Distributed Collision-free Data Aggregation Approach for Wireless Sensor Networks

Mohammed Amine Merzoug*, Ahmed Mostefaoui† and Samir Chouali†
*Department of Computer Science, Faculty of Exact Sciences
University of Bejaia, 06000 Bejaia, Algeria
amine.merzoug@gmail.com
†FEMTO-ST Institute, DISC Dep.
University of Bourgundy-Franche-Comte, France
ahmed.mostefaoui@univ-fcomte.fr, samir.chouali@univ-fcomte.fr

***Abstract*—This paper presents a distributed serial data aggregation approach, called Spreading Aggregation (SA), in which one packet hops sequentially over nodes and aggregates their data. The next hop of the aggregation packet is determined locally by each traversed node using only its one-hop neighborhood information, so no network topology information needs to be known by nodes, nor collisions are generated as only one node is communicating at any given time. This localized and distributed characteristic makes the proposed approach highly scalable and very efficient in terms of communication-reduction, energy conservation, and aggregation time, as confirmed by the numerous simulation results we obtained. These results confirm also the superiority of the proposed approach over the state-of-the-art serial approaches, particularly in very large scale network deployments.**

***Index Terms*—Collision-free data aggregation, Serial data aggregation, Wireless sensor networks.**

I. Introduction

Typically, a Wireless Sensor Network (WSN) is composed of hundreds or even thousands of smart autonomous wireless sensor nodes [1]. After being deployed in the area of interest, nodes aggregate sensed data and send it to the sink. The way data is aggregated has a great impact on the overall network performances, in particular energy consumption and response time. In general, we distinguish in the literature two main categories [2]: (a) tree-based approaches [3] [4] that rely on some pre-constructed spanning trees for both query dissemination and data collection/aggregation. In addition to the high cost in terms of energy consumption for maintaining such structures, these approaches exhibit a very poor scalability which renders them unsuitable for large scale network deployments. (b) Serial approaches [2][5] where data is aggregated sequentially, from node to node, until visiting the entire network. The main drawback of serial aggregation is that it requires the construction of a path that starts from the sink and crosses every node in the network. However, in opposition to tree-based approaches, they combine path-construction and data-aggregation; while the path is constructed, data is aggregated at the same time. Furthermore, serial approaches exhibit another interesting feature that is *collision-free* since at any given instant of time, in the entire network, only one node is allowed to communicate. This unique characteristic allows serial approaches to outperform tree-based approaches in terms of data aggregation time (i.e., latency), as confirmed by previous research works [2].

In this paper, we propose a novel serial approach, specifically tailored for very large scale network deployments. The proposed approach ensures the following requirements:

- *Network exploration and aggregation completeness*: provided that the network is connected, the proposed approach has to be able to cope with any possible topology and aggregate data of all nodes.
- *Structure-free design and failure-robustness*: in order to be robust against links and nodes failures, the proposed approach has to construct the path gradually. That is, instead of attributing a next hop to each node, the current traversed node has to be able to autonomously select the next hop. We mention that a pre-constructed path renders the aggregation process very vulnerable to links/nodes failures. If at some point, the predetermined next hop is unavailable, the current node will have no choice but to stop the aggregation process.
- *Localized design and scalability*: in order to be scalable, the proposed approach has to be localized. That is, when selecting the next hop, the current traversed node has to use only its local one-hop neighbor information, no further information is required.

In order to show the improvement of the proposed approach, we have conducted several simulation series. The obtained results confim the effectiveness of our proposal in particular in very large scale network deployments.

The rest of the paper is organized as follows: Section II reviews the preliminaries required for this paper. Section III presents the proposed distributed approach and details its behavior through examples. Section IV presents and comments the simulation results. Finally, Section V concludes the paper and discusses possible future research directions.

II. Background and preliminaries

A. Network model

We denote the finite set of nodes by $\mathcal{N} = \{N_1, \ldots, N_n\}$, and the finite set of links by $\mathcal{L} = \{L_{(i,j)} \mid N_i,\ N_j \in\ \mathcal{N} \wedge$

$i \neq j\}$. We consider non-oriented bi-directional links, i.e., link $L_{(i,j)}$ is the same as $L_{(j,i)}$. A link $L_{(i,j)}$ exists, if nodes N_i and N_j communicate directly with each other. We assume that the network is connected and all nodes are aware of their locations through any localization technique [6]. Further, we assume that the communication range of all nodes is set to the same value R [7][8] and that each node is aware of its one-hop neighbors and their locations. The neighbors set of node N_i is denoted by $V_i = \{N_j \mid L_{(i,j)}\ or\ L_{(j,i)} \in \mathcal{L}\}$.

B. Boundary Traversal

The key idea behind our serial data aggregation approach is boundary traversal. So, in this section, we explain the priniciple of operation of boundary traversal algorithms.

1) Notion of Boundaries: in a wireless network, a boundary can be either the boundary of a hole inside the network or the external boundary of the network. Fig. 1 gives an illustrative example of boundaries. Note that the boundary of the network or that of a hole, are both composed of a set of nodes called *boundary nodes*. For example, the boundary of hole 1 is composed of the boundary nodes N_1, N_2, N_3, and N_4.

