cidon [15] and corrected by Tsin [16], which does not assume the FIFO rule. Interleaved into the determination of the depth-first tree is the calculation of the articulations using a technique from the sequential algorithm for this problem due to Tarjan [17]. The main challenge is to develop an algorithm within the computational model that keeps the total number of messages and their lengths as small as possible.

4.1 Informal Description of the Algorithm

The algorithm uses five types of messages and marks the links at each end with an attribute. Initially all link ends are marked `UNVISITED`. The messages `Forward` and `Backtrack` are used to explore the network in a depth-first order. The message `Visited` is used to avoid sending the message `Forward` to a node that is already discovered. Upon receiving the message `Forward` for the first time, a node sends concurrently to its normal course `Visited` messages to all neighbors except to the father of this node. Nodes receiving a `Visited` message mark their end of the link as `VISITED`. This attempts to prevent the neighbors from considering this node as yet undiscovered and from sending a `Forward` message to it at some later time. If however due to delays, a node v_1 is discovered before a `Visited` message from a previously visited neighbor v_2 has arrived, a `Forward` message might be sent from v_2 to v_1. To deal with this situation, node v_1 must discard the `Forward` message and v_2 must send a `Forward` message to another undiscovered neighbor (or a `Backtrack` message to the father node if all neighbors are already discovered). This behavior is enforced by the following two rules:

- An already discovered node ignores `Forward` messages received over links marked `UNVISITED`.
- A node receiving a `Visited` message over a link marked `SON` sends a `Forward` message to another undiscovered neighbor.

As shown by Tsin in [16] the unpredictability of message delivery times can cause other situations that must be taken care of to retain the depth-first character of the search. Due to these complications the classical algorithm of Tarjan to determine articulations and 2-connected components based on the so-called *low values* needs to be adapted. The low value of a node is the minimum of

1. the depth of the node,
2. the depth of the nodes reachable via back-edges from this node, and
3. the low values of the sons of the node.

The challenge is to extend the depth-first algorithm such that it computes the low values of all nodes without sending additional messages or using long messages. Compared with the algorithm of Tsin, only one additional type of message is needed (message `Inform`) and the length of some messages is increased to hold an additional integer. The message `Forward` carries the depth of the current node to a son. Upon receiving a `Forward` message for the first time, a node x determines its own depth. By broadcasting its depth with a `Visited` message, the neighbors of x get notified that x has been visited. This information is used to carry out step 2 for calculating low values. In case a `Visited` has already been received over a link, the node at the other end of the link has a lower depth, and the current node will not change its low value. Therefore, the depth is only included in `Visited` messages sent over links marked `UNVISITED`, an empty `Visited` message is still sent over links marked `VISITED` to retain the depth-first search. At any time a node stores the lowest of the depth values transmitted by

`Visited` messages. All but at most one of the `Visited` messages arriving at a node come over back edges. Consider the case where a node v_1 has two successors v_2, v_3 in a depth-first search tree. Once the node v_1 is discovered it sends a `Forward` message to v_2 and a `Visited` messages to v_3. But when the search returns to v_1 a `Forward` message is sent to v_3 (for an example consider nodes 3, 4, and 5 in Figure 1). Hence, the value carried by the `Visited` messages sent to v_3 must not be used for the calculation of the low value of node v_3. As proved in Section 4.3, this situation can easily be recognized. On receiving a `Backtrack` message carrying the low value of a son step 3 is accomplished. Finally the low value of a node is calculated immediately before the search backtracks to a father node.

A block is a maximal, non-trivial connected subgraph without an articulation, blocks are either 2-connected components or bridges. Articulations are recognized upon the receipt of a `Backtrack` message: According to [17], if the depth of a node is less than or equal to the low value of a son, then the node is an articulation. In case the depth is strictly less than the low value the corresponding edge is marked as a bridge otherwise it is marked `CLOSED`. This marks the *end* of a 2-connected component. To identify the nodes that belong to 2-connected component, the message `Inform` is used. Upon the detection of an articulation point an `Inform` message carrying the identifier of the new block is recursively sent over all reachable `SON` links except those marked `CLOSED`, i. e., over the edges of the depth first search tree of the current block. `Inform` messages are sent concurrently to the depth-first exploration of the rest of the network. This way, each node will know the block it belongs to. Block identifiers can be assembled from the id of the articulation together with the link identifier. Node identifiers are only needed for this calculation, in practical applications block identifiers can also be randomly generated. `Inform` messages are sent in parallel to exploring the rest of the graph. The opposite ends of bridges are recognized just before a `Backtrack` message is sent: If the low value is equal to the depth, the link is a bridge and it is marked accordingly.

