Efficient Algorithm for Serial Data Fusion in Wireless Sensor Networks[*]

Ahmed Mostefaoui
FEMTO-St Institute DISC dep.
University of Franche Comte
Belfort, 90000, France
ahmed.mostefaoui@univ-fcomte.fr

Mahmoud Melkemi
LMIA laboratory
University of Haute Alsace
Mulhouse 68093, France
mahmoud.melkemi@uha.fr

Azzedine Boukerche
PARADISE Research Lab,
University of Ottawa
Ottawa, Canada
boukerch@site.uottawa.ca

ABSTRACT

Although Wireless Sensor Networks (WSNs) are capable of generating a huge amount of data, the ultimate objective of the underlying applications/end-users is to derive an estimate of a parameter or function of interest through queries sent to nodes containing raw data. The way these queries are handled by the network has a deep impact on its overall performances (e.g., required communications, consumed energy, etc.). Among the numerous centralized and distributed approaches addressing this issue, serial ones have shown an interesting improvements in terms of reducing communication needed for each query and hence reducing required energy. Nevertheless, serial approaches suffer two main drawbacks: (a) they require to construct a path passing through all nodes of the network (which is known to be a NP-Complete problem) and (b) they experience poor scalability. In this paper, we investigate these issues by proposing a novel localized serial approach, called *Peeling Algorithm (PA)*. The proposed approach, because of its localized nature (i.e., no extra-information is needed rather than what it is already available at each node), has shown better support for scalability while reducing significantly needed communications to accomplish a query. The extensive simulation evaluations we made have confirmed the effectiveness of our proposed approach in comparison to other serial approaches. We also provide in this paper formal proofs of its correctness i.e., our distributed approach terminates (free of looping) and visits all connected nodes in the network.

Categories and Subject Descriptors

C.2.4 [**Computer communication networks**]: Distributed applications

Keywords

sensor networks; serial data fusion; distributed and localized algorithm

[*]This work is partially supported by NSERC, Canada Research Chairs program, and MRI/ORF Research Funds.

1. INTRODUCTION

Recent technological advances have lead to the emergence of Wireless Sensor Networks (WSNs) as a promising technology for many applications in several domains [4] (environmental monitoring, health-care diagnostics, animals tracking, etc.). A WSN consists in a collection of nodes, each of which provided with sensing, computing and wireless communication capabilities. Nodes are then deployed in a region of interest, usually in an random fashion, for monitoring purposes. Sensors collect data that is forwarded to the base station, also called *Sink*, in order to answer the application/end-user queries. This process is referred to as *query processing*.

Because of their intrinsic constraints (energy limitation, large deployment, etc.) query processing in WSNs remains a challenging research problem. Many approaches have been proposed in the literature and could be classified into three categories: (a) Centralized approaches [3], also called *warehouse approaches*, in which raw data are first collected by sensors and sent to the sink. The latter, with the help of a database management system, could then answer the queries. In those approaches, the query processing is independent of the the data collection. Although centralized approaches are simple and efficient in terms of queries answering, they however have several drawbacks among them we note overutilization of network resources, mainly communications, and poor scalability. (b) Distributed approaches [8] have then been proposed to specifically overcome these limitations. In such approaches, as the general objective is to derive an estimate of a parameter or function of interest from raw data (e,g., average, source location [15], , ...), sensor nodes do not need to hold global knowledge about the current network topology since each node communicates only with its immediate neighbors. The unknown parameter is then successively (i.e., iteratively) carried out through local computation from the exchanged data between neighbors. The advantages of such approaches are numerous: (1) no central base station is required as every node holds the estimate of the unknown parameter; (2) multi-hop communications are avoided and consequently maintaining rooting data is not needed any more; (3) better behavior is observed in front of communication and nodes unreliability. Nevertheless, distributed approaches observe an important resources consumption, particularly communications as the number of iterations could be important. Furthermore, because of their synchronized iterations, these approaches generate an important number of packets collisions which may decrease the overall network performances (query responsiveness). (c) Serial approaches [13, 12] have then been proposed as an interesting alternatives to reduce communications, including collisions, overhead observed previously in both centralized and distributed approaches. The basic idea behind is to let one node initiating the query and the estimate is then successively (i.e., serially) updated from node to node until all nodes of the network are visited. The last node holds the correct estimate [10]. As stated in [13], serial approaches perform very efficiently in terms of reducing needed communications.

Although theoretical foundations of serial approaches have been provided into [13, 11, 10], some practical issues have not been, to the best of our knowledge, addressed yet: in fact, serial approaches make the assumption that an hamiltonian path within the network exists. However, constructing such a path is known to be a NP-Complete problem [6] which is not easy to meet, especially in sparce and large scale networks. The cost of finding such a path, in a decentralized manner to ensure scalability, could generate a prohibitive communication cost. In [12], the authors propose to use *space filling curves* to derive a path through the network. Besides of its poor scalability, this approach does not efficiently handle irregular network topologies, i.e., networks with holes. Moreover, it does not ensure all nodes visiting.