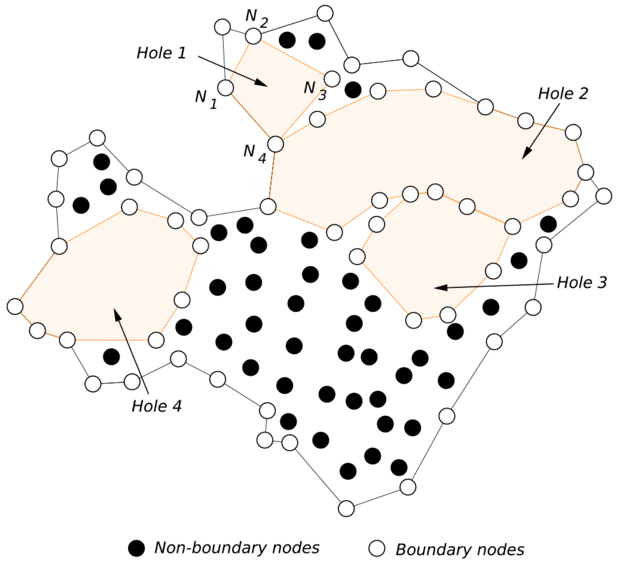

Fig. 1. Example of boundaries in a Wireless Sensor Network.

A hole in a network can be defined as follows:

Definition 1. *Hole*
A hole is a closed region empty of nodes, and delimited by the non-intersecting links of at least four nodes.

For instance, in Fig. 1, hole 1 is delimited by the following non-intersecting links: $L_{(1,2)}$, $L_{(2,3)}$, $L_{(3,4)}$, and $L_{(4,1)}$.

2) Boundary nodes identification: in order to identify boundary nodes, several definitions have been proposed in the literature [2] [9] [10]. The definition adopted in this paper is based on a specific geometric shape called *Rolling-Ball*. Actually, a *rolling-ball* is a virtual circle $B_i(c_i,\ R/2)$ attached to a node N_i and containing no other node. Fig. 2 shows a rolling-ball attached to node N_1 (circle with center $c_1 \in \mathbb{R}^2$, and radius $R/2$, where R is the communication range of nodes). The details and definition of the rolling-ball can be found in [9]. Using the rolling-ball, the definition of a boundary node can be given as follows:

Definition 2. *Boundary Node*
A node N_i is said to be Boundary Node, iff when rollling locally the ball starting from its farthest neighbor, node N_i can be hit (touched) by the ball.

Considering Definition 2, a node can determine whether it is boundary or not, based only on its one-hop neighborhood information.

3) Boundary Traversal: the objective of boundary traversal algorithms is to sequentially browse nodes belonging to a given boundary [9] [10]. First, boundary traversal algorithms are localized and do not require any information other than the one-hop neighbors of each node, which serves our objective of proposing a localized data aggregataion approach. Second, boundary traversal algorithms are proven to ensure boundary traversal, which serves our objective of ensuring aggregation completeness.

The boundary traversal algorithm used in this paper is the rolling-ball boundary traversal [9]. In this algorithm, as its name suggests, the rolling-ball is used as a tool to traverse boundaries. As Fig. 2 shows, in order to choose the next hop, the initial node N_1 spins the rolling-ball counterclockwise. The first touched (hit) neighbor (node N_2) is considered as the next hop. In its turn, the second node N_2 also spins the received rolling-ball to determine the next hop. The process is repeated at each visited node until traversing the whole boundary. We mention that in order to ensure boundary traversal, the rolling-ball must be all the time empty of nodes. We mention also that the counterclockwise direction has been adopted in this paper just for the sake of clarity. The clockwise direction can be also used.

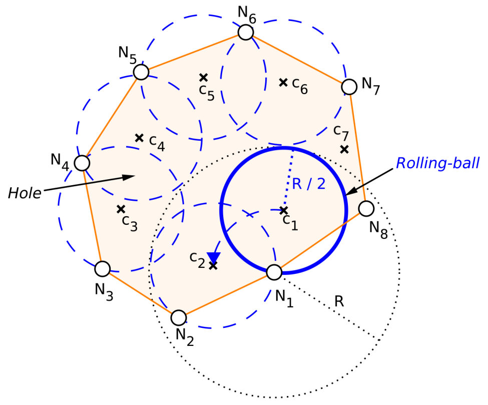

Fig. 2. Rolling-ball boundary traversal.

III. Distributed Approach

In this section, we present the proposed serial data aggregation approach. We refer to the node that launches the data aggregation process as the APL (Aggregation Process Launcher). We define the set of unvisited nodes and the set of visited nodes as follows:

Definition 3. *Unvisited Set*
The unvisited set (Ω) is the set of all currently unvisited nodes.

Definition 4. *Visited Set*
The visited set (Γ) *is the set of all the current visited nodes.*

Initially, all nodes in the network are marked as *unvisited* ($\Omega = \mathcal{N}$ and $\Gamma = \emptyset$).

A. Aggregation process initialization

In the proposed approach, the APL can be any node in the network. So, we distinct two cases: (1) whether the APL is a boundary node or (2) the APL is a non-boundary node.

1) Boundary APL: in this case, the APL can be located on a hole's boundary or on the network's boundary. In both cases, to initiate data aggregation, the APL marks itself as *visited* and launches a rolling-ball. That is, the APL is removed from Ω and added to Γ, and the aggregation packet is sent to the first neighbor hit by the rolling-ball (Fig. 3(a)). Upon receiving the packet, the second node aggregates its data with the APL's data, marks itself as *visited*, spins the rolling-ball and forwards the packet to the first hit unvisited neighbor (Fig. 3(a)).