4.2 Formal Description of the Algorithm

All nodes execute the same algorithm, which consists of a handler for every of the five types of messages. All nodes allocate memory for the following seven variables:

boolean *root* ← *articulation* ← **false**
boolean *state* ← UNDISCOVERED
int *low* ← *depth* ← MAX_VALUE
List *links*, *block_ids* ← ∅

The variable *root* indicates that the node is the root of the search tree. The variable *articulation* will hold the value **true** if and only if the node is an articulation. All nodes are initially in state `UNDISCOVERED`, upon receiving the first `Forward` message, the state changes into `DISCOVERED`. The list *links* stores the state of each link of the node, there are eight different states. Initially all links

have state `UNVISITED`. The initially empty list *block_ids* will finally contain the identifiers of all blocks the node belongs to. All nodes implement the following two routines:

```
procedure SEARCH
    if ∃ link l s.t. l.state = UNVISITED then
        l.state ← SON
        send new Forward(depth) over l
    else if root = true then
        if |block_ids| = 1 then
            articulation ← false
        end if
    else
        l ← link in state FATHER
        low ← min(depth, low)
        if low = depth then
            l.state ← BRIDGE_FATHER
        end if
        send new Backtrack(low) over l
    end if
end procedure
```

```
procedure RESTART(link l)
    if l.state = UNVISITED then
        l.state ← VISITED
    else if l.state = SON then
        l.state ← VISITED
        SEARCH
    end if
end procedure
```

At every node the following operations are carried out when a message of the specified type is received over link l. It is assumed, that the nodes execute the operations corresponding to a message completely before executing the code of the next message, i. e., incoming messages are put into a queue of bounded length.

▷ Message $Backtrack(son\_low)$

```
if l.state = SON then
    if depth <= son_low then
        if depth < son_low then
            l.state ← BRIDGE_SON
        else
            l.state ← CLOSED
        end if
        articulation ← true
        generate new block_id
        block_ids.insert(block_id)
        send new Inform(block_id) over l
    else
        l.state ← BACKTRACKED
    end if
    low ← min(son_low, low)
    SEARCH
end if
```

▷ Message $Forward(p\_depth)$

```
if state = UNDISCOVERED then
    state ← DISCOVERED
    l.state ← FATHER
    depth ← p_depth + 1
    if low = p_depth then
        low ← depth
    end if
    SEARCH
    send new Visited(depth) over all
          links in state UNVISITED
    send new Visited() over all links
          in state VISITED
else
    RESTART(l)
end if
```

```
▷ Message Start
state ← DISCOVERED
depth ← low ← 0
root ← true
SEARCH
send new Visited(depth) over all links in state UNVISITED
```

```
▷ Message Inform(block_id)
if block_id ∉ block_ids then
    block_ids.insert(block_id)
    send new Inform(block_id) over all links in state BACKTRACKED
end if
```

```
▷ Messages Visited(p_depth) and Visited()
if l.state = UNVISITED || l.state = SON then
    if p_depth included && p_depth < low then
        low ← p_depth
    end if
    RESTART(l)
end if
```

The procedure SEARCH implements the depth-first search, a node explores every possibility of extending the search via all links before backing up to the father node. The procedure RESTART is used to restart the search after a message has been received out of order due to message delays. The structure of the depth-first tree is available through the states of the links. Links marked `FATHER` or `BRIDGE_FATHER` lead to the predecessor of a node and links marked `CLOSED`, `BRIDGE_SON` or `BACKTRACKED` lead to successors. The ends of edges that are bridges are labeled `BRIDGE_SON` and `BRIDGE_FATHER`. The algorithm is invoked by sending the parameterless message `Start` to a node that will be used as the root of the search tree. Figure 1 demonstrates an application of the algorithm.