In this paper, we propose a novel serial data fusion approach that is totally *distributed and efficient*. Two important requirements have guided our work. First scalability which means that the protocol should be totally distributed to scale proportionally with the size of the network. Second, completeness which means that all nodes of the network should be visited i.e. contribute in the parameter estimate. Our approach, thanks to its localized nature i.e., no extra-information than what is usually available (1 hop-information), well fulfills the scalability and robustness requirements as the next hop decision is taken locally and independently by each node. We also provide theoretical proofs of its completeness. The simulation results show clearly the effectiveness of our approach in terms of reducing communication, energy and responsiveness time in comparison to other serial approaches.

The rest of the paper is organized as follows: in Section 2 we present our serial algorithm. Section 3 provides proofs of correctness of the proposed approach. Performance evaluation and results are presented into Section 4 while Section 5 concludes the paper.

2. PEELING ALGORITHM

In this section we present our proposed serial approach, named Peeling Algorithm (PA). We begin first by sketching the overall PA behavior through illustrative examples. Then after, we present the boundary traversal algorithm we used which of course fulfills our major requirement (e.g., of a distributed and localized nature).

We consider a geographic wireless sensor network in which nodes are static and homogeneous, with R as their communication range. We note $\mathcal{N}$ as the set of nodes and $\mathcal{E}$ as the set of links. We consider that each node is aware of its location by means of a positioning system like GPS or as a result of a localization process [2]. Furthermore, each node N_i is aware of its neighboring set $V_i = \{N_j \mid E_{(i,j)} \in \mathcal{E}\}$ and their corresponding locations.

The commonly communication model used in wireless sensor networks is of a broadcasting nature [4] i.e., when a node sends a packets, all neighbors within its transmission range will receive it. Hence, it is worthwhile to notice that the listening process does not incur any additional transmission (no explicit request is sent to the transmitter). This interesting feature of the communication model will be used later on in our approach to update, without any additional communication costs, the visited neighbors table of each node.

We refer in the rest of the paper to the node that issues the request as the *"Query Initiator Node" (QIN)*. In opposition to other approaches, mainly centralized ones, the QIN in our approach could be any node in the network.

We now introduce some definitions used later to highlight our peeling approach.

Definition 1. *(Boundary Node)*
A node N is said to be a boundary node in the direction $\angle N_iNN_j$, noted BN, iff it has at least two angularly adjacent neighbors N_i, N_j such that N_i could not communicate with N_j or the angle $\angle N_iNN_j \geq \pi$

In other terms, messages issued on a boundary must remain on the same boundary. Hence, a node is facing a boundary (hole or network boundary) if any message coming from its right neighbor to its left neighbor "transits" by it. Two cases are possible: either its neighbors could not communicate with each others or the angle is greater or equal to π as illustrated into Figure 1. Each node could face at most six boundaries.

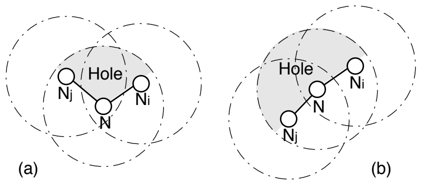

Figure 1: Boundary Node.

With the help of this definition, every node in the network could locally determine if it is located either on the boundary of a hole inside the network or on the boundary of the network. The computation can be performed with only information on 1-hop neighbors.

Definition 2. *(Hole Boundary Nodes)*
For each hole, noted H, we define the set of nodes belonging to it as a cyclic sequence of boundary nodes:
$H = \{N_j, N_{j+1}, ..., N_{j+k}, N_j\}$ such that the closed region bounded by this non self-intersecting polygonal sequence is empty of any node.

Definition 3. *(Network Boundary Nodes)*
We define the set of Network Boundary nodes as a cyclic sequence of boundary nodes: $NBN = \{N_i, N_{i+1}, ..., N_{i+k}, N_i\}$ such that the closed region bounded by this non self-intersecting polygonal sequence contains all nodes of the network.

As explained below, our serial query processing approach starts from a network boundary node. The identification of such node is not straightforward and could not be processed locally as it was the case for boundary nodes. To this end, we present below a distributed and localized algorithm that is able to identify a starting network boundary node.

Definition 4. *(unvisited Sub-Network)*
We define the current unvisited sub-network, noted Ω, as the set of all currently unvisited nodes.

2.1 PA Overview

The key idea behind our algorithm is to let the serial process visiting nodes, starting from one node, located on the external boundary of the network, by means of a boundary traversal process. Hence, from this node, the process starts visiting nodes based on their localization on the boundary of the unvisited region Ω. Initially, for any query we have $\Omega = \mathcal{N}$. Then, at every step of the serial process, the currently visited node is removed from Ω (marking itself as visited) and in turn it selects the next unvisited node located on the boundary of Ω. This process is repeated until all nodes are visited: the last node that has all its neighbors visited detects query termination and sends the result to the QIN by mean of any geographical routing approach [5]. The term peeling comes from the fact that all currently visited nodes belong to the boundary of Ω. Figure 2(a) illustrates the peeling process.