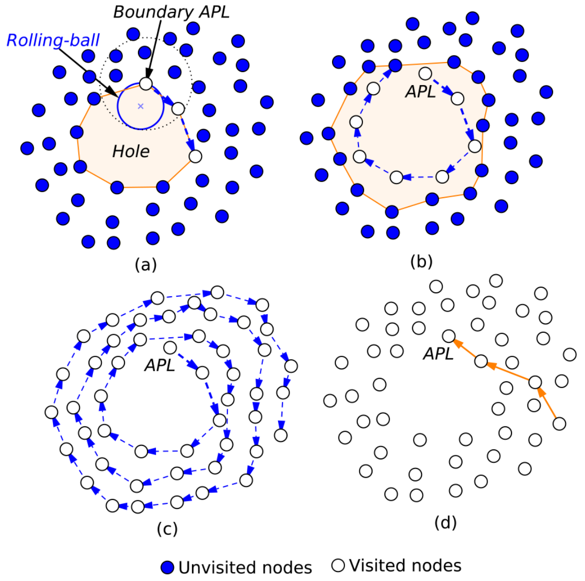

Fig. 3. Spreading aggregation (case of a boundary APL).

Once all nodes of the initial boundary have been visited, this boundary is removed from Ω and added to Γ (Fig. 3(b)). In other words, a new boundary will appear in Ω and the same process will be applied to it (i.e., this boundary will be also visited using the rolling-ball). This way, the network will be traversed boundary by boundary. The aggregation termination is detected at a node when this latter has no remaining *unvisited* neighbors (Fig. 3(c)). Once the aggregation process is completed, the aggregated data can be found at the last node in the path. Given the fact that the last node can be different than the APL, the aggregated data needs to be sent to this latter. To do so, any geographic routing can serve the purpose [1] (Fig. 3 (d)).

To be able to correctly select the next hop, each node must be aware of the status of its neighbors (*visited* or *unvisited*). To this end, attributing a field for each neighbor (in the neighbors table of a node) will indicate whether a neighbor has been visited or not. This way, the current traversed node will recognize its visited neighbors and will not select them as next hop. The process of updating neighbors' status does not require any additional communication other than the aggregation packet itself. The credit for this is due to the broadcast communications of wireless sensor networks [1] (when a node sends a packet, all its neighbors will hear it and can receive it).

2) Non-Boundary APL: being non-boundary means that the APL cannot launch (hold) a rolling-ball (Fig. 4(a)). Actually, the ball must be empty of nodes to ensure boundary traversal. In this case, to launch the aggregation process, the APL initializes a shrunken rolling-ball. This ball is centered at the APL and its radius is equal to the distance between the APL and its nearest neighbor (Fig. 4(b)). Setting the ball this way, ensures that the ball is empty of nodes. Sometimes, the distance between the APL and its nearest neighbor can be larger than the optimal radius of the ball ($R/2$). In such a case, the ball must be adjusted to its optimal shape (the radius of the ball must be always $\leq R/2$).

After creating the shrunken ball, the APL marks itself as *visited* and forwards the aggregation packet to its nearest neighbor (Fig. 4(c)). Upon receiving the packet, the nearest neighbor of the APL, checks first the received ball. If the ball is optimal then it simply continues the aggregation as described in the previous section. Otherwise, i.e., if the ball is shrunken then the nearest neighbor of the APL enlarges the ball as much as possible and then continues the aggregation process by spinning the ball (shrunken or optimal) and delivering the packet to the first unvisited neighbor hit by the ball. The same process is applied by any subsequent traversed node. That is, each node that receives the aggregation packet, checks the received ball and acts accordingly.

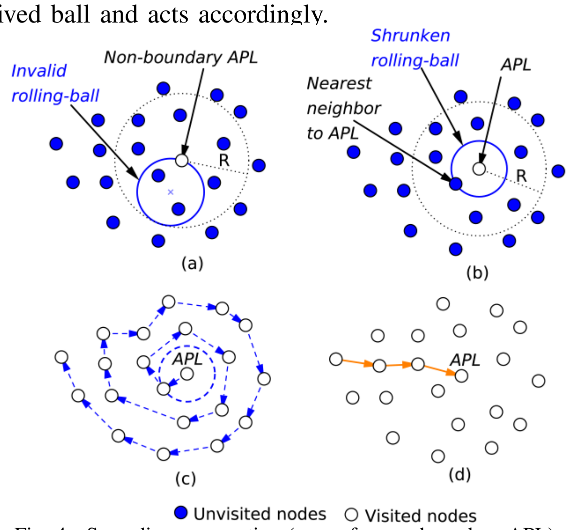

Fig. 4. Spreading aggregation (case of a non-boundary APL).

B. Notion of linking nodes

The aggregation technique described above performs efficiently in dense topologies. However, this technique does not always ensure the aggregation of all data present in the network. The case in which this aggregation technique fails, is when a node connecting two or more parts of Ω, marks itself as visited. For example, in Fig. 5(a), node N_3 ensures the connectivity of two parts of Ω. Removing N_3 from Ω and adding it to Γ (Fig. 5(b)) leads to the disconnectivity of Ω, and therefore it leads to the impossibility of visiting all nodes ($\Omega \neq \emptyset$ and $\Gamma \neq \mathcal{N}$) (Fig. 5(c)).

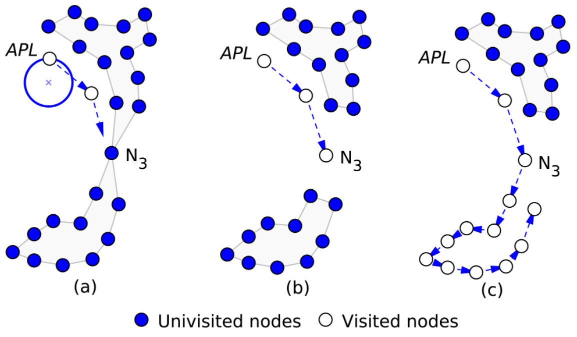

Fig. 5. (a) Node N_3 ensures the connectivity of Ω. (b) If node N_3 marks itself as *visited* (i.e., no longer participates in the aggregation process), Ω will be partitioned and (c) data aggregation will not be complete.

