D-LPCN: A Distributed Least Polar-angle Connected Node Algorithm for Finding the Boundary of a Wireless Sensor Network

Saoudi Massinissa, Farid Lalem, Ahcène Bounceur, Reinhardt Euler, M. Tahar Kechadi, Abdelkader Laouid, Madani Bezoui, Marc Sevaux

D-LPCN: A Distributed Least Polar-angle Connected Node Algorithm for Finding the Boundary of a Wireless Sensor Network

Massinissa Saoudi[a,f,c], Farid Lalem[a,f], Ahcène Bounceur[a,f,*], Reinhardt Euler[a,f], M-Tahar Kechadi[c], Abdelkader Laouid[d], Madani Bezoui[b], Marc Sevaux[a,e]

[a] *Lab-STICC UMR CNRS 6285*
[b] *University of Boumerdes, Department of Mathematics, Boumerdes, Algeria*
[c] *Lab-CASL Laboratory,University College Dublin, Belfield, Dublin 4, Ireland*
[d] *LIMED Laboratory, University of Bejaia, 06000, Bejaia, Algeria*
[e] *Centre de Recherche, BP 92116, F-56321 Lorient, France*
[f] *Université de Bretagne Occidentale, 20 Avenue Victor Le Gorgeu, 29238 Brest, France*

Abstract

A boundary of wireless sensor networks (WSNs) can be used in many fields, for example, to monitor a frontier or a secure place of strategic sensitive sites like oil fields or frontiers of a country. This situation is modeled as the problem of finding a polygon hull in a connected Euclidean graph, which represents a minimal set of connected boundary nodes. In this paper we propose a new algorithm called D-LPCN (Distributed Least Polar-angle Connected Node) which represents the distributed version of the LPCN algorithm introduced in [1]. In each iteration, any boundary node, except the first one, chooses its nearest polar angle node among its neighbors with respect to the node found in the previous iteration. The first starting node can be automatically determined using the Minimum Finding algorithm, which has two main advantages. The first one is that the algorithm works with any type of a connected network, given as planar or not. Furthermore, it takes into account any blocking situation and contains the necessary elements to avoid them. The second advantage is that the algorithm can determine all the boundaries of the different connected parts of the network. The proposed algorithm is validated using the CupCarbon, Tossim and Contiki simulators. It has also been implemented using real sensor nodes based on the TelosB and Arduino/XBee platforms. We have estimated the energy consumption of each node and we have found that the consumption of the network depends on the number of the boundary nodes and their neighbors. The simulation results show that the proposed algorithm is less energy consuming than the existing algorithms and its distributed version is less energy consuming than the centralized version.

Keywords: Distributed algorithms, Wireless Sensor Networks, Boundary detection, Energy consumption, Polygon hull.

*Corresponding author
Email addresses: Massinissa.Saoudi@univ-brest.fr (Massinissa Saoudi), Farid.Lalem@univ-brest.fr (Farid Lalem), Ahcene.Bounceur@univ-brest.fr (Ahcène Bounceur), Reinhardt.Euler@univ-brest.fr (Reinhardt Euler),

1. Introduction

The rapid improvement in wireless communication has enabled the development of large amounts of tiny devices called sensor nodes [2], which are able to sense, process, and communicate data in order to perform two important tasks: sensing and communicating. To achieve these, the sensors must be well connected and the target area should be well covered by these sensors. The communication must be reliable in order to ensure good exchanges of messages between nodes over short distances using radio modules. A good coverage will guarantee a good acquisition of data that are interesting for the study of physical phenomena.

According to the range of the communication, sensor nodes are deployed either densely or sparsely, in a deterministic way or randomly, in the region to monitor (e.g., industrial processes, residential estates, potential disasters [3], military applications of tracking and numerous other applications). In some WSN applications, recognizing boundary nodes is necessary to discover the network's topology [4], to perform geographical routing [4, 5], tracking and guiding [4], to monitor boundaries of strategic and sensitive sites (e.g., oil fields or nuclear sites, frontiers of a country, etc.).

Since energy consumption is a limiting factor for the lifetime of a node, the communication time has to be minimized. To overcome this limitation, we propose in this paper to activate the sensing units of the whole network and to activate the radio module of the network's boundary nodes to ensure the communication between them and to keep the radio module of the other nodes asleep periodically (duty cycling) in order to preserve their energy. Indeed, if all sensor nodes of the inner region participate actively in communication, an important size of data transmitted between nodes leads to an important energy consumption, and thus to an important reduction of the network's lifetime.

Existing boundary determination methods mainly deal with planar networks, which requires additional algorithms to transform the initial network into a network without crossings (e.g., the Gabriel graph [6]). These algorithms are often non-exact, based on heuristics and in some situations it is even not possible to eliminate crossings that lead to a non-communicating sensor node. Moreover, they may use all the nodes of the network to determine the boundary and therefore, all the nodes will consume energy.

Our proposed algorithm, called *D-LPCN (Distributed Least Polar-angle Connected Node)* works without any restriction and transformation on the network, relies exclusively on the nodes of the boundary and their neighbors, and works with any topology even unconnected (i.e., with many connected sub-networks) and with any density of the network. In the case of an unconnected network, the proposed algorithm can determine the boundary nodes of each connected component. On the basis of our results, we may even conclude that the energy consumption is reduced with respect to the energy consumption of the existing algorithms.

`tahar.kechadi@ucd.ie` (M-Tahar Kechadi), `laouidkader@gmail.fr` (Abdelkader Laouid), `mbezoui@univ-boumerdes.dz` (Madani Bezoui), `Marc.Sevaux@univ-ubs.fr` (Marc Sevaux)

The D-LPCN algorithm will find a minimal set of connected boundary nodes that have been deployed deterministically or randomly and which are situated on the boundary of a WSN. Basically, it represents the distributed version of the LPCN algorithm that we have presented in [1], where in each iteration the subsequent boundary node v_{i+1} is given by the node that has the minimum angle $\varphi_{min}(v_{i-1}, v_i, v_{i+1})$ formed by the edges $\{v_i, v_{i-1}\}$ and $\{v_i, v_{i+1}\}$ (cf. Figure 1), v_i being the boundary node selected in the current iteration i and v_{i-1} the one found in the previous iteration $i-1$.

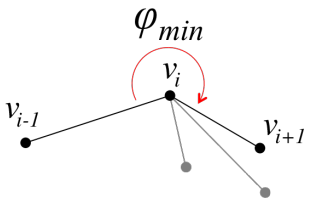


Figure 1: Angle formed by three sensor nodes.

The proposed algorithm is based on the localization information of the nodes that only need to communicate with their one-hop neighbor nodes. Many existing methods can be used for the location information. We will give in the following some existing methods with their advantages and drawbacks. Traditionally, a GPS receiver is the main device used to localize sensor nodes [7], especially when high accuracy is needed. Indeed, it is very energy consuming in the case of continuing use. However, it can be used normally for smart-city WSNs since they can be powered continually using the urban electricity. Also, in the case of static WSNs, the GPS can be used only once in order to get the location without any impact on the energy consumption, since the GPS coordinates will remain unchanged during the network's lifetime. Similarly, in the case of WSNs with mobile nodes, the GPS can be used once and then it is possible to use some existing methods to predict the new position of the mobile node based on its trajectory. The GPS can also be used in the case of autonomous mobile nodes that are limited in time of use, such as UAVs intercepting a malicious flying object. Except for these cases, unfortunately, the GPS must be avoided. Other less energy consuming techniques can be used. Authors of [8] summarize some location techniques which are compared according to their accuracy. Some techniques consider the sensor nodes individually where each sensor node calculates and estimates its location by calculating the distance and the angle formed with its neighbor nodes [8]. Many localization techniques have been presented in the literature, such as TOA, TDOA, RSSI, AOA, HIRLOC, SERLOC, and ADLA [9] which use the power of the received signals in order to determine the location. Other methods use hybrid sensor nodes where only some sensor nodes are equipped with GPS, called anchor nodes [10] or beacon nodes [7]. These nodes will be used to determine the location of other un-localized nodes. The accuracy of these methods depends on the number of anchor nodes.

The structure of this paper is as follows. In the next section, related work is presented. The LPCN algo-

rithm will be discussed in Section 3. Next, in Section 4, we will introduce the proposed D-LPCN algorithm. Section 5 exhibits the devices used for its implementation and discusses the obtained results. Section 6 contains a comparison between the centralized and the distributed version of the proposed algorithm. Section 7 is on some technical improvements of the algorithm. Finally, Section 8 concludes the paper.

2. Related Work

In order to find the boundary nodes of a wireless sensor network several methods have been developed. They can be classified into three categories: geometrical, statistical, and topological. The geometrical methods [11, 4, 12] are the most accurate. They use the location information of each node to determine the boundary nodes [12, 4]. Statistical methods make assumptions on the probability distribution of the node deployment, and identify the boundary nodes based on statistical properties under certain network conditions [12, 4]. The topological methods use the connectivity information to determine the boundary nodes [12]. These methods allow the exchange of connectivity information among neighbor nodes, and they detect the boundary nodes without any information on their location. Generally, they outperform the statistical methods [11, 12, 4].

In [13], a distributed algorithm based on topological methods has been proposed. It first determines the boundary of a set of nodes that will be considered as a set of seeds. Then, it determines the maximum hop distances around each seed and examines whether the contour around a seed forms a closed cycle. Only nodes that are near to the boundaries are identified and they are not explicitly connected as a polygon. In addition, this approach requires high network densities.

Another distributed algorithm based on topological methods is presented in [5]. To determine the boundary nodes, this algorithm assumes that the nodes can communicate with their one-hop and two-hop neighbor nodes. It tests whether the nodes within the two-hop neighbors can be used to form a cycle with the concerned node. If not, it is classified as a boundary node. This distributed algorithm does not distinguish between the nodes near the boundary and those of the voids, and it works only with dense networks.

In [4], a heuristic based on finding boundary cycles inside a network is developed. The main idea of this approach is to start from a first cycle which will be increased gradually by adding other vertices until it hits the boundary of the network. A vertex with the highest degree will be chosen as the center of the cycle. In case that other cycles are found, they will be merged. This approach can work with low density wireless sensor networks.

The authors of [14] have presented a technique to obtain a boundary node using image processing algorithms after transforming the network graph into an image. Unfortunately, this technique is centralized and defines only the convex hull.