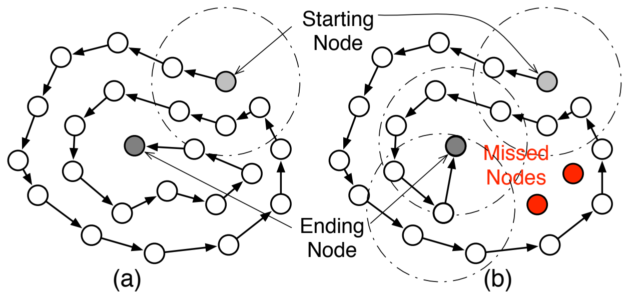

Figure 2: Peeling Serial Algorithm.

This simple process whereas it behaves very efficiently in dense and hole free topologies (it nearly derives the optimal path), it however does not ensure completeness (i.e., be able to visit all nodes) for all network topologies. A simple example is when a node is connecting two parts of the network. Once the latter is visited, the unvisited region connectivity is not ensured any more leading thus to *missed unvisited nodes* as illustrated into Figure 2(b).

To prevent this situation, certain nodes have to be maintained *"alive"* to ensure network connectivity even though they have been visited. For this reason, we have introduced the notion of **Bridge Node** as follows:

Definition 5. *(BRidge Node)*
A node is said Bridge Node, noted BRN, when it verifies the two following conditions: (a) at least two of its unvisited or BRN neighbors could not communicate and (b) none of the other unvisited or BRN neighbors are able to connect these two neighbors.

In other terms, the current node is the only node able to ensures connectivity between these two neighbors. An illustration of BRN is given into Figure 3. The left node is considered as BRN whereas the right one is not.

It has to be noted that the current node could locally decide whenever it is a BRN or not based only on neighboring information. No additional information than what is already available is required. This ensures that our approach

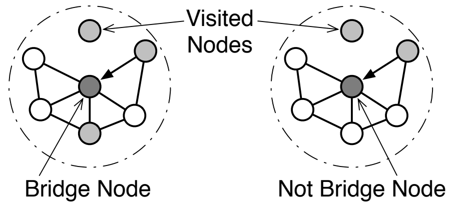

Figure 3: Bridge Node definition.

remains localized and does not generate any additional communication overhead. We also note that a node could change its "status" from BRN to visited node depending on its localization on one hand and on its currently visited neighbors on the other hand. The broadcasting/listening nature of the communication model used in wireless sensor networks ensures that neighboring status updates do not incur any additional communication overhead.

Property 1.
Bridge nodes ensure unvisited sub-network (e.g. Ω) connectivity

PROOF. We prove by contradiction. Initially Ω is connected. We assume that the current node, N_i is not a bridge node whereas at the same time it breaks Ω connectivity. From Definition 5, if N_i is not a bridge node, this means that for every pair of its neighbors, there is a link connecting them without passing by N_i. Hence, Ω is then connected which contradicts the fact that N_i breaks Ω connectivity. $\square$

Although bridge nodes resolve the problem of unvisited sub-network connectivity, they however rise another problem when the network contains hole. In fact, according to Definition 5, all hole boundary nodes, belonging to the same hole H, will be marked as bridge nodes. The consequence, in this case, is that some nodes, with particular localization around the hole, could not be visited any more i.e., they prevent the peeling process to access those nodes leading hence to query completeness violation as illustrated into Figure 4. Moreover, as all hole nodes are "labeled" as bridge's nodes, the termination of the peeling process, as stated earlier, is not guaranteed too. It leads usually to looping situation as showed into the example of Figure 4.

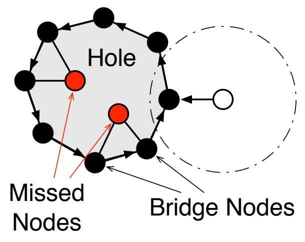

Figure 4: Example of peeling looping.

To resolve these two important issues, while continuously maintaining the connectivity of the unvisited nodes (e.g., Ω), we have associated a particular rule to special hole nodes defined as follow:

Definition 6. *(Hole Gate Node (HGN))*
For each hole H, we define its Hole Gate Node (HGN) ($HGN \in H$) as the first node that holds the peeling process.

The goal behind defining Hole Gate Nodes is to let the peeling process going inside the hole. By doing so, we avoid situations in which nodes, located *"around"* holes, could not be visited as explained in the example. The questions that follow are how a node could be labeled as HGN? and what is/are the associated rule(s)?

To answer the first question, two steps are necessary: (a) a node must first determine if it is located on the boundary of a hole or not? As explained above in Definition 1, this could be done locally by each node thanks to the neighboring available information ; (b) Secondly, when a node is aware of its localization on the boundary of a hole, it could determine if it is the HGN of this hole or not when it receives the peeling message for the first time: if the peeling comes from a node that does not belong to the hole then it deduces that it is the first visited node of the hole and consequently it is the HGN of this hole. Otherwise, the hole has already its HGN. According to that, we note that each hole has only one corresponding HGN. For these two steps, we mention that no extra-information is needed and could then be processed locally.

The second question has to be handled carefully in order to prevent looping inside the hole. We observe that each hole node has two hole neighbors. We note them the *Left Neighbor (LN)* and the *Right Neighbor (RN)* as presented into Figure 5. When a node detects itself as a HGN, it performs the following steps:

1. It contributes into the query process and marks it self as a HGN.

2. According to the peeling direction (i.e., clockwise or counterclockwise), it chooses either LN or RN as the next query holder. For sake of simplicity, we assume that it chooses LN as the next hop.