To prevent this scenario from happening, nodes that ensure the connectivity of Ω cannot be marked as visited and must remain involved in the aggregation process. To achieve this end, in addition to the ***unvisited*** and ***visited*** statuses, a new status of nodes has to be introduced, namely the ***linking nodes***.

Definition 5. *Linking Node*
A node is said to be Linking Node, iff at least two of its one-hop unvisited/linking neighbors cannot communicate without its help.

Fig. 6 gives an example of both a linking and a non-linking nodes. Node N_3 in Fig. 6(a) and Fig. 6(b) is a linking node because its neighbors, N_4 and N_5 cannot communicate with N_2 without its help. While, node N_3 in Fig. 6(c) is not a linking node because its neighbors (N_2, N_6, N_4, and N_5) can communicate with each other without its help. We underline that by considering the network model described in Section II, a node can locally (based only on its one-hop neighborhood information) decide whether it is a linking node or not.

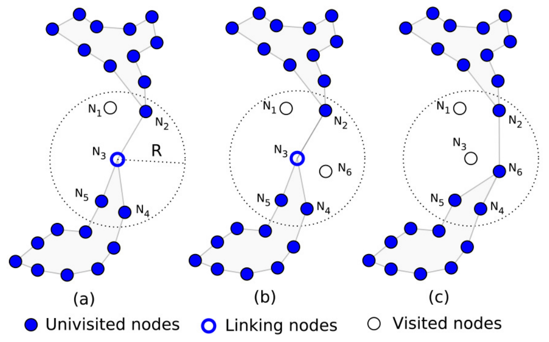

Fig. 6. Example of linking and non-linking nodes.

In order to ensure the connectivity of Ω, linking nodes are not removed from this set and must be re-selected again as next hop. For example, in Fig. 7(b), when node N_3 receives the aggregation packet, it marks itself as *linking node* and forwards the packet to the next hop. When node N_3 receives the aggregation packet for the second time (Fig. 7(c)), it marks itself as *visited* and forwards the packet.

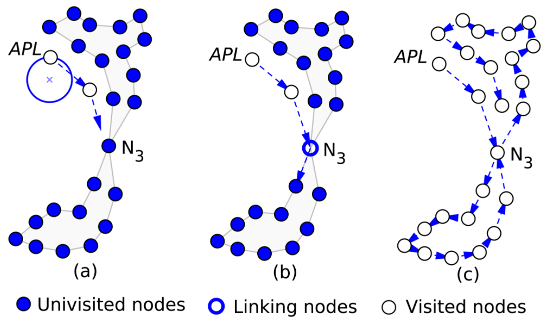

Fig. 7. (a) Node N_3 ensures the connectivity of Ω. (b) N_3 marks itself as linking node and forwards the aggregation packet. (c) Once no longer needed to ensure the connectivity of Ω, N_3 changes its status to *visited*.

C. Linking nodes and looping issue

Cycles are a very common problem in distributed graphs [11]. Due to its reliance on the local limited knowledge of nodes, Definition 5 serves well our objective of proposing a localized aggregation approach. However, this definition creates a looping issue. Precisely, the problem arises in the presence of disjoint boundaries (cycles) in the network. Fig. 8 gives an example in which the aggregation process loops.

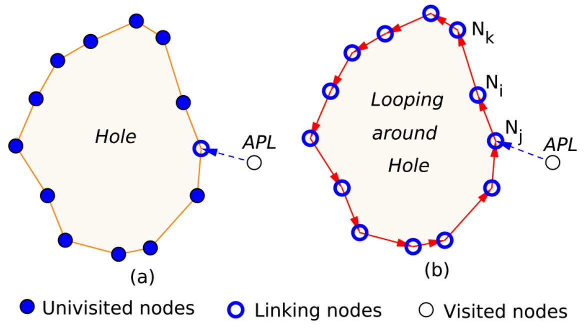

Fig. 8. Example of networks in which the proposed approach loops.

To solve the looping problem, the idea is simple, it consists of opening the cycles, i.e., *ensuring* the existence of one boundary in the whole network. For example, in Fig. 8, there are two disjoint boundaries: the boundary of the network and the boundary of the hole. To avoid looping, these two disjoint boundaries must be connected together (i.e., the boundary of the hole must be opened). To allow nodes to recognize the disjoint boundaries, the initial boundary (from which the aggregation will start), has to be marked. To do so, the APL issues a control packet that traverses the initial boundary node by node. Each traversed node marks locally the initial boundary as *explored*. Actually, each node stores locally its list of boundaries. A boundary is defined by a right side, a left side, and a Boolean indicating if the boundary has been explored. For example, in Fig. 8(b), node N_i has two boundaries. The initial boundary defined by N_j and N_k as its respective right and left sides, has been explored. While, the boundary defined by N_k and N_j as its respective right and left sides, is a disjoint boundary that has not been explored yet.

After having marked the initial boundary, the determination of the disjoint boundaries (cycles) can be done locally by each node. Simply, if the current traversed node has an unexplored boundary, then this boundary is a cycle that must be opened. To do so, we introduce the concept of *portal nodes*.

Definition 6. *Portal Node*
A node is said to be Portal Node iff it has an unexplored boundary.