In [15], an algorithm is proposed which starts from the sink and which selects the node with the maximum

or minimum x- or y-coordinate. This procedure is repeated for each selected node until the sink is reached. Unfortunately, this algorithm does not handle some cases which may cause infinite loops.

In [16], an algorithm called *ABBD* (Angle Based Boundary Detection Algorithm) is developed to detect boundaries using the distance between neighbor nodes. The main idea of this algorithm is to test whether each sensor node is covered by the polygon formed by its neighbor nodes. A sensor node will be declared a non-boundary node, if it is covered by at least three nodes. This condition is not always verified because it is possible to find nodes that are covered by less than three nodes and that are not a boundary node (e.g., an inner sensor node with one neighbor). This algorithm suffers from an excess of communications.

Another algorithm proposed by [17] is based on Ircod sensors. These sensors determine a cyclic order for their neighbors and choose the first neighbor in this order. Then, the Right Hand Without Crossings (RHWoC) rule is used to determine the complete boundary. The drawback of this algorithm is that it assumes that there is no more than one neighbor node of the Ircod sensor node with the same angle or on the same line. However, it detects only the convex hull of the given nodes. This algorithm suffers from the stopping condition, i.e., it finishes as soon as the sink is revisited, and it requires to find the middle of the network which is a difficult task.

In [18], two algorithms called Sequential Boundary Node Selection (SBNS) and Distributed Node Boundary Selection (DBNS) are developed to find out the boundary nodes of the WSN without or with the presence of obstacles over the deployed region. The resolution of the algorithm starts from the sink in the middle, and selects the network and its neighbors. After iterative processes, only the last nodes are selected as boundary nodes forming the perimeter of the network.

In [19] , an algorithm called Distributed Boundary Detection (DBD) is developed for identifying the boundaries of obstacles and networks. Each node requires only the information of its three-hop neighbors. A node equipped with this algorithm can determine whether it is itself a boundary node in a distributed manner, and identify the outer boundary of a network. Hence, this method is based on a heuristic approach which approximately estimates the border of the network.

In [20], authors introduce a communication model based on a virtual disk unit graphics (QUDGs). They presented a new framework for the self-organization of a large sensor network. The baseline scenario can be described as follows: Given a large swarm of stationary sensor nodes that have been dispersed in a polygonal region, the authors give a deterministic distributed approach to identify nodes that are within the polygonal area, or near its limit. Their algorithm is based on topological considerations and geometric packing arguments. Using the boundary structure, they describe a deterministic distributed method to extract the graph of the street of the swarm.

In [21], authors propose a decentralized boundary detection technique without location information. In this technique, each sensor node knows the information on three-hop, two-hop and one-hop neighbors by means of HELLO messages. In fact, this technique ensures the accuracy of border and hole detection in

comparison with a heuristic technique. However, this method does not support densely deployed architectures due to the resource restriction and its principle to request information on one-, two- and three-hop neighbors.

Authors in [22] propose a large-scale coverage hole algorithm for hole boundary detection based on the notion of minimum critical threshold constraint (BLW-MCT). The BLW-MCT is a geometric topological hybrid method, which allows topological global rough boundary detection and geometric local large-scale rough boundary detection. Their work is based on the following idea: if the range of large-scale coverage holes in each global rough boundary can be quantized into a reasonable threshold, then it will be easily introduced to BLW-MCT. In fact, this method is efficient in terms of computational complexity and network overhead. The proposed solution is probabilistic and not accurate since based on the minimization of computational complexity and network overhead.

The authors of [23] propose a new algorithm called Sensor Localization and Obstacle Discovery (SLOD) for localizing obstacles and sensor nodes based on four GPS enabled special nodes called anchor nodes. The authors estimate the position of the sensor nodes with an approximate rectangular boundary of each obstacle detected. Then, they determine the accurate shapes of the obstacles, based on the data communication between neighbor sensor nodes, deployed around the obstacles and using a heuristic method.

In [24], the authors propose two distributed algorithms based on computational geometry, Distributed Sector Cover Scanning (DSCS) and Directional Walk (DW). The first algorithm is used to identify the nodes on hole borders and the outer boundary. The second is used to locate the coverage holes based on the boundary nodes identified with DSCS and which they enclose. These algorithms are based on the observation that a node can have its sensing area divided into different sectors by its neighbors. Each node needs to collect the information of its one- and two-hop neighbors. DSCS is carried out after the required information is collected. Then each node computes the absolute angle of each neighbor and arranges them in ascending order of their absolute angles. Based on both informations (one- and two-hop neighbors and their absolute angles), the node can decide to be boundary or not. However, these algorithms require a lot of information (one- and two-hop neighbors) which require extensive communication and high energy consumption. In addition, the algorithms are not accurate, i.e., 10% of the calculated boundary nodes are false. Both are evaluated and implemented in MATLAB which is not a realistic platform for WSNs.

The work of [25] presents a theoretical analysis method for boundary node detection based on localized Voronoi polygons. This method originates from computational geometry in the sense that each sensor node in the network tests if it is on the coverage boundary by calculating *Voronoi polygons* related to itself and based on the range of sensing and that of communication.

The study of [26] is focused on the detection of an external border of WSN by using a mechanism of identification of trigger nodes. The idea of the algorithm is to initiate the process by identifying one or more boundary nodes. These nodes of the network are located at the farthest distance from the sink. Each

node calculates its distance from both the sink and its neighbors. The node that is the farthest from the sink represents a boundary node which is used as trigger to discover the external boundary. Then, this node uses a geometric method to detect the next boundary nodes to the left and the right of it. However, this algorithm is not adapted to different types of topologies and it assumes that the sink is in the network exterior which is not always realistic.

The authors in [27] propose a topological approach for the detection of the boundaries of holes and the external boundary of a sensor network. This approach is based on the identification and checking of the connectivity of the x-hop neighbors surrounding every node and it includes three steps: information collection, path construction and path checking. In the first step, each node identifies the list of its x-hop neighbors. In the second step, the connectivity of the x-hop neighbors is used to produce a communication path around each node. These paths are checked in a third step in order to detect the boundary and the inner nodes, i.e., when the path is closed, the node is an inner one, else if the path is broken, other tests are necessary. This algorithm suffers from the choice of the factor x, as the factor x is given a high value, which means higher communication overheads and thus increased energy consumption. In addition, if the factor x is given a low value, the accuracy of the algorithm decreases.

The work of [28] is focused on the perimeter detection using a decentralized algorithm in wireless sensor networks based on the location neighborhood information combined with the Barycentric technique (used to determine whether a point is inside or outside of a triangle ABC). The algorithm uses the location neighborhood information of three neighbor nodes of each node to form a triangle and determines if it is enclosed by them and therefore not a perimeter node. Otherwise, it is a perimeter node. This algorithm is not accurate and requires a high network density which means that it presents a high false rate to detect a perimeter node in a network with low density and also in a network with nodes having a small communication radius.

In [29], the authors propose a distributed algorithm to detect a subset of boundary nodes and they develop a greedy routing algorithm used to gather the information of boundary nodes at the sink node. They assume a starting node to be affected, if its sensed data crosses a threshold value Th. Then, the affected nodes execute the proposed distributed algorithm to detect the boundary nodes. The algorithm is divided into three phases. In the first phase, each affected node declares itself as a non-boundary node if its neighbors are all affected. Otherwise, if some of its neighbors are not affected, it computes the function f based on the number of its neighbors and the value Th. In the second phase, it broadcasts the value of f in order to give a vote of other nodes. At the final phase, each node identified as a boundary node checks whether it has a smallest f, case in which it remains a boundary node, otherwise it becomes an inner node. However, this algorithm requires a high network density, it is not very accurate and it consumes high energy due to the communication with n-hop neighbor nodes.

We have observed that some of these algorithms work with planar networks and that they do not take into

account the crossings between edges which can lead to blocking situations arising from specific subgraphs or to cycles producing an infinite loop. We have illustrated these situations in [1] and shown how to avoid them. We have also observed that in general they do not give the optimal boundary in terms of the number of nodes. Table 1 summarizes 8 boundary determination algorithms in terms of message complexity (number of exchanged messages per node) and accuracy which represents the percentage of the real boundary nodes, where n is the number of nodes, n_b is the number of the boundary nodes, k is the number of nodes which execute simultaneously local computation, d is the maximum degree of the network, h the size of the table of the 3-hop neighbors and h_{max} is the maximum hop-count of a node which is given by the user. We added some observations on the communication overhead in the last column. We have compared our algorithm with only those for which the energy consumption is provided in their paper. Another comparison based on the energy consumption is presented in Section 5.4.

Table 1: Comparison with existing algorithms.

Algorithm	Message complexity	Accuracy	Observations
ABBD [16]	$O(n/k + n_b/k)$ = number of rounds $O(kd^3)$ = number of messages per round	$\simeq 88\%$	Communication overhead
EBDG [29]	$O(d\ h_{max})$	$\simeq 92\%$	Requires a dense network
LVP [25]	$O(n\ d)$	$\simeq 100\%$	Computation overhead
PDiscovry [28]	$O(n\ d\ h)$	$\simeq 95\%$	Communication overhead
DBD [19]	$O(n(d^3 + d^2 + d))$	$\simeq 90\%$	Requires 3-hop neighbors' information
DBNS [18]	$O(n/k)$ = number of rounds $O(k\ d)$ = number of messages per round	$\simeq 100\%$	Required to remove redundant nodes to form a complete boundary
Hop-based [27]	$O(n\ d\ h_{max})$	3% to 99%	Depends on the list of x-hop neighbors. Communication overhead
D-LPCN	$O(d\ n_b + n)$	$= 100\%$	Communication and energy are reduced ; only the boundary nodes and their neighbors are used.

3. LPCN Algorithm

Since the proposed algorithm D-LPCN represents the distributed version of LPCN, we will briefly present in this section the LPCN algorithm used to find the polygon hull in a connected Euclidean graph that we have presented in [1, 30] and where we have proven its optimality and its convergence. This problem can be formulated as follows. Given an undirected graph $G = (V, E)$, where $V = \{v_0, v_1, ..., v_{n-1}\}$ is the set of vertices of the graph and $E = \{e_0, ..., e_{m-1}\}$ its set of edges, the objective is to determine a smallest closed polygon circumscribing all the vertices of G. In other words, the goal is to find a finite and minimal set of vertices, able to contain all other vertices within the envelope formed by them. This envelope is called a Polygon Hull which is defined by two sets: a set of vertices $\mathbb{B}_V$ in a polygon of G and a set of edges $\mathbb{B}_E$ between the vertices $\mathbb{B}_V$ that are defined as follows:

$$\mathbb{B}_V = \{v_0, v_1, ..., v_{i-1}, v_i, v_{i+1}, ..., v_{h-1}\} \subseteq V,$$

$$\mathbb{B}_E = \{\{v_0, v_1\}, \{v_1, v_2\}, \cdots, \{v_{h-1}, v_h\}\} \subseteq E$$

with $v_0 = v_h$.