3. Node RN is then notified to update its neighbors table with "visited" label for node HGN even when the latter is considered as BRN. In other terms, we break the link between the two nodes *HGN* and *RN* (see Figure 5).

4. HGN sends the query to LN.

When LN holds the query, the process could continue normally by using the boundary traversal algorithm. Step 3 prevents the peeling to loop inside the hole since RN will at its turn consider HGN as visited node and could not select it as next hop.

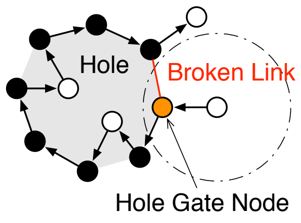

Figure 5: Hole Gate Node Rule.

These rules, applied by the HGN of each hole, were practically sufficient to ensure query completeness. In fact, among the numerous network configurations, generated randomly as explained into Section 4, none has experienced network completeness violation i.e., all alive nodes in the network have been visited by our approach. Nevertheless, from theoretical point of view, we have constructed by hand some topologies in which our approach fails to visit all nodes. An example of such a topology is presented into Figure 6(a). When node N_1 receives the query, it considers itself as the HGN of $hole_1$ and applies the corresponding rules according to previous steps. Hence, the link $E_{(1,17)}$ is broken and the query is oriented toward node N_2. Similarly, when node N_4 receives the query, it considers itself as the HGN of the hole located on its left side and decides to break the link $E_{(4,5)}$. It sends then the query to node N_{10}. By doing so, all nodes located on $hole_2$ are lost and could never be visited later on. The origin of this misbehavior comes from the fact that node N_4 could not locally know that it is located on the same hole. In other terms, node N_4 locally "views" two holes whereas it is the same hole, leading hence to a *False HGN* detection. This information is not available for N_4 at the moment of receiving the query.

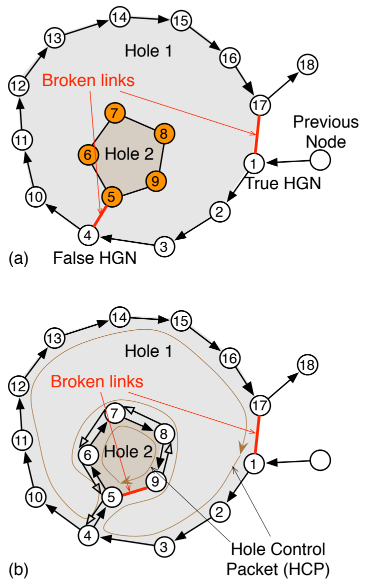

Figure 6: Peeling Misbehavior.

To alleviate this problem, we have added an additional rule to the HGN, when it detects itself as a HGN. Before selecting its next hop according to step 2, it first sends a control packet, named Hole Control Packet (HCP), toward the hole as showed in Figure 6(b). The objective of this packet is twice: first it informs all hole nodes that the hole has "got" its HGN and consequently they will not consider this hole when they receive the query. Secondly, it allows nodes to detect if their "local holes" do not belong to the same hole. For instance, node N_4 will know that its left and right holes belong to the same hole, which has already got its HGN. In this case, when it receives the query, it will not consider itself as HGN of this hole and hence lets the peeling process continuing after contributing in it. However, node N_5 could be able to detect itself as the HGN of $hole_2$ and apply consequently the corresponding rules i.e, sending a HCP and breaking the link with the right neighbor as shown into Figure 6(b).

The HGN rules could be seen somehow as a way of *linearizing* holes preventing hence looping situations and allowing inside holes exploration. On the other hand, they always preserve the unvisited region connectivity i.e, the removed links could not lead to Ω partition.

Property 2. *HGN rules do not partition the unvisited region Ω i.e., Ω connectivity remains.*

PROOF. We assume that the broken link, says $E_{(HGN,RN)}$, leads to Ω partition. This means that no path could connect nodes HGN and RN. However, we note that both sides of the broken link $E_{(HGN,RN)}$ could not by construction belong to the same hole, thanks to HCP rule lunched by the HGN of the hole. Also, by construction, each hole holds one and only one HGN. Consequently, it exists a path connecting HGN to every node in the hole without passing through link $E_{(HGN,RN)}$, in particular node RN. This contradict the initial assumption. $\square$

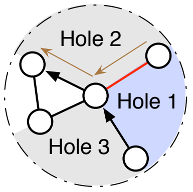

Figure 7: Node's local holes selection.

The overall serial query algorithm, executed by each node receiving the query message, is depicted into Algorihm 1. We assume that the boundary traversal algorithm is using counterclockwise direction.