In the case where the current traversed node is not *Portal* (i.e., has no unexplored boundaries), this node just changes its status (to *linking* or *visited* node) and forwards the aggregation packet. However, in the case where a node is *portal*, the following steps have to be executed (to avoid looping):

- The portal node N_i divides its one-hop neighbors into two sets: the left and right sets.
  - **Left set** ($X \subset \Omega$): contains all neighbors located in the sector defined by the angle $\angle N_{\text{left}}, N_i, N_{\text{prev}}$. Where N_{left} is the left side of the unexplored boundary, and N_{prev} is the previous hop of node N_i. In Fig. 9, all black nodes belong to the left set X. Note that N_{prev} (if it has not been visited), N_i, and N_{left} belong to X.
  - **Right set** ($Y \subset \Omega$): is composed of the rest of neighbors of node N_i including the right side of the unexplored boundary. In Fig. 9, all gray nodes belong to Y.

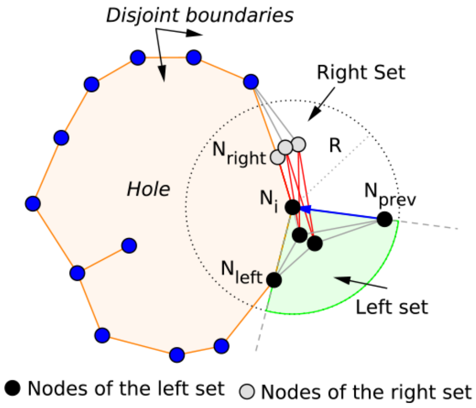

Fig. 9. Left and right sets.

- After creating the left and right sets, the portal node issues a control packet named BSP (*Boundary Scan Packet*), which traverses the unexplored boundary, node by node. Actually, the BSP packet has a twofold role. First, it marks the disjoint boundary as explored, and second, it allows nodes to determine if their local stored boundaries constitute the same boundary. For instance, as Fig. 10 shows, the portal node N_i issues a BSP packet that traverses the boundary of the hole and marks it as explored. In addition to that, the BSP packet informs node N_2 that the boundary defined by N_1 as its right side and the boundary defined by N_3 as its right side, are actually the same boundary. Being aware of this fact, when node N_2 receives the aggregation packet later, it will not consider these two boundaries as disjoint.

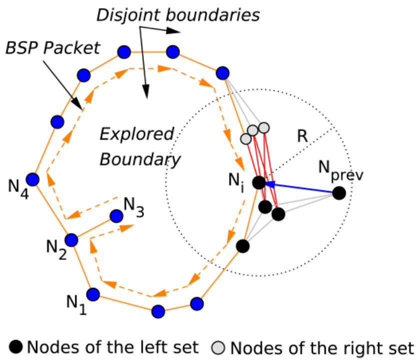

Fig. 10. Boundary Scan Packet.

- After receiving the BSP back, the portal node (which belongs to the left set X) deletes virtually its links with every node of the right set Y. Which means that the portal node N_i will not consider nodes of Y as its neighbors. Formally, the set of links that need to be deleted from $\mathcal{L}$ is: $\mathcal{D} = \{L_{(\text{i,j})} \ or \ L_{(\text{j,i})} \mid N_j \in Y(N_i)\}$.
- Once the appropriate links have been removed, the portal node changes it status to *linking* or *visited* node.
- The last step that needs to be executed by the portal node consists of creating and broadcasting an LCP packet (*Link Cut Packet*) to its immediate neighbors. This packet carries the left and right sets.

To finish the cycle break process, upon receiving the LCP packet, each neighbor of the portal node performs the two following steps. First, it determines to which set it belongs (X or Y). Second, it virtually deletes its links with evey node of the other set (whenever such links exist). Fig. 11 gives an example of boundary opening (cycle break). As described earlier, after receiving the aggregation packet from N_{prev}, the portal node N_i executes the steps described above, and finally broadcasts an LCP packet to its neighbors. After receiving this packet, each neighbor of N_i, cuts the appropriate links. Once the cycle has been broken, the aggregation process continues from the left side of the opened boundary (i.e., N_{left}).

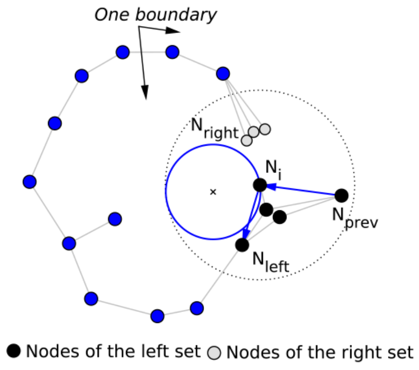

Fig. 11. LCP packet (boundary opening).

The main steps of the proposed distributed approach are summarized in Algorithm 1. We underline that due to the lack of space, we cannot provide in this paper, the formal proofs that demonstrate the correctness of the proposed approach.

That is, Spreading Aggregation terminates (free of looping) and traverses all connected nodes in the network.

IV. Performance Evaluation

In order to evaluate its performance, we have implemented the proposed serial approach on OMNeT++/Castalia [12] [13] and conducted several simulation experiments. In addition to the proposed algorithm, we have implemented a tree-based approach [3] and another serial approach called Peeling Algorithm [2]. As its name indicates, the tree-based approach requires the construction of a tree structure that is rooted at the sink and covers all sensor nodes. Once the tree has been created, leaf nodes can start data aggregation by forwarding their raw data to their corresponding parents. Afterwards, each intermediate node waits to receive data from all its children, aggregates it with its own data, and sends the result to its parent. This operation is repeated until the root. As regards the Peeling Algorithm, this serial approach is based on boundary traversal. More precisely, it uses the Curved Stick Boundary Traversal algorithm [10]. The main drawback of the Peeling algorithm is that the aggregation must start from a node that belongs to the external boundary of the network, which requires a noticeable overhead.