4.3 Correctness and Analysis

To prove the correctness of the algorithm the concept of a token traveling through the network is used. The token is introduced by the message `Start`. It is passed on by the messages `Forward` and `Backtrack`, but only in the following two cases:

1. `Forward` passes the token to the receiving node if this node is in state `UNDISCOVERED`.
2. `Backtrack` passes the token to the receiving node if the message is received over a link in state `SON`.

First, we prove that at every point in time there is a single token in the network. Every attempt to pass on the token is initiated by a call of the procedure SEARCH. It suffices to prove that whenever a node calls procedure SEARCH it

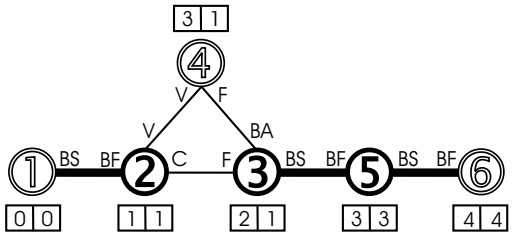

Fig. 1. Network with 6 nodes, bold nodes are articulations and bold edges are bridges, and node 1 is the root. The nodes are visited in ascending order of their identifiers. The nodes are annotated with the pair $depth, low$ and the links are annotated with their final state. The blocks are $\{1, 2\}, \{2, 3, 4\}, \{3, 5\}$ and $\{5, 6\}$.

possesses the token and since the node last received the token the node has not passed it to another node. Initially only the root node possesses the token. If SEARCH is called upon receiving a `Forward` message at a node v_1, then either the node was in state `UNDISCOVERED` when the message arrived and thus just received the token, or it was in state `DISCOVERED`. In the latter case SEARCH is only called when the message was received from a node v_2 over a link l in state `SON`. Thus, a `Forward` message was previously sent over l (but in the opposite direction). Since the state of l is still `SON`, no `Backtrack` message was received over l. This implies that this is the link over which v_1 has sent its last `Forward` message. If this `Forward` message has passed the token to v_2, then v_2 must have been in state `UNDISCOVERED` and had not sent any message over this this link and also would never send a `Forward` message over that link. But since the link is in state `SON`, this is impossible. Hence, the last `Forward` message did not pass the token and v_1 is still in possession of the token.

SEARCH is also called upon receiving message `Visited` over a link marked `SON`. As in the previous case, the last `Forward` message did not pass the token and the node possesses the token at this time. The only other case SEARCH is called, is when a `Backtrack` message is received over a link in state `SON`. But then the node has just been given the token. This proves, that at every point in time there is a single token in the network and thus, no two branches are followed concurrently. Since every node explores every possibility of extending the search via all links before backing up to the father node, the search employs the depth-first order and the token traverses the network in depth-first order. This also proves that the algorithm terminates. Figure 2 depicts all possible transitions of states of ends of links. The states `FATHER` and `BRIDGE_FATHER` are called father-states and `SON`, `CLOSED`, `BRIDGE_SON` and `BACKTRACKED` are called son-states. Note that once an end of a link has a son- or father-state, this property holds forever with only one exception, it is possible to transit from `SON` to `VISITED`.

In the following we prove that the algorithm correctly labels all links, i. e., tree-edges with father- respectively son-states and back-edges with state `VISITED`. As shown above, the token traverses the network in depth-first order, i. e., the

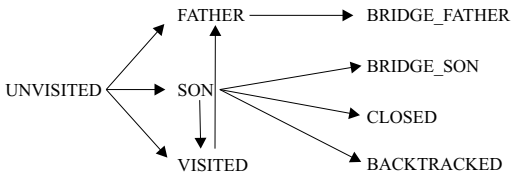

Fig. 2. Transitions of states of ends of links.