Algorithm 1 Peeling Algorithm.

```
Require: Receive message M from the previous hop
 1: if (Not visited for this query) then
 2:    Contribute in the query.
 3:    Mark itself as visited node for this query.
 4:    if (current node is HGN) then
 5:       // In case where the current node is located on more
 6:       // than one unexplored hole (see Figure 7).
 7:       From the incoming message, select the first coun-
          terclockwise hole among local unexplored holes.
 8:       Send HCP toward the selected hole
 9:       Upon receiving the HCP, mark itself as HGN
10:       Select the left neighbor as the next hop.
11:    else
12:       Select the next hop among unvisited and BRN
          neighbors.
13:    end if
14: else
15:    Select the next hop among unvisited and BRN neigh-
       bors
16: end if
17: Compute its status (i.e., is the current node a BRN or
    not?)
18: send the message to the next hop $\mathcal{N}_{next}$
```

2.2 Boundary Traversal Algorithm

As seen before, the use of a boundary traversal algorithm is crucial in our peeling algorithm to ensure its completeness (i.e., visiting all nodes). Of course, the distributed and localized nature of the boundary algorithm must be maintained in order to reach good communication performances. In previous work [9], we have proposed an efficient boundary traversal algorithm, called Curved Stick (CS). In addition to deriving shorter paths, this approach has the advantage of being fully localized while at the same time formally ensures boundary traversal. It works as follows:

The process always starts from a boundary node, called Boundary Traversal Initiator (BTI). The BTI initiates a *"Curved Stick"* (CS) which has the form of an arc with radius of R (node's communication range). Initially the CS, hinged at the current node, is oriented in the direction of the hole/network boundary as shown into Figure 8. From this position, the BTI sweeps the CS counterclockwise[1] until a neighbors is hit. The latter is then selected as the next hop and will repeat the same process starting from its corresponding *"starting point"*, defined as fellow:

Definition 7. *(Starting Point)*
We define the Starting Point of node N_{i+1}, noted SP_{i+1} as the intersection point between the two circles of range R, centered at N_i and N_{i+1} and following the direction $\angle N_{i-1}N_iN_{i+1}$ where N_{i-1} is the previous hop of node N_i and N_{i+1} is the next hop of N_i.

And so on until the message gets back to the BTI which means that a cycle has been achieved (hole boundary detection for instance).

Theorem 1.
Whenever started from a boundary node, curved stick process ensures boundary traversal.

PROOF. Because of space limitation, we provide here just the idea behind our proof. Details can be found in [9]. We assume that CS does not ensure boundary traversal i.e., there is a node located in the grey area if Figure 8 which is not hit by CS and show that this leads to a contradiction. $\square$

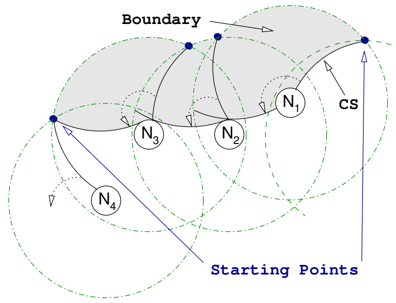

Figure 8: Curved Stick boundary traversal algorithm.

In our peeling implementation, we have made use of the curved stick approach.

2.3 Starting Node Detection

As mentioned previously, our peeling approach must start from a node located on the external boundary of the network. Furthermore, each network boundary node must update the list of its local holes in order to avoid considering the external network boundary as a hole.

[1]The sweeping direction does not impact the process i.e., the clockwise direction could also be used but the direction must remain the same during the whole process.

Several approaches could be developed to inform nodes about their localization on the external boundary of the network. One could though about a centralized approach in which every node sends its localization information to the sink and the latter, through local processing, detects those located on the external boundary of the network. After, this information is sent back to NBNs. Besides its important energy consumption, this approach presents another very limitative weakness: it is not scalable any more. Scalability remains however an important requirement to protocols in large-scale sensor networks.

To overcome these limitations, we propose a distributed and localized approach, inspired from our previous work [9], to ensure external boundary nodes detection. This approach does not require any additional knowledge than node's localization and neighbors information.

The key idea behind our approach is to use geographic greedy routing to try to reach a virtual node located outside the network area, as illustrated into Figure 9. Hence, a message is initiated from a source node, usually the sink, to reach this virtual node. The source node, knowing the location of the destination node, sends the packet to its 1-hop neighbor which is the closest node to the destination among all neighbors. Knowing the destination location from the received message, every node in the path repeats this process until the message fails into local minimum situation (i.e., the message could not be delivered to the next hop as the current node is the closest one to the destination). From this node, also called local minimum node, recovering process is lunched using Curved Stick boundary traversal approach as explained earlier. This process is repeated until the current selected node is close to the virtual node than the local minimum node. Then the greedy forwarding is used again, and so on.

Following this approach and knowing that the virtual node is unreachable by construction, the message will make a cycle and gets back to the last local minimum node, noted N_l.

Property 3.
The last local minimum node, noted N_l, belongs to the set NBN.

PROOF. Initially, all nodes of the networks are connected (i.e., network connectivity assumption). Also, from the fact that CS ensures boundary traversal (c.f., Theorem 1) on one hand and that greedy rule selects the closest node to the virtual node on the other hand, it results that the last local minimum node has the lowest distance to the virtual node among all nodes of the network. Hence it is located on the network boundary. □

Once a node has been identified as belonging to the set NBN, we note Node N_l (N_{14} in the example of Figure 9), from this node and using the CS boundary traversal algorithm, all visited nodes belong to the set of network boundary nodes. Hence the latter could easily update their local holes list.