A. Simulation parameters and settings

Simulations were executed in an area of 1000 x 1000 meters where nodes, including the sink (APL), were randomly deployed. To evaluate the scalabilty of the proposed approach, we have varied the number of deployed nodes. We started by 100 nodes, and each time we added (deployed) 50 other nodes up until 500 nodes. Table I summarizes the simulation parameters.

Parameter	Value(s)
Network area	1000 x 1000 m^2
Transmission range of nodes	150 m
Location of the APL (sink node)	Random
Number of nodes	100, 150, 200, …, 500
Deployment of nodes	Uniform
Aggregation packet size	50 Bytes

TABLE I
Simulation parameters

B. Evaluation metrics

To evaluate the performance of the proposed approach and compare it with the two other approaches, we have considered the three following metrics:

- ***1) Required communications:*** total number of packets used to aggregate data (i.e., control and data packets).
- ***2) Aggregation time:*** duration required to aggregate data. That is, the time between the instant when the sink launches aggregation and the instant when it receives the answer.
- ***3) Consumed energy:*** the most important resource in WSNs is energy. This metric measures the total energy used by all nodes to aggregate data. To compute the energy consumed by each node, we have used the energy consumption model proposed in [14]. In this model, the radio consumes $E_{TX}(k, d) = E_{\text{elec}} * k + \epsilon_{\text{amp}} * k * d^2$ to send a k-bit packet a distance d, and it consumes

Algorithm 1 Spreading Aggregation (SA).

```
- Each node creates its list of boundaries (Definition 2).
- The APL calls initializeAggregationProcess();

void initializeAggregationProcess(){
  if (APL is boundaryNode){
    - Set rolling-ball (on any boundary);
    - Send init_packet to mark initialBoundary (using rolling-ball);
  } else { // APL is not boundaryNode
    - Set a shrunken rolling-ball;
    - change status of APL to visited;
    - Send aggregation_packet to nearestNeighbor;
  }
}

void receiveInit_packet(){
   - Upon receiving init_packet, each node marks locally initialBoundary as explored
     and forwards initPacket to nextHop;
   - Upon receiving back the init_packet, APL marks initialBoundary as explored
     and calls continueAggregationProcess();
}

void receiveAggregationPacket(){
   - Upon overhearing aggregation_pkt, each neighbor updates the status of the
     sourceNode in its local neighborsTable;
   - Upon receiving aggregation_pkt, destinationNode:
       - updates the status of the sourceNode in its neighborsTable;
       - aggregates data (if it has been visited for the first time);
       - calls checkRollingBall();
}

void checkRollingBall(){
  if (rollingBall is optimal) { // i.e., rolling-ball radius = $R/2$.
    - Call continueAggregationProcess();
  } else { // rolling-ball is shrunken
      - Enlarge rolling-ball;
      if (rollingBall became optimal) {
          - Call continueAggregationProcess();
      } else { // rolling-ball is still shrunken
          - Spin rolling-ball, change status of currentNode
            and send aggregation_packet to nextHop;
      }
  }
}

void continueAggregationProcess(){
  - Spin rolling-ball;
  if (nextHop is undefined){ // Aggregation process ends here.
    - Change status of currentNode to visited;
    - Send aggregatedData to APL using geographic routing;
    - return;
  }

  if (currentNode is not portalNode){
    - Change status of currentNode to linking or visited node;
    - Send aggregationPacket to nextHop;
  } else { // currentNode is portalNode
    - Create leftSet and rightSet;
    - Send BSP (BoundaryScanPacket) inside disjointBoundary to mark it
      (using rolling-ball);
  }
}

void receiveBSP_packet(){
   - Upon receiving BSP_packet, each node marks locally disjointBoundary as
     explored and forwards BSP;
   - Upon receiving back the BSP_packet, portalNode :
      - Marks locally disjointBoundary as explored;
      - Cuts links with nodes of rightSet;
      - Changes its status to linking or visited;
      - Broadcasts LCP (LinkCutPacket);
}

void receiveLCP_packet(){
   - Upon receiving LCP_packet, each node:
      - Updates locally the status of portalNode;
      - Determines to which set it belongs: right or left;
      - Cuts links with nodes of the other set;
   - Left side of disjointBoundary continues aggregation by:
       - aggregation data (if it has been visited for the first time);
       - calling continueAggregationProcess();
}
```

$E_{RX}(k) = E_{\text{elec}} * k$ to receive this packet. Where $\text{E}_{\text{elec}} = 50 \text{ nJ/bit}$ is the energy consumed to run the transmitter/receiver circuitry, and $\epsilon_{\text{amp}} = 100 \text{ pJ/bit/m}^2$ is the energy consumed to run the transmitter amplifier.

C. Evaluation results

Fig. 12 depicts the communications required to aggregate data by SA (proposed approach), PA (Peeling Algorithm), and the tree-based approach. Fig. 12 depicts also the communications required to build a Hamiltonian path. Theoretically, a network composed of n connected nodes, should be traversed using exactly $n-1$ communications (sent packets or hops). However, realistically not every network contains such Hamiltonian path [5]. Since the number of communications required by SA and PA cannot be expressed using a mathematical expression, the depicted optimal number of communications can be used as a reference to measure the effectiveness of these two serial approaches in terms of communications.