The algorithm starts from a vertex v with minimum x-coordinate, and determines the vertices v_1, ..., v_{h-1} of the polygon hull, where h is the number of vertices of the polygon hull and v_0 is a vertex with minimum x-coordinate. In each iteration i of the algorithm, the next vertex v_{i+1} in the polygon hull is determined by the vertex v_i that has the minimum angle $\varphi_{min}(v_{i-1}, v_i, v_{i+1})$ formed by the edges $\{v_i, v_{i-1}\}$ and $\{v_i, v_{i+1}\}$ (cf. Figure 1), where v_i is the vertex of the polygon hull selected in the current iteration i and v_{i-1} is the vertex of the polygon hull found in the previous iteration $i-1$. This algorithm covers some problematic situations whose treatment is illustrated in our paper [30]. One of these situations is given by a pair of intersecting edges in the graph G. By definition of a polygon hull, no two edges of a polygon hull should intersect. Whenever such a crossing is detected by the algorithm, it will be eliminated as follows:

Intersecting edges: we require that two edges of the polygon hull must not intersect outside the endpoint corresponding to the next vertex v_{i+1} chosen by the algorithm. As an example, Figure 2(a) shows that accepting the intersecting edges $\{A, B\}$ and $\{E, F\}$ of the polygon hull will lead to non-visited vertices on the polygon hull, which are I , J and G. However, if this intersection is considered then, as shown by Figure 2(b), all the vertices of the polygon hull are visited (i.e., A, B, C, D, E, G, H, I, J). In addition, since the next chosen point is B, this intersection must not consider the endpoint B of the edge $\{D, B\}$. That is to say, if we have two edges $\{A, B\}$ and $\{C, D\}$ and if the intersection with B results in one of the vertices A, B, C or D then it will not be considered as an intersection. This situation is justified by Figures 2(c) and (d). In the first one, if we accept the normal intersection $\{B, C\} \cap \{D, B\} = \{B\}$, the algorithm will choose a vertex which is different from B. Then, it will choose the next vertex C, which will lead to an infinite loop B, C and D. However, if we consider intersection without the endpoints of the edges, then

when the algorithm comes back to B, no intersection is detected and the algorithm will choose vertex A as it is shown by Figure 2(d).

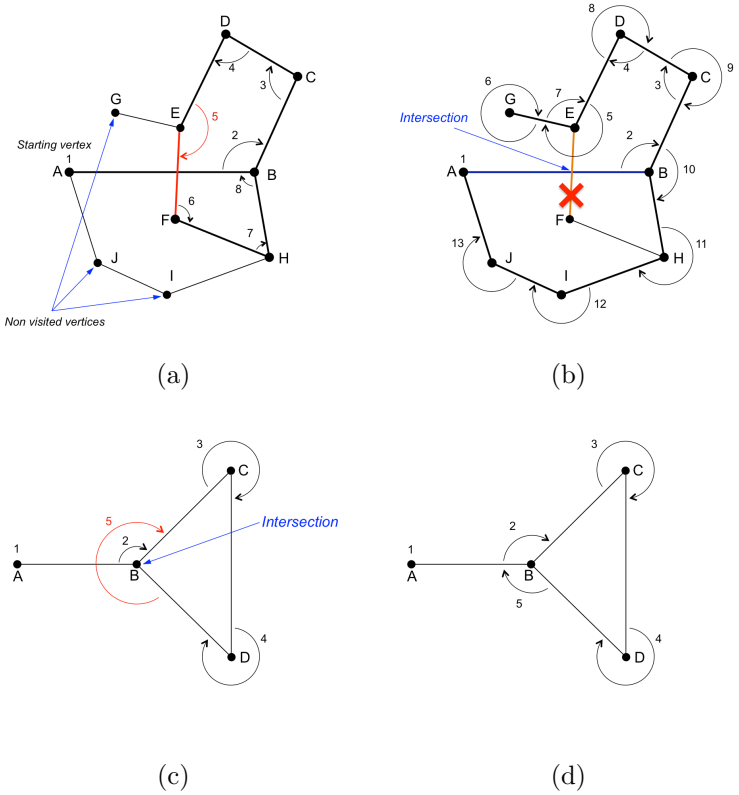


Figure 2: Edges' intersection.

The pseudo-code of the LPCN algorithm is given by Algorithm 1. First, it starts from a vertex having the minimum x-coordinate which will be a part of the solution (lines 2 and 3). In each iteration, of the repeat loop section that starts from line 7, the algorithm will search for the next vertex v_{min} of the polygon hull that will follow the currently found vertex v_c of the current polygon hull $\mathbb{B}_V$. This is done by calculating the angles formed by the edge $\{v_c, v_p\}$ and all edges $\{\{v_c, v\}/v \in \mathbb{A}\}$, where v_p is the previously found vertex and $\mathbb{A}$ is the set of neighbors of v_c which verify the condition that each edge $\{v_c, v\}$ with v element of $\mathbb{A}$ does not intersect the currently found polygon hull $\mathbb{B}_E$.

4. D-LPCN Algorithm

In this section, we will present the main steps of the proposed distributed algorithm for the discovery of the boundary nodes in a WSN. First, we introduce some definitions and primitives. Then, we present our Start-node algorithm allowing to launch the proposed D-LPCN algorithm, which will be presented thereafter.

Algorithm 1 LPCN Algorithm

procedure LPCN(V, E)
  $v_c \leftarrow$ vertex having the minimum x-coordinate
  $\mathbb{B}_V \leftarrow [v_c]$
  $\mathbb{B}_E \leftarrow \emptyset$
  $v_{first} \leftarrow v_c$
  $v_p \leftarrow$ fictive vertex situated in the left of v_c
  repeat
    $\mathbb{A} \leftarrow \{v \in N(v_c)/\mathbb{B}_{\mathbb{E}} \cap \{\{v_c, v\}\} = \emptyset\}$
    $v_{min} \leftarrow \underset{v \in \mathbb{A}}{\operatorname{argmin}}\{\varphi(v_p, v_c, v)\}$
    $\mathbb{B}_V \leftarrow \mathbb{B}_V \cup \{v_{min}\}$
    $\mathbb{B}_E \leftarrow \mathbb{B}_E \cup \{\{v_c, v_{min}\}\}$
    $v_p \leftarrow v_c$
    $v_c \leftarrow v_{min}$
  until $v_{min} = v_{first}$
  return $\mathbb{B}_V, \mathbb{B}_E$
end procedure

4.1. Definitions and Primitives

We assume that the communication between two sensor nodes is symmetrical. In this case, a WSN can be modeled as an undirected graph $G = (S, E)$, where $S = \{s_1, s_2, ..., s_n\}$ is the set of sensor nodes, $n =\mid S \mid$ their total number and E the set of communication links. The link between two nodes can be defined as follows:

$$e_{ij} = \begin{cases} 1 & if\ the\ node\ s_i\ communicates\ with\ s_j \\ 0 & otherwise \end{cases} \tag{1}$$

The neighbor nodes $N(s_i)$ of a given node s_i are the nodes that communicate with it.

$$N(s_i) = \{s_j / e_{ij} = 1,\ \ j = 1, ..., n\ \ and\ \ j \neq i\} \tag{2}$$

For a better understanding of the proposed algorithm, we define in Table 2 some message primitives and their definitions and in Table 3 the functions used in the algorithm. Note, that the proposed algorithm works in the case of bidirectional communication in order to obtain the optimal boundary nodes and every node knows its location in space in terms of (x, y) coordinates.

Table 2: Message primitives and their definitions

Primitive	Definition
AC	ask for coordinates
CS	send coordinates
SN	select a node

Table 3: Functions of the D-LPCN algorithm

Function	Definition
`getId()`	returns the node identifier
`getCoord()`	returns the node coordinates (x, y)
`getNumberOfNeighbors()`	returns the number of neighbors of the node
`send(`a`,` b`)`	sends the message a to the sensor node having the identifier b, or in a broadcast ($b = *$)
`read()`	waiting for receipt of messages

4.2. Determining the Starting Node

In this section, we will present a distributed algorithm that allows to determine a node with minimum x-coordinate in a network. This algorithm is based on the Minimum Finding algorithm presented in [31, 32] which relies on the tree-based broadcast algorithm. The aim of this algorithm is to determine the starting node of the D-LPCN algorithm which will be presented in Section 4.3. The principle of the Start-node algorithm can be described as follows. First, each node of the network determines locally its x-coordinate and assigns it to the variable x_{min} assumed to represent the minimum x-coordinate of the network. Then, it will broadcast it and wait for other x_{min} values coming from its neighbors. If a received value x_{min} is less than its local x_{min} value then this one will be updated and broadcasted again. This process is repeated by each node as long as a received value is less than its local x_{min} value. After a certain time t_{max}, there will be only one sensor node that has not received a value that is less than its local x_{min} value. This node is at the extreme left of the network. It will be considered as the starting node of the D-LPCN algorithm. The pseudo-code of this process is given by Algorithm 2, where t_0 is the time of the first execution of the algorithm, which can correspond to the first powering-on of a sensor node, t_c the current local time of a sensor node, and t_{max} the maximally tolerated running time of the algorithm from the first execution to the current time of a sensor node.

Algorithm 2 *MinFind*: The pseudo-code of determining the starting node.