edges the token is passed on are exactly the tree-edges. A node receiving the token for the first time - except the root node - marks that link with a father-state and it remains in a father-state. A node marks a link as `SON` before sending a `Forward` message. If this message passes on the token, it remains forever in a son-state. Otherwise, the node will later receive a `Visited` or `Forward` message that will change the label to `VISITED`. Furthermore, if a node in state `DISCOVERED` receives a `Forward` message over a link l, then l is already in state `VISITED` or will change into that state. This proves, that the algorithm correctly labels all links. Because of the idempotence of routine RESTART, receiving duplicate messages of type `Visited` or `Forward` is not a problem. On the first receipt of a message `Backtrack` the state of the link is changed, hence any `Backtrack` message not received over a link in state `SON` can be safely discarded. A similar argument proves that duplicate `Inform` messages also do no harm.

To prove the correctness of the algorithm it remains to prove, that the algorithm correctly computes the low values. First, the case of receiving a `VISITED` message over a tree-edge is considered. The depth value included in this message must not be used to calculate the low value of the receiving node. A correction is necessary, if and only if the depth value carried by a `Forward` message equals the current lowest depth value included in all `Visited` messages received so far. This only occurs if the `Forward` message was received over the same link as the `Visited` message leading to the current lowest depth value. In this case the current lowest depth value is equal to the depth of the father node. Since a low value of a node is bounded by the depth of the node, the error can be safely corrected by taking the depth of the node as the current lowest depth value, this is just the current lowest depth value incremented by 1.

When a node is ready to send a `Backtrack` message to its father, there are no links marked `UNVISITED`. Furthermore, the node will pass the token to the father node and the token will never return to this node. Hence, after this event the node receives no more `Visited` messages over links marked `SON`. Since `Visited` messages are never received over links marked `VISITED` or `FATHER`, the node receives no more `VISITED` messages at all. Thus, the node has received the depth of all neighbors reachable via back edges with `Visited` messages and has discarded `VISITED` messages received over tree-edges. Furthermore, the node knows its own depth and the low values of all sons from the corresponding `Backtrack` messages. Hence, the node is in a position to calculate its own low value and to include it in the `Backtrack` message to be sent to its father. On

receiving a `Backtrack` message with the low value of the son, a node can decide whether it is an articulation.

The depth-first search algorithm of Tsin transmits at most $4m - (n - 1)$ messages. Our algorithm uses in addition the `Inform` messages, one per tree edge. Thus, the total number of messages is bounded by $4m$. In the best case, there are only two `Visited` messages per back edge and one `Forward`, `Backtrack`, and `Inform` per tree edge, summing up to $2m + n - 1$ messages in the best case. All but $m - (n - 1)$ messages carry 3 bits to identify the kind of the message and an integer less than n. `Visited` messages sent over links marked `VISITED` carry only the message identifier, there is at least one such message for every back edge. Thus, the message length is $O(\lg n)$. In the best case our algorithm transmits $m - (n - 1)$ integers less than Hohberg's algorithm, in the worst our algorithm transmits $m - (n - 1)$ more integers.

The time complexity is the maximum duration from sending the `Start` message until the termination of the algorithm under the assumption that the time of delivering a single message over each link is at most one unit of time. This assumption is only made for this calculation, the algorithm operates correctly for any finite message delivery time. In the following it is proved, that passing the token takes at most one unit of time. For this purpose a node that passes successfully the token with a `Forward` message at time t to a node v is considered. The node v must have been in state `UNDISCOVERED` when receiving the `Forward` message, this did not happen later than time $t + 1$. All `Visited` and `Forward` messages destined for node v were sent prior to time t, the time the father of v send the `Forward` message to v. Hence, at time $t + 1$ node v must have received all of these messages. Thus, v knows whether it has send a `Forward` message without passing the token. This yields, that at time $t + 1$ node v passes the token either to a son or back to its father. Since the token is passed twice along every tree edge, the depth first search needs $2n - 2$ units of time. In the worst case when the graph is 2-connected sending the `Inform` messages is started after the depth-first search has finished. In this case our algorithm needs d additional units of time (d is the depth of the search tree). In total the algorithm requires no more than $2n - 2 + d$ units of time. This is a considerable improvement over previous algorithms. Using induction, it is straightforward to prove that the sum of the lengths of the lists *block_ids* of all nodes is at most $2(n-1)$. This bound is attained for trees only. Hence, on the average there are about two identifiers in each list. Furthermore, on the average each node has $2m/n$ neighbors, the state of each can be safely stored in one byte.