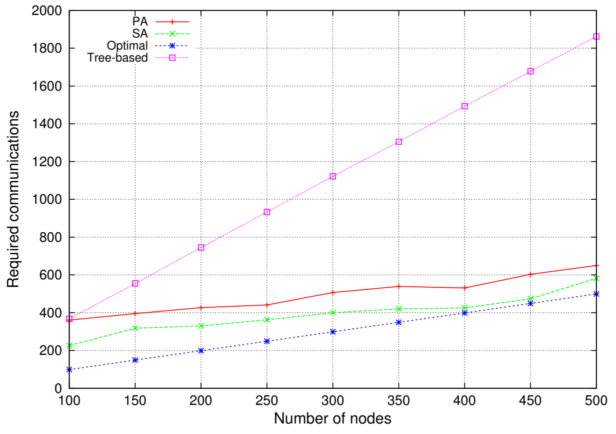

Fig. 12. Required communications to aggregate data.

Fig. 12 shows clearly that our proposed approach (SA) outperforms PA and the tree-based approach, and requires a number of communications that is very close to the optimal number of communications. We mention that for the three approaches, the required communications include (1) the communcations used to construct the structure (tree or path), (2) communications used to diffuse a query through the network, and (3) comunications used to report the aggregated data to the sink. In the tree-based approach, structure construction, query dissemination and data aggregation are three separate tasks. The tree is first constructed. Second, the query is spread throughout the network, then, finally, data is aggregated by sensor nodes. On the contrary, in SA and PA, which are both serial approaches, the three steps are all combined together. While the path is constructed, query is disseminated and data is aggregated at the same time. Clearly, this merge significantly reduces the amount of sent packets (communications), enhances data aggregation time and conserves energy. Fig. 13 and Fig. 14 show respectively the time and energy required to aggregate data by the three considered approaches. Given the fact that there is a big gap in terms of energy consumption between the tree-based approach and the two serial approaches (Fig. 14), we have depicted in Fig. 15, the energy required only by SA and PA (without considering the tree-based approach).

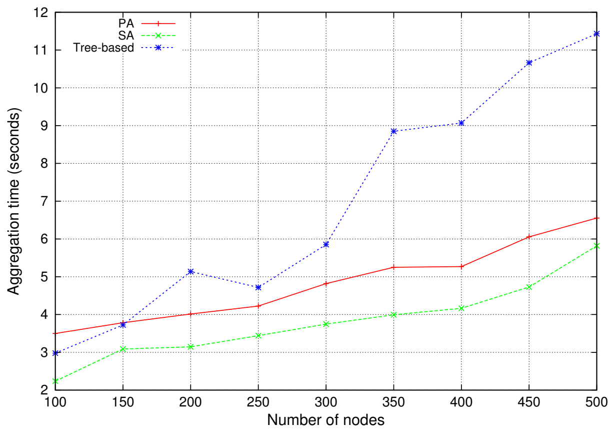

Fig. 13. Time required to aggregate data.

Comparing serial approaches in terms of aggregation time, we can say that given their sequential nature, the more a serial approach requires communications, the more it will require time. As Fig. 13 shows, since SA requires less communications than PA, it outperforms it in terms of aggregation time. However, intuitively, this rule (i.e., less communications implies less time) does not apply to the tree-based approach. Because, in fact, given its concurrent parallel nature, even with a large number of communications, tree-based approach should outperform serial approaches. But, as a matter of fact, as Fig. 13 shows this is not the case. Actually, SA and PA perform better than the tree-based approach.

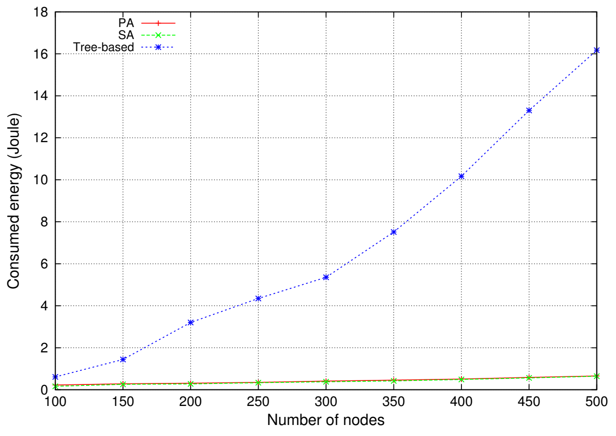

Fig. 14. Required energy to aggregate data.

In terms of energy, as Fig. 14 demonstrates, serial approaches also outperform the tree-based approach. For a wireless sensor node, energy is consumed by the different tasks the node performs, mainly: wireless communications, computation (memory and CPU), and sensing. Considering a simple data aggregation function and a simple sensing operation, communications take over and become the main source of energy consumption. In such a scenario, if we consider only the serial approaches (Fig. 15), we can say that the consumed energy is related to the number of communications. The more a serial approach necessitates communications, the more it will consume energy. Here again, since our approach requires less communications, it performs better than PA in terms of energy consevation (Fig. 15).

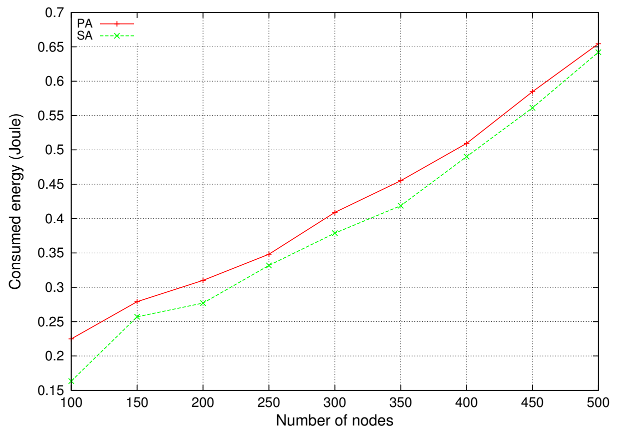

Fig. 15. Required energy to aggregate data (without considering tree-based approach).