Input: t_{max}

```
1: first_node = TRUE;
2: t_0 = getCurrentTime();
3: x_min = getX();
4: send(x_min, *);
5: repeat
6:     x = read();
7:     if (x < x_min) then
8:         first_node = FALSE;
9:         x_min = x;
10:        send(x_min, *);
11:    end if
12:    t_c = getCurrentTime();
13: until (t_c − t_0 > t_max)
```

Calculating the time complexity of this *MinFind* algorithm will help to set the value of t_{max}. To do this, let us consider a linear network with n nodes representing the worst case. The node with minimum x-coordinate, which is in the extreme left, will send only 1 message and will receive only 1 message. However, the node which is in the extreme right will receive $n-1$ messages and will send $n-1$ messages to send the received and assumed x_{min} coordinate, because it is the node with the largest x-coordinate, and thus, each of the other nodes, except the extreme left one, has at least one node on its left.Therefore, these nodes will systematically send a message to broadcast the newly received x_{min}. Altogether, the message complexity is equal to $M[MinFind] = 2(n-1) = 2n-2$ and if we consider that a sensor can send and receive messages at the same time (full-duplex communication), the time complexity is equal to $T[MinFind] = n-1$. This complexity is reduced in the case where the network is not linear. Since the time complexity is known, it is possible to estimate the value of t_{max}, the required time to find the starting node. As an example, for a network with 100 nodes, where the size of each message is equal to 1024 bits, sampled with a frequency rate of 250 kb/s (case of the 802.15.4 standard) we need 406 ms to find the starting node. Using the CupCarbon simulator, we have simulated two networks with 100 sensor nodes. The first one is linear (cf. Figure 3) and the second one is random (cf. Figure 4). The simulation results show that the starting node is obtained in 406 ms with a consumption of $1J$ to $9J$ per node for the linear network and in 70 ms with a consumption of $1J$ to $5J$ per node for the second one. In this simulation, the serialization of messages from the microcontroller to the radio module is neglected. However, if the serialization time is taken into account and if we assume that it is equal to 38400 b/s then to determine the starting node, we need 1.5 s

for the linear and 190 ms for the random network. Determining an accurate estimator of the value of t_{max} in the case of random networks could be a topic for future work.

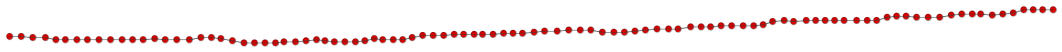


Figure 3: A linear network with 100 sensor nodes.

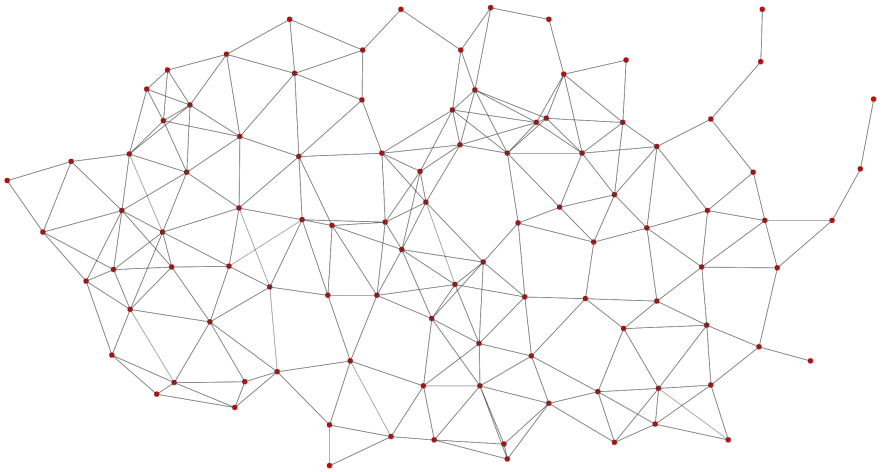


Figure 4: A random network with 100 sensor nodes.

4.3. The D-LPCN algorithm

The D-LPCN algorithm uses the same principle as LPCN presented by Algorithm 1 with the exception that the program is written in a distributed form. This means that each sensor node will execute its own program, and by communicating with its neighbors, it can decide if it is a boundary node or not. Algorithm 3 shows the main steps of the D-LPCN algorithm that can be described as follows:

- *Step 1 (lines 1 to 4)*: Initialization.
- *Step 2 (line 5)*: Run Algorithm 2 to determine whether the current node is a starting node or not.
- *Step 3 (lines 6 to 10)*: If the current node is a starting node than it launches the D-LPCN algorithm by broadcasting the "AC" message to its neighbors in order to ask for their coordinates.
- *Step 4 (lines 12 to 13)*: Each node is waiting to receive a message.

- *Step 5 (lines 14 to 17)*: The variable i determines the number of received "CS" messages. Receiving n "CS" messages means that the coordinates of all neighbors have been received.

- *Step 6 (lines 18 to 20)*: If the node receives an "AC" message then it will send a "CS" message containing its coordinates to the transmitter having the identifier *id*.

- *Step 7 (lines 21 to 27)*: For all received "CS" messages, the node calculates the minimum angle formed with them by taking into account the intersections with the received boundary set (in line 15). This calculation is done using the function angleWI (angle without intersections).

- *Step 8 (lines 28 to 23)*: The reception of an "SN" message by a node means that this node has been selected as a boundary node by its previous boundary neighbor node. This node will then restart the process of finding the next boundary node by broadcasting a "CS" message.

Finally, Algorithm 3 stops when the first boundary node is selected a second time with an "SN" message. This algorithm, as it is presented, will continue to determine the boundary nodes continually. If we want to add the stop condition, we have just to add the following code after line 28:

if (first_node) **then**
  STOP;
end if

To better explain how this algorithm works, we will use the example of Figure 5 which represents a WSN with eight sensor nodes. Let us consider the set $S = \{S1, S2, ..., S8\}$ of these nodes and the set B of the boundary nodes, which initially is empty.

First, after the initialization (*Step 1*), we run Algorithm 2 (*Step 2*). The only node which will be considered as the starting node is $S1$, and thus the boundary set is updated to $boundary\_set = \{S1\}$.

Next, the node $S1$ broadcasts an "AC" message to its neighbors $N(S1) = \{S2, S3, S4, S7\}$ (*Step 3*) to ask for their coordinates (cf. Figure 5(a)) while the other nodes are waiting for the receipt of messages (*Step 4*).

Next, each neighbor node $S2, S3, S4$ and $S7$, which receives the "AC" message, sends a "CS" message (*Steps 5 and 6*) to the boundary node $S1$ (cf. Figure 5(b)) which will calculate the angle formed by the fictive node $S1'$ with itself and with each of its neighbor nodes $S2, S3, S4$ and $S7$ (*Step 7*). This situation is illustrated by Figure 5(c).

Then, as shown by Figure 5(d), the boundary node $S1$ will send an "SN" message to the new boundary node $S3$ which will update its boundary set to $boundary\_set = \{S1\}$ (*Step 8*). The obtained situation is shown by Figure 5(e).

The next boundary node $S3$ will then perform the same procedure from *Step 2* on. Figure 5(f) shows the final iteration of the D-LPCN algorithm.

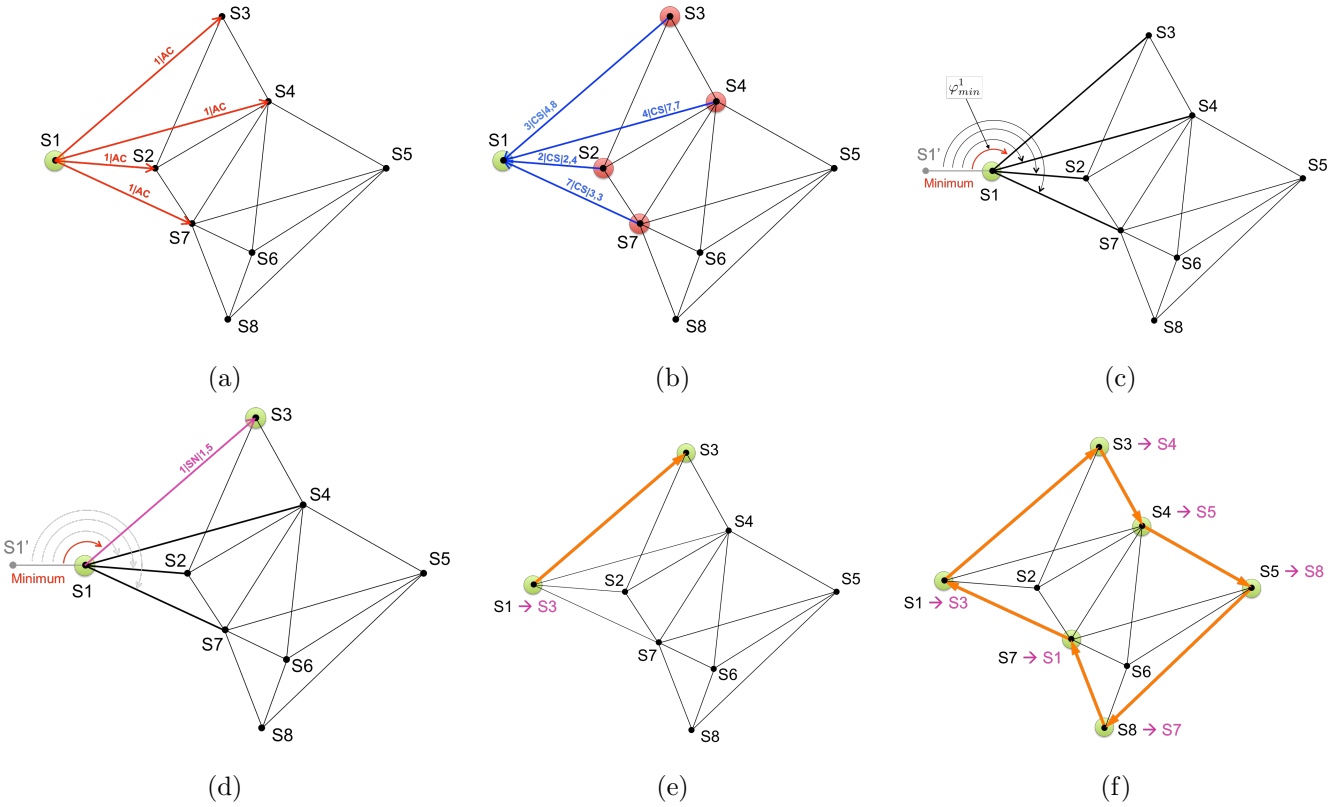


Figure 5: An example for the D-LPCN algorithm.

Altogether, the pseudo-code of D-LPCN is given by Algorithm 3.

5. Implementation and Simulation Results

In this section, we give a brief description of simulators and platforms that we have used to simulate and to implement the proposed algorithm. In particular, we will present some simulation results in terms of energy consumption. The remainder is then divided into three main parts. The first part is dedicated to present the simulation of the D-LPCN algorithm under the CupCarbon simulator. The second part is dedicated to its real implementation on the Arduino/XBee and TelosB platforms. This part allows to validate the proposed algorithm. In the last part, we will study the energy consumed by the proposed algorithm.