5 Conclusion

This paper presented a novel distributed algorithm for computing bridges, articulations, and 2-connected components of undirected graphs in time $O(n)$ using at most $4m$ messages of length $O(\lg n)$. In order to improve this result either a more efficient distributed depth-first search algorithm is needed or a completely different approach must be taken (i. e., not based on depth-first search). The al-

gorithm is suitable for usage in many application of wireless sensor networks due to its short messages and its robustness, i. e., it does not assume a FIFO rule for message delivery and is immune from message duplication. The algorithm has been successfully implemented and tested on a real wireless sensor network consisting of ESB and ECR nodes of the ScatterWeb platform developed at the FU Berlin [18]. Each node was equipped with 2 kByte RAM and 64 kByte EEPROM and with a wireless communication device working at 19200 Bits/s. The experiment was conducted using 24 nodes with varying topologies.

References

1. Bettstetter, C.: On the minimum node degree and connectivity of a wireless multihop network. In: Proc. ACM MobiHoc, ACM (2002) 80–91
2. Ramanathan, R., Rosales-Hain, R.: Topology control of multihop wireless networks using transmit power adjustment. In: IEEE INFOCOM 2000. (2000) 404–413
3. Borbash, S., Jennings, E.: Distributed topology control algorithm for multihop wireless networks. In: Proc. Int. J. Conf. on Neural Networks. (2002) 355–360
4. Lloyd, E., Liu, R., Marathea, M.V., Ramanathan, R., Ravi, S.: Algorithmic aspects of topology control problems for ad hoc networks. In: Proc. of the 3rd ACM Int. Symposium on Mobile ad hoc Networking and Computing. (2002) 123–134
5. Liu, J., Li, B.: Distributed topology control in wireless sensor networks with asymmetric links. In: Proc. of IEEE Globecom 2003. (2003) 1257–1262
6. Tseng, Y., Chang, Y., Tzeng, B.: Energy-efficient topology control for wireless ad hoc sensor networks. J. of Information Science and Engineering **20** (2004) 27–37
7. Wu, J., Li, H.: On calculating connected dominating set for efficient routing in ad hoc wireless networks. In: Proc. of the 3rd ACM Int. Workshop on Discrete algorithms and methods for mobile computing and communications. (1999) 7–14
8. Hara, T.: Replica allocation methods in ad hoc networks with data update. Mobile Networks and Applications **8** (2003) 343–354
9. Chen, B., Jamieson, K., Balakrishnan, H., Morris, R.: Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. Wireless Networks **8** (2002) 481–494
10. Aspnes, J., Eren, T., Goldenberg, D.K., Morse, A.S., Whiteley, W., Yang, Y.R., Anderson, B.D.O., Belhumeur, P.N.: A theory of network localization. To appear, IEEE Transactions on Mobile Computing (2006)
11. Hohberg, W.: How to find biconnected components in distributed networks. Journal of Parallel and Distributed Computing **9**(4) (1990) 374–386
12. Chaudhuri, P.: An optimal distributed algorithm for computing bridge-connected components. Computer Journal **40**(4) (1997) 200–207
13. Thurimella, R.: Sub-linear algorithms for sparse certificates and biconnected components. Journal of Algorithms **23**(1) (1997) 160–179
14. Swaminathan, B., Goldman, K.: An incremental distributed algorithm for computing biconnected components in dynamic graphs. Algorithmica **22** (1998) 305–329
15. Cidon, I.: Yet another distributed depth-first search algorithm. Inform. Process. Lett. **26**(6) (1988) 301–305
16. Tsin, Y.H.: Some remarks on distributed depth-first search. Inform. Process. Lett. **82**(4) (2002) 173–178
17. Tarjan, R.: Depth first search and linear graph algorithms. SIAM Journal Computing **1** (1972) 146–160
18. ScatterWeb. `http://www.scatterweb.net` (2006)