To explain the bad performance of the tree-based approach, let us consider a network of n nodes. In this approach, if we assume that each node participates in the query dissemination phase (i.e., rebroadcasting the received query packet) then n packets will be sent. Now, as for data aggregation, it is agreed that each node sends a packet. That is, n other packets need to be sent, which means that in total $2 * n$ packets are required to aggregate data. And this, without taking into account the retransmissions of the collided packets and without counting the cost of building or probably fixing the tree. Certainly, the construction and maintenance of an organizational structure is a non-straightforward process that costs a non-negligible overhead. By against, as regards serial approaches, if we assume the existence of a path that passes exactly once by each node in a network of n nodes, then only $n - 1$ packets will be required to disseminate the query and aggregate data.

In addition to their unsuitability for large-scale deployments and their poor scalability, tree-based approaches suffer from several other issues, among them we cite. First, given their concurrent parallel nature, tree-based approaches witness a lot of collisions especially in large scale and dense networks. Collisions waste energy and time because collided packets need to be re-transmitted. Second, due to (1) collisions, (2) tree construction, and (3) tree maintenance, tree-based approaches require a remarkable overhead and consequently they considerably consume the energy of nodes. Third, tree-based approaches suffer from the unbalanced energy consumption problem. This problem occurs because nodes that are close to the sink are overused to relay the traffic [15]. Finally, a tree is a set of pre-determined paths. Thereby, tree-based approaches are very sensitive to links and nodes' failures. Any failure necessitates the maintenance of the tree, which could require a considerable amount of energy and time particularly in dense large-scale networks.

V. Conclusion

Compared with tree-based aggregation, serial aggregation has shown its effectiveness and outperformance in terms of time and energy, however, due to the path-construction complexity, serial approaches are not optimal and can be further enhanced. To achieve this end, we proposed in this paper a serial approach called Spreading Aggregation (SA). The efficiency of this approach comes from its localized nature, i.e., its reliance on the one-hop neighborhood information of each node. The obtained simulation results confirm the efficiency of the proposed approach in terms of time and energy, and confirm also that the proposed approach ensures aggregation completeness. Nevertheless, the proposed approach needs to be proven. That is: (1) it visits all connected nodes in the network, and (2) it terminates and does not loop indefinitely. Actually, we have already proven these two points, but due to the lack of space, we cannot present the proof of correctness in this paper.

References

[1] A. Boukerche, *Algorithms and Protocols for Wireless Sensor Networks*, ser. Wiley Series on Parallel and Distributed Computing. John Wiley & Sons, 2008.

[2] A. Mostefaoui, A. Boukerche, M. A. Merzoug, and M. Melkemi, "A scalable approach for serial data fusion in wireless sensor networks," *Computer Networks*, vol. 79, pp. 103–119, 2015.

[3] R. Rajagopalan and P. K. Varshney, "Data aggregation techniques in sensor networks: A survey," *IEEE Comm. Surveys & Tutorials*, vol. 8, pp. 48–63, 2006.

[4] M. Li, Y. Wang, and Y. Wang, "Complexity of data collection, aggregation, and selection for wireless sensor networks," *IEEE Transactions on Computers*, vol. 60, no. 3, pp. 386–399, 2011.

[5] M. G. Rabbat and R. D. Nowak, "Quantized incremental algorithms for distributed optimization," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 4, pp. 798–808, 2005.

[6] A. Boukerche, H. Oliveira, E. F. Nakamura, and A. A. F. Loureiro, "Localization systems for wireless sensor networks," *IEEE Wireless Communications*, vol. 14, no. 6, pp. 6–12, 2007.

[7] A. Boukerche, X. Fei, and R. B. Araujo, "An optimal coverage-preserving scheme for wireless sensor networks based on local information exchange," *Computer Communications*, vol. 30, no. 14, pp. 2708–2720, 2007.

[8] A. Boukerche and X. Fei, "A coverage-preserving scheme for wireless sensor network with irregular sensing range," *Ad hoc networks*, vol. 5, no. 8, pp. 1303–1316, 2007.

[9] W.-J. Liu and K.-T. Feng, "Greedy routing with anti-void traversal for wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 8, no. 7, pp. 910–922, 2009.

[10] A. Mostefaoui, M. Melkemi, and A. Boukerche, "Localized routing approach to bypass holes in wireless sensor networks," *IEEE transactions on computers*, vol. 63, no. 12, pp. 3053–3065, 2014.

[11] A. Boukerche and C. Tropper, "A distributed graph algorithm for the detection of local cycles and knots," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 8, pp. 748–757, 1998.

[12] "OMNeT++ : Simulation Environment," http://www.omnetpp.org/.

[13] "Castalia : Wireless Sensor Network Simulator," http://castalia.research.nicta.com.au/index.php/en/.

[14] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Transactions on Wireless Communications*, vol. 1, no. 4, pp. 660–670, 2002.

[15] A.-F. Liu, P.-H. Zhang, and Z.-G. Chen, "Theoretical analysis of the lifetime and energy hole in cluster based wireless sensor networks," *Journal of Parallel and Distributed Computing*, vol. 71, no. 10, pp. 1327–1355, 2011.