5.1. Simulation

To validate the proposed D-LPCN algorithm, we have used three simulators: CupCarbon [33], Tossim [34], and Contiki OS [35].

Algorithm 3 The D-LPCN algorithm

```
1: boundary = false; phi_min = 10;
2: c_id = getId(); c_coord = getCoord();
3: boundary_set = ∅;
4: n = getNumberOfNeighbors(); i=0;
5: Run Algorithm 2 to determine the value of first_node;
6: if (first_node) then
7:     boundary = true;
8:     p_coord = (c_coord.x−1, c_coord.y);
9:     send(c_id+"|"+"AC", *);
10: end if
11: repeat
12:     id = read();
13:     type = read();
14:     if (i==n) then
15:         boundary_set = boundary_set ∪ {c_id};
16:         send(c_id+"|"+"SN"+"|"+c_coord+"|"+boundary_set, n_id);
17:     end if
18:     if (type=="AC") then
19:         send(c_id+"|"+"CS"+"|"+c_coord, id);
20:     end if
21:     if (type=="CS") then
22:         n_coord = read(); i=i+1;
23:         phi = angleWI(p_coord, c_coord, n_coord, boundary_set);
24:         if (phi<phi_min) then
25:             phi_min = phi; n_id = id;
26:         end if
27:     end if
28:     if (type=="SN") then
29:         boundary = true; phi_min = 10; i=0;
30:         p_coord = read();
31:         boundary_set = read();
32:         send(c_id+"|"+"AC", *);
33:     end if
34: until false
```

CupCarbon [36] is a Smart City and Internet of Things simulator. Its objective is to design, visualize, debug and validate distributed algorithms for monitoring, environmental data collection, and to create environmental scenarios, generally within educational and scientific projects. CupCarbon offers two simulation environments. The first simulation environment is a multi-agent environment [33], which enables the design of mobility scenarios and the generation of events such as fires and gas as well as the simulation of mobiles such as vehicles and flying objects [37]. The second simulation environment represents a discrete event simulation of wireless sensor networks which takes into account the scenario designed on the basis of the first environment. It allows to generate code for a real Arduino/XBee platform from the simulation.

Tossim is a discrete event simulator for TinyOS sensor networks [34]. It can simulate the behavior of a sensor within a WSN and upload programs ready to be integrated directly into TelosB [38] sensor nodes. In the same way, Contiki-OS [35] is an open source and portable operating system designed specifically for resource limited devices such as sensor nodes. It brings the benefits of both events and thread execution models. It also supports a full TCP/IP stack via uIP and the programming abstraction "Protothreads". The main reason for using Contiki in our work is to be able to accurately measure the energy consumed by real sensor nodes when executing the D-LPCN algorithm. This option is done using the *Powertrace* tool [39].

The red nodes of Figure 6, highlighted by the arrows, show the nodes with the minimum x-coordinate in each sub-network of a network containing several connected components. This result is obtained by Algorithm 2 (Minimum Finding algorithm) presented in Section 4.2, which constitutes the necessary step before running the D-LPCN algorithm. The boundary nodes, highlighted in yellow, are found by Algorithm 3 (D-LPCN algorithm) presented in Section 4.3 which starts from the red nodes determined in the previous step.

Figure 7 shows an example of detecting boundary nodes using the ring topology. The boundary nodes are represented in yellow. The red nodes represent the neighbors of the boundary nodes. In this example, the algorithm is executed periodically for instance to recalculate the new boundary if some of the boundary nodes fail. As shown by Figures 7(b) and (c), a new boundary is found after injection of faulty nodes. Figure 7(c) also shows that a ring topology is useful for monitoring a sensitive site since the D-LPCN can continue to work on the basis of the inner nodes even if a complete part of the ring is broken or destroyed.

5.2. Implementation

Once the algorithm validated by simulation, we have implemented it using two real WSN platforms. Figure 8 shows the first WSN which is based on the Arduino cards [40] and the XBee (Series 1) radio module working with the 802.15.4 protocol. Figure 9 shows the second WSN which is based on the TelosB sensor nodes. In each figure, the boundary nodes are represented by the nodes with switched-on leds that are designated by arrows.

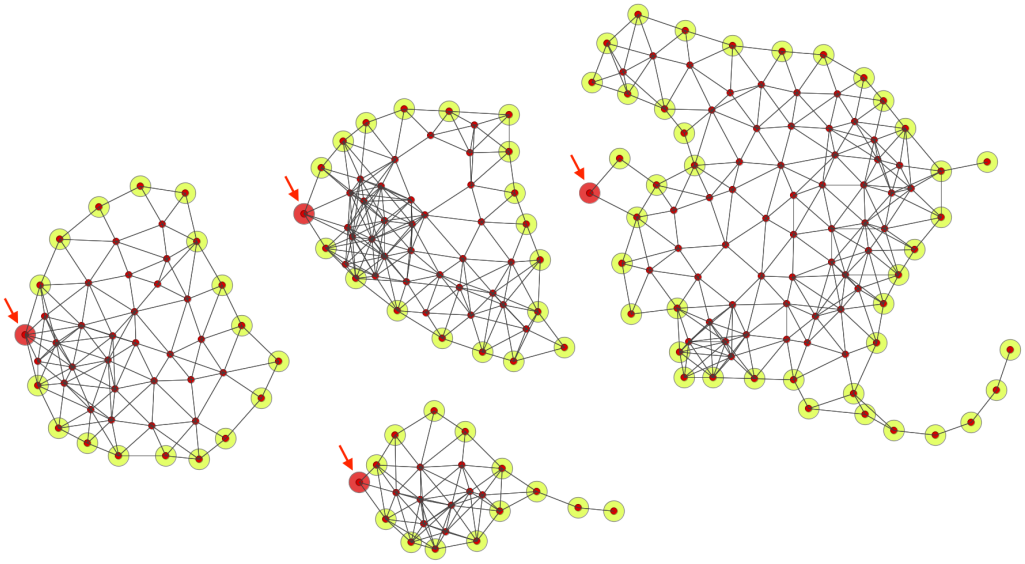


Figure 6: CupCarbon simulation of Minimum Finding and D-LPCN algorithms for a disconnected network.

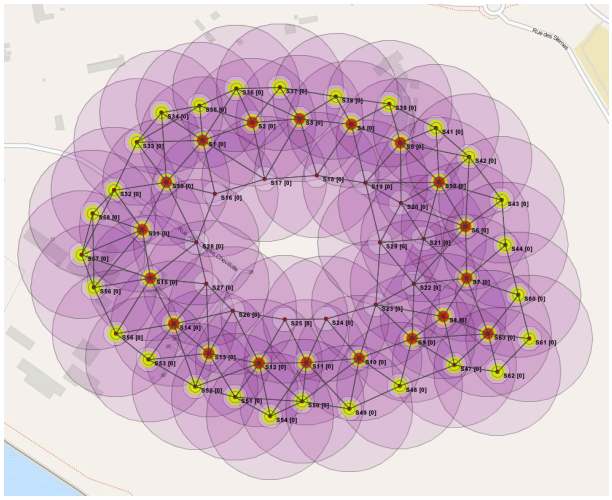
5.3. Energy consumption

To estimate the energy consumption of the proposed algorithm, we use in this paper the energy model of the TelosB sensor nodes which is described in [41]. Its energy consumption is estimated to $59.2\mu J$ for the transmission of one byte and to $28.6\mu J$ for the reception of one byte. To profile the power consumed by the sensor nodes, we have used the *Powertrace* tool [39] which estimates the energy consumption with an accuracy very close to the real one. Experiments in [35] show that the energy consumption obtained by the *Powertrace* tool are very close with 94% to the energy consumption of a real device. *Powertrace* calculates the power consumption of the local node based on the monitoring of the power state. Then, this value is encapsulated according to different activities like reception or transmission of packets, computation, idleness, etc. The energy consumption is computed in mW as follows:

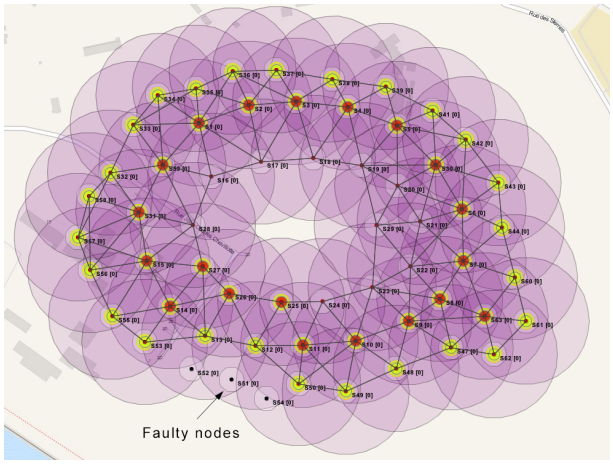
$$E = \frac{Energest\_Value \times Current \times Voltage}{RTIMER\_SECOND \times Runtime} \tag{3}$$

where $Energest\_Value$ is the difference between the number of ticks in two time intervals (for example difference between CPU in time interval 10 and 20), $Current$ is equal to $17.4mA$ for packet reception, to $18.8mA$ for packet transmission and to $0.33mA$ for CPU operations (computing). The variable $Voltage$ is set to $3V$ and $RTIMER\_SECOND$ is set to $32KHz$ which represents the number of ticks per second and which is predefined in the RTIMER-library of the Contiki software. It provides a scheduling and execution of real-time tasks (with predictable execution times).

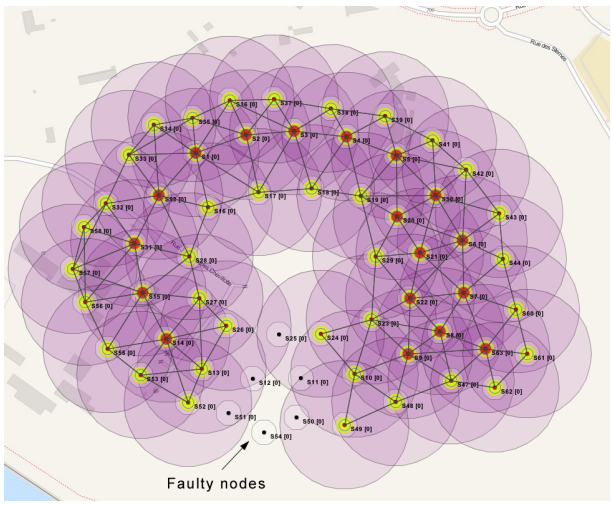
We use Equation (3) to study the energy consumption of the proposed algorithm. To this end, we have



(a)



(b)



(c)

Figure 7: Simulation of D-LPCN using the CupCarbon simulator.