3. PROOF OF CORRECTNESS

As for any decentralized algorithm, we provide in this section correctness proofs of our peeling algorithm. More precisely, we prove that it terminates (i.e., does not fail into looping) and visits all nodes of the networks (i.e., query

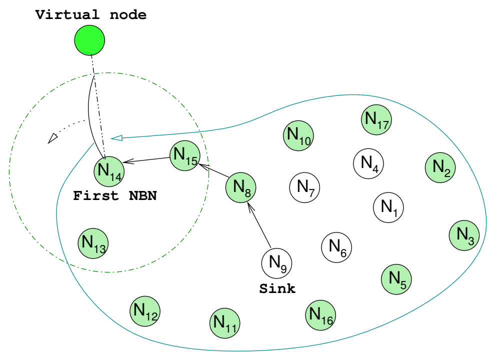

Figure 9: Network Boundary Nodes detection process.

completeness). We point out that PA is based on two main ideas. The first one is to iteratively shrink the resulting network by visiting the external boundary without loss of connectivity (Property 1), while the second one (i.e., HGN rules) allows the transformation of the input network into a connected one without holes (Property 2).

Formally, PA constructs iteratively the set Ω_i, following the recurrent relationship defined as follows:

$$\begin{cases}\Omega_0 = \mathcal{N} \\ \Omega_{i+1} = \Omega_i - \{P_i\}\end{cases}$$

Where P_i is the first visited node which is not a bridge node during the scan of Ω_i. For instance, in Figure 6(b), if we note the sub-network of nodes 1 to 18 as Ω_k, then $\Omega_{k+1} = \Omega_k - \{N_1\}$, $\Omega_{k+2} = \Omega_{k+1} - \{N_2\}$, $\Omega_{k+3} = \Omega_{k+2} - \{N_3\}$, $\Omega_{k+4} = \Omega_{k+3} - \{N_9\}$, $\Omega_{k+5} = \Omega_{k+4} - \{N_8\}$, ...

Lemma 1.
At each iteration i, PA visits a no bridge node P_i (i.e., P_i exists).

PROOF. Suppose that at step i, P_i does not exist. Since CS ensures boundary traversal (cf. Theorem 1), this means that all visited nodes are bridge nodes . Consequently, the visited nodes form a hole. This contradicts the no existence of holes guaranteed by HGN rules. Therefore, we deduce that at each step i, P_i exists. □

With the help of the previous lemma, we have:

Theorem 2.
PA generates a finite sequence $\mathcal{P} = \bigcup_{i=0}^{k} P_i$ (i.e., PA terminates).

PROOF. To prove that PA performs a finite number of iterations and terminates, we should show that the number of nodes in Ω_i decreases as i grows. Initially we have $\Omega_0 = \mathcal{N}$, which is a finite set of nodes. From Lemma 1, P_i exists at each iteration i. Therefore, it exists k such that $\Omega_{k+1} = \emptyset$. Hence, we prove that $\mathcal{P}$ is finite and PA terminates. □

We define the set of Ω_i boundary nodes, noted $B(\Omega_i)$, as follows:

Definition 8. *(Ω_i Boundary Nodes)*
The boundary of Ω_i is a cyclic sequence of unvisited or bridge boundary nodes such that the closed region bounded by this non self-intersecting polygonal sequence contains all the nodes of Ω_i.

Lemma 2.
At each iteration i, node $P_i \in B(\Omega_i)$ i.e., P_i is on the boundary of the current sub-network Ω_i.

PROOF. We prove by induction. Initially, PA starts at a node located on the boundary of the network Ω_0 and scans the nodes of the boundary of Ω_0 (as explained in Boundary Traversal Algorithm, see Subsection 2.2) until it hits P_0. Thus P_0 is on the boundary of Ω_0, hence the property is true for $i = 0$. Suppose that the property is true for i and we should show that it remains true for $i+1$. As the property is true for i, the current node is P_i and it is located on the boundary of Ω_i. Boundary Traversal Algorithm guarantees that next hop P of P_i is on the boundary of Ω_i. This next hop is also on the boundary of Ω_{i+1}. If P is not a bridge then $P_{i+1} = P$ otherwise starting at P, PA scans the boundary nodes of Ω_{i+1} until it hits a no bridge node P_{i+1}. Hence the property is also true for $i + 1$. □

Theorem 3.
The sequence $\mathcal{P}$ generated by PA contains all nodes of the network (i.e $\mathcal{P} = \mathcal{N}$).

PROOF. We assume that $\mathcal{P}$ is different from $\mathcal{N}$ and we seek to derive a contradiction. We note N_u as an unvisited node missed by PA algorithm, namely $N_u \neq P_i$ for all $i = 0, ..., n$. Specifically, consider the integer $0 < j < n$ such that N_u belongs to Ω_j and does not belong to Ω_{j+1} (i.e N_u is missed at iteration j). As at step j, PA hits only nodes on the boundary of Ω_j and does not miss any boundary node (cf. Lemma 2 and Theorem 1) then N_u cannot be located on the boundary of Ω_j. Thus N_u belongs to Ω_{j+1}, which contradicts the fact $N_u \notin \Omega_{j+1}$, and consequently contradicts the initial assumption. □

In fine, PA terminates and ensures query completeness.