Figure 8: Execution of D-LPCN using real Arduino/XBee sensor nodes.

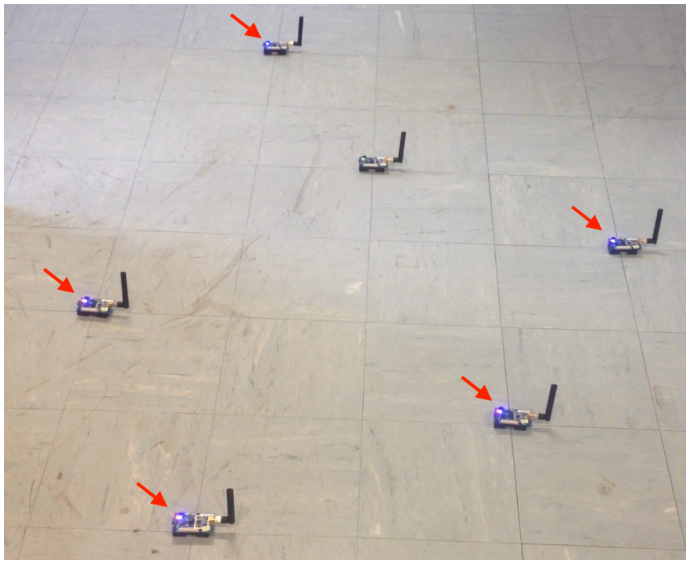


Figure 9: Execution of D-LPCN using real TelosB sensor nodes.

considered two main cases to estimate the energy consumption of a boundary node. In the first case, we consider a boundary node which is connected to only two boundary nodes and to other non-boundary nodes. In the second one, we consider a boundary node that is connected only to boundary nodes. Figure 10 shows the first case where a boundary node S_1 is connected to two boundary neighbor nodes, to zero such nodes, one (S_7), or many other non-boundary neighbor nodes ($S_7, ..., S_m$).

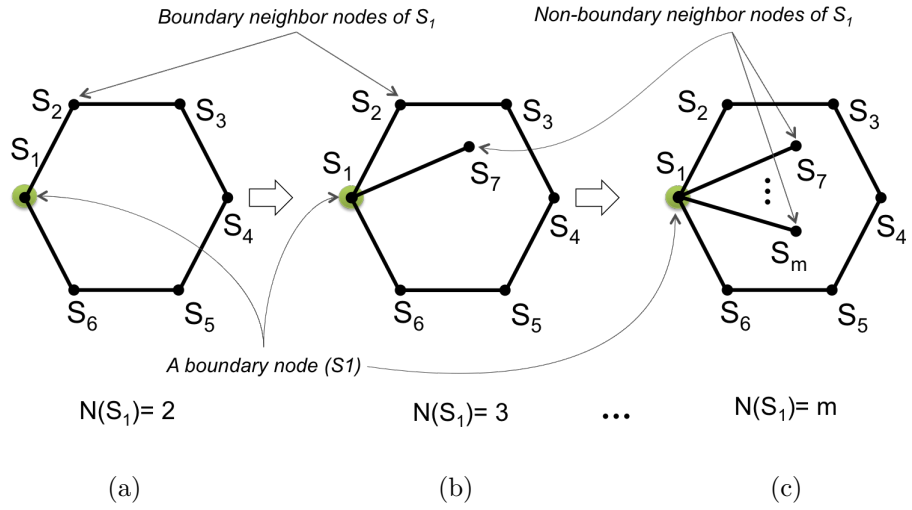


Figure 10: A boundary node S_1 connected to two boundary nodes (S_2 and S_6) and to non-boundary nodes (S_7), ($S_7, ..., S_m$).

Figure 11 shows the evolution of the energy consumption of a boundary node S_1 as a function of the number of its neighbors including two boundary neighbor nodes and other non-boundary neighbor nodes. We consider the case where this boundary node S_1 is connected only to two boundary nodes, i.e., it is not connected to any other non-boundary neighbor node (cf. Figure 10(a)). Then, we observe from the first marker of the curve of Figure 11 that the energy consumed by node S_1 when running continually the D-LPCN algorithm is equal to $8.6\mu J$ each second. If we assume that this node has a source of two Super Alkaline AALR6 batteries with a capacity of $9580J$ each (i.e., a total of $19160J$), we can conclude that this node consumes $45 \cdot 10^{-6}\%$ of energy each second. This means that the lifetime of this node is around 25 days. If we consider the case where a boundary node is connected to 10 non-boundary neighbor nodes (cf. Figure 10(c)), the energy consumption ($\simeq 12,000\mu J$) increases by $18 \cdot 10^{-6}\%$ (cf. the 9^{th} marker of the curve of Figure 11). This means that the lifetime of this node decreases by 18 days. Therefore, we can conclude that each additional 10 non-boundary neighbor nodes lead to a reduction of 7 days of the lifetime of a boundary node.

Now, if we consider the case where the neighbors of S_1 are only boundary nodes, too, as shown by Figure 12, then if the neighbor list table is determined by each sensor node, the situation is identical to the one explained above.

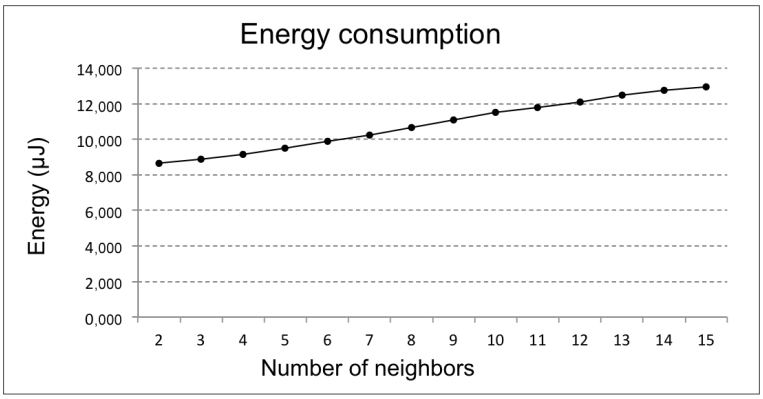


Figure 11: Energy consumption of a boundary node S_1.

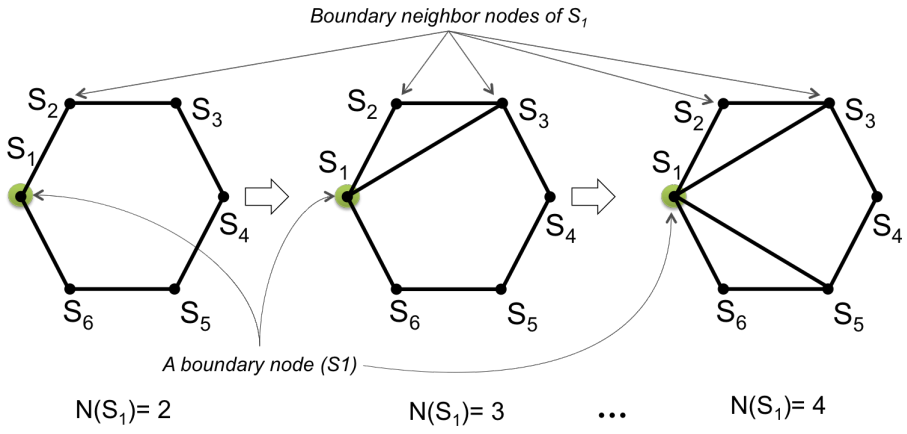


Figure 12: A boundary node S_1 connected to up to 4 boundary nodes (S_2, S_3, S_5, S_6).

However, if the neighbors must be determined periodically or before each angle calculation, as it is the case in Algorithm 3, then the consumption will increase. This situation is useful in order to take into account the possibility of faulty nodes (cf. Figure 7), but it is energy consuming. Figure 13 gives an illustration where we consider a network with 35 nodes including 17 boundary nodes (highlighted in yellow) and 11 nodes that are connected to the boundary nodes (highlighted in red). The remaining nodes are not connected to the boundary nodes.

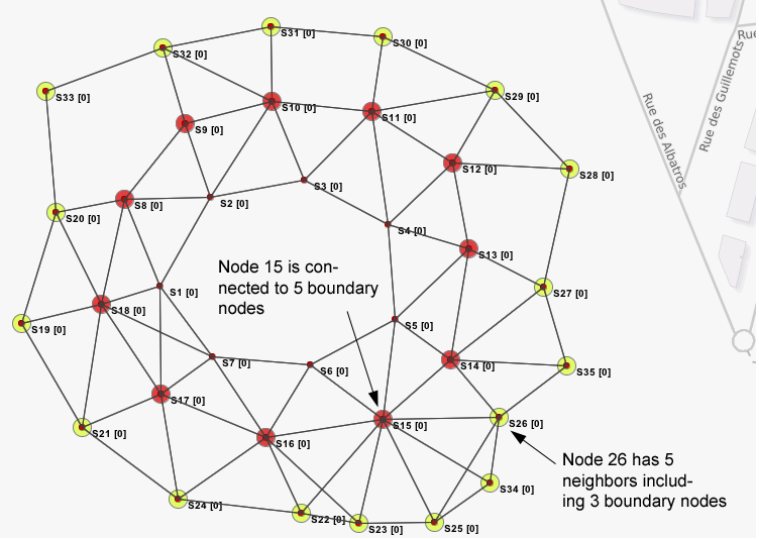


Figure 13: A Wireless Sensor Network with 35 nodes.

Figure 14 shows the energy consumption of these nodes. The green bars represent the energy consumption of the yellow nodes, the red bars represent the energy consumption of the red nodes and the black bars, which are equal to zero, represent the energy consumption of the other nodes. As we can see, node S_{15} is the node that consumes the maximum energy among the neighbors of the boundary nodes because it has the maximum number of boundary neighbors. We can also observe that even if this node is not a boundary node it consumes more energy than the boundary node S_{33}, for instance. In the same way, node S_{26} is the node that consumes the maximum energy among the boundary nodes because it has the maximum number of neighbors. Furthermore, if we compare the energy consumption of these two nodes, we find that S_{26} consumes more energy than S_{15} even though S_{15} has more neighbors. This is justified by the fact that the node S_{15} is only asked 5 times by its 5 boundary neighbors to send its coordinates. However, while the node S_{26} is asked to send its coordinates by only 3 boundary neighbors, it has to send a message to ask the other nodes for their coordinates and to send another message to choose the next boundary node. Moreover, this node has received a message from the previous boundary node saying that it has been chosen as a boundary node. Finally, the inner nodes that are not connected to the boundary nodes will consume a negligible

energy since they do not communicate with them. These nodes can be useful for a replacement of possibly faulty nodes, which ensures a real fault-tolerant wireless sensor network.

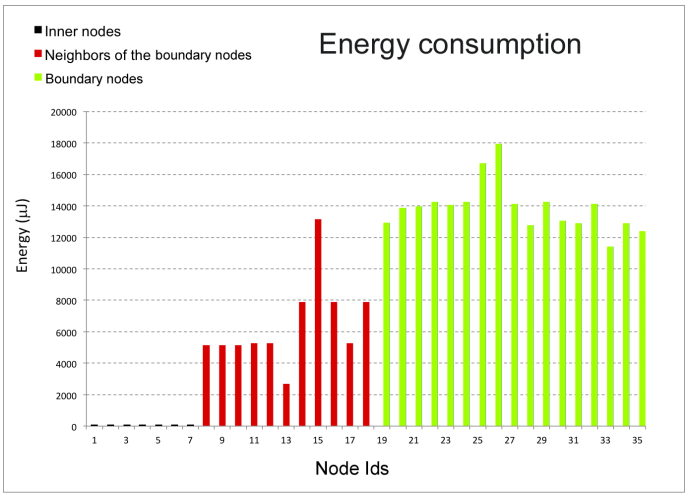


Figure 14: Energy consumption of the nodes from Figure 13.

We conclude that the energy consumption does not depend on the number of nodes of the network nor on the number of the boundary nodes, but instead depends on the number of the neighbors of the boundary nodes.

5.4. Comparison with existing methods

Figure 15 presents the energy consumption of the boundary nodes with respect to the number of their neighbor nodes given by some existing algorithms [18][24][26]. It shows that the energy consumption increases with the number of neighbor nodes (degree).

As we can see, the energy consumptions obtained by LVP, DBD, D-LPCN algorithms are close to each other (between $10mJ$ and $150mJ$). However, the energy obtained by the Hop-based algorithm is important (between $10,000mJ$ and $100,000mJ$). This is due to the number of hops required by this algorithm. The D-LPCN algorithm consumes less energy than all the others. This can be explained for the case of the DBD algorithm by the necessity to use 3-hop communications, in the case of the LVP algorithm by the important local computations (use of Voronoi diagrams) and for the case of the Hop-based algorithm by the important number of multi-hop communications. In addition, the D-LPCN algorithm uses only the boundary nodes and their neighbors. The other nodes remain stable in terms of energy.

6. Comparison with the centralized version

In this section we will compare the non-centralized version of the D-LPCN algorithm with the centralized one. The comparison will be done according to the message complexity (the number of messages exchanged

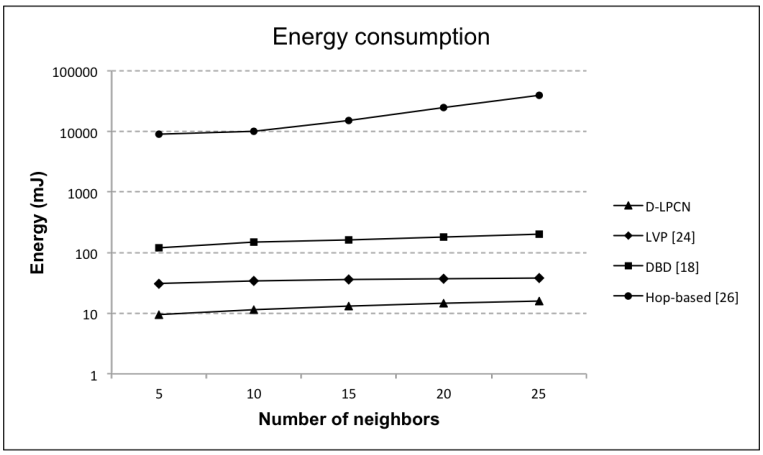


Figure 15: Comparison of the energy consumption with those of existing methods.

between nodes) and the energy consumption. The centralized algorithm is designed to work on a powerful central machine which is considered as a sink. Each sensor node will send its identifier and its position to the sink. For the case that the sensors cannot communicate directly with the sink, these informations will be sent over multiple hops. The sink will then run the classical LPCN algorithm [1] to find the boundary nodes and send the obtained solution to the nodes. The energy consumption of this version increases with the network size. The nodes close to the sink will be called often by the other sensor nodes, but then they will die quickly. For example, we can see in Figure 16 that the nodes S_{12} and S_{13} will be called for each message sent by any other node to the sink. Also, in this version, all sensor nodes are working.

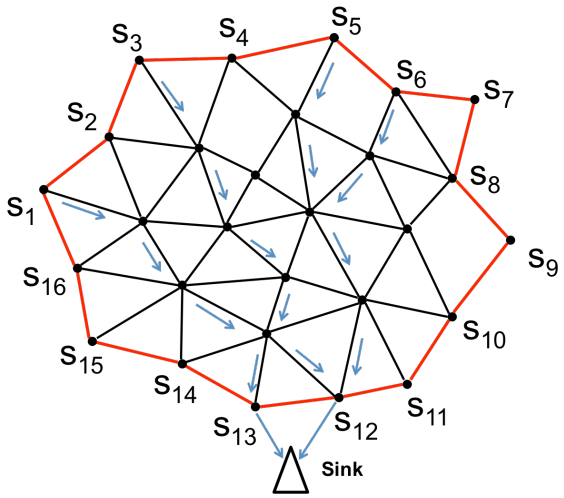


Figure 16: The centralized version of the D-LPCN algorithm.

In the distributed version, however, each node communicates only with its neighbors and takes decisions locally to determine the next boundary node. In this case, the sink is not required. As we can see in Figure 17 some nodes are not called at all. The only nodes that are necessary to run the algorithm are the boundary nodes and their neighbors. Therefore, the non-called nodes can be either removed or be used to replace possibly faulty nodes.

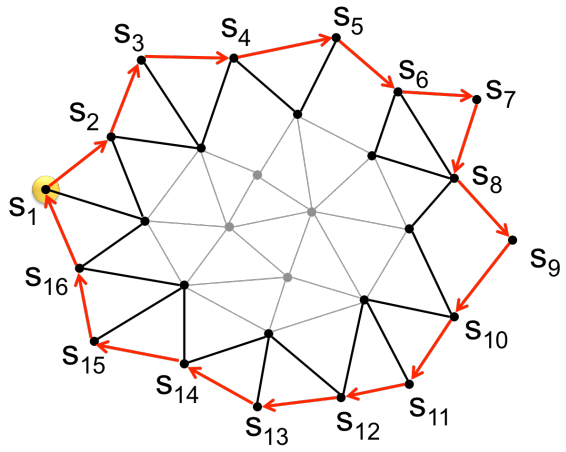


Figure 17: The distributed version of the D-LPCN algorithm.

Overall, we can conclude that the distributed version is more useful than the centralized one. The calculations that are done in each sensor node are not energy consuming. The energy consumption is significantly improved by reducing the communications between sensor nodes. According to [42], the energy required for transmitting a single bit could be used to execute 1000 to 2000 instructions. The complexity of the centralized version is equal to $O(ln^2)$ where n is the total number of sensor nodes, l the number of messages sent in multiple hops from the node s_i to the sink or from the sink to any other node. The square comes from the exchanged messages that can be done in both directions. However, the complexity of the distributed algorithm is $O(kh)$, where k is the maximum degree of the network and h the number of the boundary nodes. This complexity can be further improved by increasing the number of starting nodes. If this number is equal to m then the new complexity is given by $O(kh/m)$.

7. Other variants

The D-LPCN algorithm can be implemented with some minor modifications allowing to increase the speed of the algorithm which can be very useful in the case of applications where this parameter is important. We propose to start the algorithm from several starting nodes instead of just one, which all have to be boundary nodes. As shown by Figure 18, the network is divided into two parts and two starting nodes are chosen (S_1

and S_9). The algorithm stops when all starting nodes have been visited twice. The same process can be duplicated. Finally, it is possible to choose any number of starting nodes. However, we may end up with situations where the algorithm stops without finding the final boundary nodes.

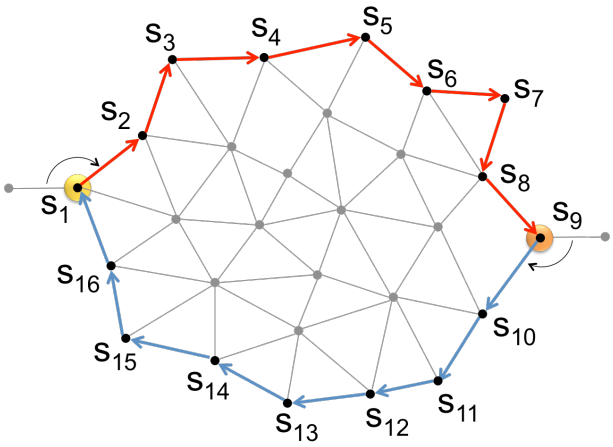


Figure 18: D-LPCN with two starting nodes.

Another variant of the D-LPCN algorithm starts with a boundary node from which two other boundary nodes will be chosen. As shown by Figure 19, the first one is chosen using the minimum polar angle (in a clockwise direction) and the second one is chosen using the maximum polar angle (which also represents the node with the minimum angle in the anticlockwise direction).

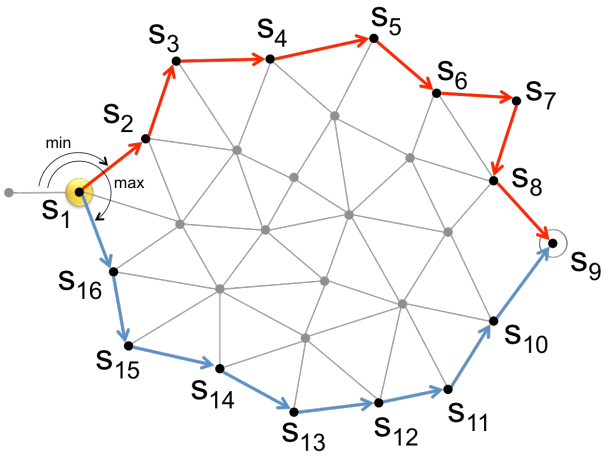


Figure 19: D-LPCN with two boundary nodes chosen from a starting node.