4. PERFORMANCE EVALUATION

For comparison purposes, we have implemented, in addition to our peeling algorithm, the well known serial approach called Depth First Search (DFS) [14]. DFS constructs a rooted tree by expending, at each time, this tree towards unvisited nodes. Hence, each node, except the root, has a "father" node and each node has one or several children except the leafs. To explore a network with N connected nodes, this algorithm requires $2 * (n - 1)$ communications whereas an optimal Hamiltonian path, that does not exist in every network, requires $(n - 1)$ hops. DFS ensures completeness since it allows visiting all nodes. We have used OMNET++ simulator [1], with the Castalia Package to run our simulations.

In our comparison, we have considered three main metrics: (a) *communications efficiency* which records the number of communications needed to end a query; (b) *energy consumption* [7] of all nodes to finish a query and (c) *Time-To-End* that reflects the time taken by each query to deliver the result.

The overall simulation parameters are depicted into Table 1. For each query configuration (i.e., network deployment and query initiator), we have run the two studied approaches and recorded the obtained results. The presented results below are the mean values of 30 runs for each network configuration.

Figure 10 plots the results of required communications (e.g., number of hops). We have also plotted the theoretical optimal path which does not necessary exist in any network configuration.

Parameter	**value(s)**
Network Area (m^2)	250 x 250
Transmission range (m)	50
Nodes distribution	Uniform
Number of nodes	25, . . . , 300
Query initiator location	Random
Packet size	50 Bytes

Table 1: Simulations parameters

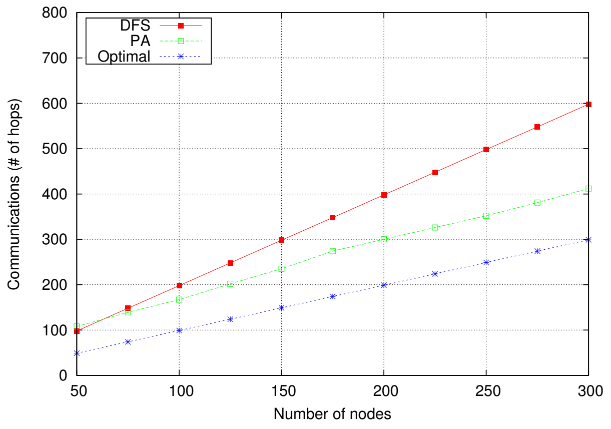

Figure 10: Communication requirement for one query.

As expected, the communication requirement increases with the increase of the number of nodes in the network for the two approaches. This increase is linear for DFS approach. In sparce networks, with low densities, DFS slightly outperforms PA. This is due to the fact that in such networks, the probability of facing linear topologies is high. We recall that linear topologies are the most disadvantageous configurations for our peeling algorithm. In fact, a deep analysis of PA behavior in such topologies reveals that the initialization phase (i.e., starting node detection) is the most communications consuming part. As an example, we have constructed a linear topology composed of 10 nodes, and chosen the worst case where the query initiator was located on the opposite side of the virtual node, as explained into Subsection 2.3. In this configuration, the initialization phase required $[9 + (2 * 9)] + (2 * 9)$ communications, representing respectively the starting node detection and the notification of external network boundary nodes, whereas the query processing required 9 communications. In comparison, DFS required $(2 * 9)$ communications. Nevertheless, it is worthwhile to notice that our approach does not require explicitly initialization phase for each query execution. In fact, once a node has detected itself as the starting node and noticed all network boundary nodes about their localization during the first query, it could avoid this operation for all the remaining queries. For instance, in our linear example, the second query, even in the worst case (i.e., initiated from a node located far away form the starting node), will require 9 communications to reach the starting node and 9 communications to execute the query. An so on. Finally, for an evaluation of k queries, in the worst cases, peeling approach requires $[9+(2*9)+(2*9)]+k(9+9)$ communications whereas DFS requires $k(2*9)$ communications. When k becomes large, the difference between peeling and DFS in terms of communication performance will not be noticeable. This analysis is valide only in the worst network configuration for our peeling approach (i.e., linear topologies). However, in randomly deployed networks, beyond a certain density, PA largely outperforms DFS as shown in the figure, even for single query as it is the case in our results. For instance, the improvement when the network is composed of 300 nodes is more than 31%.

Figures 11 and 12 present the obtained results for the total energy consumption and the query duration (i.e., time to end) respectively. Similarly, PA algorithm presents better behavior than DFS from a certain network density (e.g. 75 nodes). For energy consumption, the improvement ranges from 5% in sparce networks to 43% in dense networks whereas the query responsiveness reduction ranges from 7% to 30%. In summary, the overall results show clearly the effectiveness of our proposed peeling algorithm.