We conclude this section with the remark that starting from many nodes can improve the speed of finding the boundary nodes. However, as the stopping condition is based on visiting the starting node twice, it is

possible to get such a situation for all starting nodes without finding the final boundary nodes.

8. Conclusions and Future work

We have proposed a new algorithm called D-LPCN (Distributed Least Polar-angle Connected Node) in order to find a minimal set of connected boundary nodes in a Wireless Sensor Network. This algorithm represents the distributed version of the LPCN algorithm that we have presented in [1]. It chooses in each iteration and for each boundary node its nearest polar angle node with respect to the node found in the previous iteration. The main advantage of the proposed algorithm is that it works with any type of network, planar or non-planar, and also in the case where the network is composed of several connected sub-networks. Furthermore, it takes into account any blocking situation, which it is able to overcome. The proposed algorithm is validated using the CupCarbon, Tossim and Contiki simulators. It has also been implemented using real sensor nodes based on the TelosB and Arduino/XBee platforms. We have shown that the distributed version is less energy consuming than the centralized one. This is due to the important number of messages that are exchanged between the sink and the nodes in the centralized version, while in the distributed version only the boundary nodes and their neighbors are communicating. The simulation results also indicate that the proposed algorithm can be applied to very large sensor networks and it is quite optimal in terms of energy consumption. The complexity of the proposed algorithm is $O(kh)$, where k is the maximum degree of the network and h the number of the nodes on the boundary. We are currently working on how to use the D-LPCN algorithm to detect holes in a WSN and how to route messages efficiently using only the boundary nodes.

9. Acknowledgment

This work is part of the research project PERSEPTEUR supported by the French Agence Nationale de la Recherche ANR.

References

[1] A. Bounceur, R. Euler, A. Benzerbadj, F. Lalem, M. Saoudi, T. Kechadi, M. Sevaux, Finding a polygon hull in wireless sensor networks, in: European Conference on Operational Research, University of Strathclyde, Glasgow, UK, Invited talk, EURO 2015, July 2015.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, Wireless sensor networks: a survey, Computer Networks 38 (4) (2002) 393–422.

[3] M. Saoudi, A. Bounceur, R. Euler, T. Kechadi, Data mining techniques applied to wireless sensor networks for early forest fire detection, in: International Conference on Internet of things and Cloud Computing (ICC), 2015, ACM, 2015, pp. 71–77.

[4] L. Sitanayah, A. Datta, R. Cardell-Oliver, Heuristic algorithm for finding boundary cycles in location-free low density wireless sensor networks, Computer Networks 54 (10) (2010) 1630–1645.

[5] T. Le Dinh, Topological boundary detection in wireless sensor networks., JIPS 5 (3) (2009) 145–150.

[6] K. R. Gabriel, R. R. Sokal, A new statistical approach to geographic variation analysis, Systematic Biology 18 (3) (1969) 259–278.

[7] J. Xu, H.-Y. Qian, Localization of wireless sensor networks with a mobile beacon, Information Technology Journal 12 (11) (2013) 2251.

[8] N. A. Alrajeh, M. Bashir, B. Shams, Localization techniques in wireless sensor networks, International Journal of Distributed Sensor Networks 2013.

[9] M. B. Alrajeh, Nabil Ali Bashir, Localization techniques in wireless sensor networks, International Journal of Distributed Sensor Networks.
URL http://dx.doi.org/10.1155/2013/304628

[10] A. Laouid, H. Moumen, K. Chait, A distributed localization protocol for wireless sensor networks, in: Computer and Information Technology (WCCIT), 2013 World Congress on, IEEE, 2013, pp. 1–5.

[11] B. Huang, W. Wu, G. Gao, T. Zhang, Recognizing boundaries in wireless sensor networks based on local connectivity information, International Journal of Distributed Sensor Networks 2014.

[12] B. Huang, W. Wu, T. Zhang, An improved connectivity-based boundary detection algorithm in wireless sensor networks, in: In 38th IEEE Conference on Local Computer Networks (LCN), IEEE, 2013, pp. 332–335.

[13] S. Funke, C. Klein, Hole detection or: how much geometry hides in connectivity?, in: Proceedings of the twenty-second annual symposium on Computational geometry, ACM, 2006, pp. 377–385.

[14] K. Luthy, E. Grant, N. Deshpande, T. C. Henderson, Perimeter detection in wireless sensor networks, Robotics and Autonomous Systems 60 (2) (2012) 266–277.

[15] P. K. Sahoo, K.-Y. Hsieh, J.-P. Sheu, Boundary node selection and target detection in wireless sensor network, in: IFIP International Conference on Wireless and Optical Communications Networks (WOCN'07), IEEE, 2007, pp. 1–5.

[16] S. Shukla, R. Misra, Angle based double boundary detection in wireless sensor networks, Journal of Networks 9 (3) (2014) 612–619.

[17] C. Lara-Alvarez, J. J. Flores, C.-C. Wang, Detecting the boundary of sensor networks from limited cyclic information, International Journal of Distributed Sensor Networks, Hindawi Publishing Corporation, Volume 2015, Article ID 401838.

[18] P. K. Sahoo, J.-P. Sheu, K.-Y. Hsieh, Target tracking and boundary node selection algorithms of wireless sensor networks for internet services, Information Sciences 230 (2013) 21–38.

[19] W.-C. Chu, K.-F. Ssu, Location-free boundary detection in mobile wireless sensor networks with a distributed approach, Computer Networks 70 (2014) 96–112.

[20] A. Kröller, S. P. Fekete, D. Pfisterer, S. Fischer, Deterministic boundary recognition and topology extraction for large sensor networks, in: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, Society for Industrial and Applied Mathematics, 2006, pp. 1000–1009.

[21] W.-C. Chu, K.-F. Ssu, Decentralized boundary detection without location information in wireless sensor networks, in: Wireless Communications and Networking Conference (WCNC), 2012 IEEE, IEEE, 2012, pp. 1720–1724.

[22] R. Jing, L. Kong, L. Kong, Boundary detection method for large-scale coverage holes in wireless sensor network based on minimum critical threshold constraint, Journal of Sensors 2014 (2014) 1–13.

[23] S. Das, I. Banerjee, T. Samanta, Sensor localization and obstacle boundary detection algorithm in wsn, in: Advances in Computing and Communications (ICACC), 2013 Third International Conference on, IEEE, 2013, pp. 412–415.

[24] L.-H. Zhao, W. Liu, H. Lei, R. Zhang, Q. Tan, The detection of boundary nodes and coverage holes in wireless sensor networks, Journal of Mobile Information Systems, Volume 2016 (2016) 2016.

[25] C. Zhang, Y. Zhang, Y. Fang, Detecting coverage boundary nodes in wireless sensor networks, in: 2006 IEEE International Conference on Networking, Sensing and Control, IEEE, 2006, pp. 868–873.

[26] N. Senouci, M. K. El Ouahed, H. Haffaf, Detecting boundary nodes in wsn, in: Proceedings of the International Conference on Wireless Networks (ICWN), The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2014, p. 1.

[27] I. M. Khan, N. Jabeur, S. Zeadally, Hop-based approach for holes and boundary detection in wireless sensor networks, IET Wireless Sensor Systems 2 (4) (2012) 328–337.

[28] A. M. Khedr, W. Osamy, D. P. Agrawal, Perimeter discovery in wireless sensor networks, Journal of Parallel and Distributed Computing 69 (11) (2009) 922–929.

[29] S. Kundu, N. Das, Event boundary detection and gathering in wireless sensor networks, in: Applications and Innovations in Mobile Computing (AIMoC), 2015, IEEE, 2015, pp. 62–67.

[30] F. Lalem, A. Bounceur, M. Saoudi, M. Bezoui, R. Euler, M. Sevaux, Lpcn: Least polar-angle connected node algorithm to find the polygon hull in a connected euclidean graph, (submitted).

[31] N. Santoro, Design and analysis of distributed algorithms, Vol. 56, John Wiley & Sons, 2007.

[32] N. A. Lynch, Distributed algorithms, Morgan Kaufmann, 1996.

[33] K. Mehdi, M. Lounis, A. Bounceur, T. Kechadi, Cupcarbon: A multi-agent and discrete event wireless sensor network design and simulation tool, In IEEE 7th International Conference on Simulation Tools and Techniques (SIMUTools'14), Lisbon, Portugal.

[34] P. Levis, N. Lee, M. Welsh, D. Culler, Tossim: accurate and scalable simulation of entire tinyos applications, in Proc. of the 1st int. conf. on embedded networked sensor systems, ser. SenSys 03. New York, NY, USA: ACM (2003) 126–137.

[35] A. Dunkels, B. Gronvall, T. Voigt, Contiki-a lightweight and flexible operating system for tiny networked sensors, in: Local Computer Networks, 2004. 29th Annual IEEE International Conference on, IEEE, 2004, pp. 455–462.

[36] CupCarbon, Anr project persepteur, cupcarbon simulator (2015).
URL `http://www.cupcarbon.com`

[37] M. Lounis, K. Mehdi, A. Bounceur, A cupcarbon tool for simulating destructive insect movements, 1st IEEE International Conference on Information and Communication Technologies for Disaster Management (ICT-DM'14), Algiers, Algeria.

[38] Wireless Sensor Networks, Product Reference Guide, Crossbow, 2007.

[39] A. Dunkels, J. Eriksson, N. Finne, N. Tsiftes, Powertrace: Network-level power profiling for low-power wireless networks.

[40] J. Oxer, H. Blemings, Practical Arduino: cool projects for open source hardware, Apress, 2009.

[41] A. S. Wander, N. Gura, H. Eberle, V. Gupta, S. C. Shantz, Energy analysis of public-key cryptography for wireless sensor networks, in: Third IEEE International Conference on Pervasive Computing and Communications (PerCom), IEEE, 2005, pp. 324–328.

[42] G. Mao, Localization Algorithms and Strategies for Wireless Sensor Networks: Monitoring and Surveillance Techniques for Target Tracking, IGI Global, 2009.