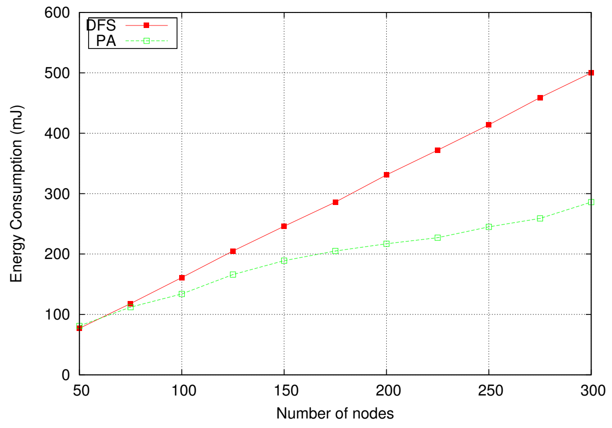

Figure 11: Total energy consumption for one query.

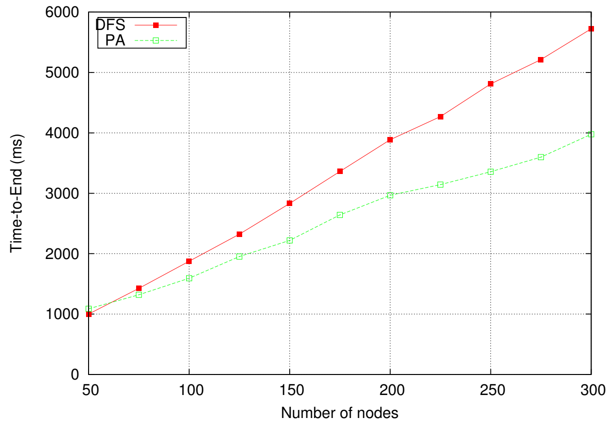

Figure 12: One query duration.

5. CONCLUSION AND FUTURE WORKS

Serial approaches for data fusion in WSNs have shown their effectiveness in terms of reducing communication overheads as well as energy consumption. Nevertheless, previous research works have not explicitly addressed the key issue of constructing the serial path. Moreover, the query completeness (in the sense of visiting all nodes) was not ensured any more. In this paper, we have tackled this important issue by proposing a *localized approach*, named peeling algorithm. The main advantage of our approach is that does not require any additional knowledge than what is traditionally available in WSNs. It is one hop approach and does not require any knowledge about the topology of the network. We have formally proven that our approach ensures query completeness and is free of looping. The obtained simulation results have highlighted its effectiveness in terms of reducing communications requirement, energy consumption and time responsiveness to accomplish one query.

In future work, we plan to consider additional network nodes distributions than the uniform one. In fact, some preliminaries results on other distributions, while they confirmed again the effectiveness of our peeling algorithm, have highlighted however other possible enhancements because we have noticed that some links and bridge nodes are solicited more frequently than the average. We believe that introducing additional rules for these specific nodes will enhance our peeling algorithm.

Another important issue, that we plan to address in future work, is robustness. in fact, serial approaches are known to be vulnerable to links and nodes failures. How to make our peeling algorithm robust will constitute our future challenge.

6. REFERENCES

[1] http://www.omnetpp.org/.

[2] J. Bahi, A. Makhoul, and A. Mostefaoui. Localization and coverage for high density sensor networks. *Computer Communications*, 31(4):770–781, 2008.

[3] P. Bonnet, J. E. Gehrke, and P. Seshadri. Towards sensor database systems. In *Second International Conference on Mobile Data Management*, pages 3–14, January 2001.

[4] A. Boukerche. *Algorithms and Protocols for Wireless Sensor Networks*. Wiley Series on Parallel and Distributed Computing, ISBN-10: 0471798134, 2008.

[5] A. Boukerche, B. Turgut, N. Aydin, M. Ahmad, L. Bölöni, and D. Turgu. Routing protocols in ad hoc networks: a survey. *Computer Networks (Elsevier)*, 55(13):3032–3080, 2011.

[6] M. R. Garey and J. D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman, p. 199, 1983.

[7] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1(4):660–670, 2002.

[8] B. Khaleghi, A. Khamis, and F. O. Karray. Multisensor data fusion: A review of the state-of-the-art. *Information Fusion*, 14(1):28–44, 2011.

[9] A. Mostefaoui, M. Melkemi, and A. Boukerche. Routing through holes in wireless sensor networks. In *15th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems*, pages 395–402, 2012.

[10] A. Nedic and D. P. Bertsekas. Incremental subgradient methods for nondifferentiable optimization. *SIAM Journal on Optimization*, 12(1):109–138, 2001.

[11] R. Nowak. Distributed em algorithms for density estimation and clustering in sensor networks. *IEEE Trans. on Sig. Proc.*, 51(8):3, August 2003.

[12] S. Patil, S. Das, and A. Nasipuri. Serial data fusion using space-filling curves in wireless sensor networks. In *IEEE Conference on Sensor and Ad Hoc Communications and Networks, IEEE SECON*, pages 182–190, 2004.

[13] M. Rabbat and R. Nowak. Quantized incremental algorithms for distributed optimization. *IEEE Journal on Selected Areas in Communications*, 23(4):798–808, 2005.

[14] E. Shimon. *Graph Algorithms (2nd ed.)*. Cambridge University Press, pp 46-48, 2011.

[15] F. Zhao and L. Guibas. *Information Processing in Sensor Networks*. Springer LNCS, 2003.