People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University A. MIRA - BEJAIA

Faculty of Exact Sciences
Department of Computer Science

THESIS

Presented by

MOHAMMED AMINE **MERZOUG**

To obtain the degree of

DOCTOR OF SCIENCE

Specialty: Computer Science
Option: Cloud Computing

Title

LOCALIZED ADAPTIVE ROUTING IN WIRELESS SENSOR NETWORKS

Defended on: 30/04/2019

Before the jury composed of:

ABDELOUHAB **ALOUI**	Assoc. Professor	Univ. of Bejaia	President
AHMED **MOSTEFAOUI**	Assoc. Professor	Univ. of BFC, France	Reviewer
ABDERRAHMANE **BAADACHE**	Assoc. Professor	Univ. of Algiers 1	Examiner
ABDELMALEK **BOUDRIES**	Assoc. Professor	Univ. of Bejaia	Examiner
HAMOUDI **KALLA**	Full Professor	Univ. of Batna 2	Examiner
HAMOUMA **MOUMEN**	Assoc. Professor	Univ. of Batna 2	Examiner

Academic Year: 2018/2019

THESIS presented by
MOHAMMED AMINE **MERZOUG**

to obtain the degree of

Doctor of Science

Specialty: Computer Science

Localized Adaptive Routing in Wireless Sensor Networks

Research Units:
1. Department of Computer Science, Faculty of Exact Sciences, University of Bejaia, 06000 Bejaia, Algeria,
2. Department of Computer Science and Complex Systems (DISC), AND Team, FEMTO-ST Institute, UMR CNRS 6174, University of Burgundy Franche-Comte, 25200 Montbeliard, France.

Defense date: 30/04/2019

جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

femto-st
SCIENCES & TECHNOLOGIES

Praise be to Allah, The Lord of the worlds,
and Peace and Blessings be upon our Prophet Muhammad PBUH,
his Family and Companions,
and those who follow his guidance.

"And lower to them the wing of humility out of mercy and say, My Lord, have mercy upon them as they brought me up [when I was] small." Quran [17:24]

To the memory of my Beloved Precious Mother who passed away when I was writing this manuscript. She was very proud of me, may Allah SWT, in his infinite mercy, forgive her sins and grant her Jannah Al-Firdous.

To my Dear Father, may Allah SWT protect him and lengthen his life,

To my Brothers and Sisters, especially the little Maria,

To all my Family.

ACKNOWLEDGMENTS

"And We have enjoined upon man [care] for his parents. His mother carried him, [increasing her] in weakness upon weakness, and his weaning is in two years. Be grateful to Me and to your parents; to Me is the [final] destination." Quran [31:14]

Now that the stress is starting to fade away, and the light at the end of the thesis tunnel is emerging, I chose to take some time alone to remember all I have been through these last couple of years and all the people who helped me during this period of my life. Actually, in these moments of relief and appreciation, I would like to start by expressing my profound thanks and my sincere gratitude to my esteemed supervisor, Dr. Ahmed Mostefaoui, Associate Professor at the University of Burgundy Franche-Comté. I am deeply indebted to him for his time and his unwavering guidance and support. In fact, I would like to thank him first and foremost for his trust in my abilities and for the chance of working with him and his research team. For me, Dr. Ahmed is more than a thesis director, I consider him as a member of my family. He showed me how to step out of my comfort zone, and taught me to embrace fear and challenge my limits. Just when you think, you have finished, Dr. Ahmed will show you how one can dig deep down and find more. My discussions with him, his constructive criticism, and more particularly his insight helped me become if I may say, not only a better researcher and computer scientist but also a better person. No matter what I say, I cannot thank Dr. Ahmed enough. May Allah SWT bless him and his entire family.

I also would like to take this opportunity to thank the research team with whom I have and am still working, particularly Prof. Azzedine Boukerche, Full Professor at the University of Ottawa, and Dr. Samir Chouali, Associate Professor at the University of Burgundy Franche-Comté. It is a true honor to work under the invaluable guidance of Professor Boukerche. I warmly and particularly thank the University of Bejaia for accepting my application and allowing me to work under the supervision of a foreign thesis director. I, as well, would like to thank the University of Burgundy Franche-Comté, especially the FEMTO-ST Institute and the DISC Department, for welcoming me with kindness and open arms.

I sincerely thank the jury members who offered their precious time and kindly accepted to evaluate my thesis and attend its defense.

I offer my special thanks to my favorite teacher Prof. Tahar Bensebaa, Full Professor at the University of Annaba. Actually, thanks to him, I discovered the beauty and fascination of the world of algorithms, programming, and optimization. To be honest, this whole journey would not have been possible also without the help of my dear brother and training partner Ali Beddiaf. Thank you so much, Ali. There is no way I can forget my dear friends; Abdelghani Boubram, Abderrezak Benyahia, Amine Barkat, Chafiq Titouna, Toufik Baroudi, and the list goes on and on. A big thank you to my friend Omar Barkat. You are a great source of inspiration and motivation. I thank my friends in France; Amir Haroun, André Naz, and Ridha Ouaguelal. Thank you so much, guys, for opening your hearts and houses, and thank you so much for the warmth of your welcome and hospitality. My special thanks go also to my dear Nigerien friend Issa Abdoua. Thank you so much, man, for all the great funny moments we had in Montbéliard, and in France in general. This work would not have been possible without the support of many other people. So, I gratefully acknowledge and thank all those who have in one way or another helped me attain my current position of Assistant Professor and contributed to the completion of this humble work.

To those whose names do not appear in this manuscript, I wholeheartedly apologize and say thank you.

Sincerely,
Mohammed A.

LIST OF PUBLICATIONS

Journal Articles

- **Mohammed Amine Merzoug**, Azzedine Boukerche, Ahmed Mostefaoui, and Samir Chouali. Spreading Aggregation: A Distributed Collision-free Approach for Data Aggregation in Large-Scale Wireless Sensor Networks. ***Journal of Parallel and Distributed Computing***,125:121–134, March 2019.

- **Mohammed Amine Merzoug**, Azzedine Boukerche, and Ahmed Mostefaoui. Efficient Information Gathering from Large Wireless Sensor Networks. ***Computer Communications***, 132:84–95, November 2018.

- Ahmed Mostefaoui, Azzedine Boukerche, **Mohammed Amine Merzoug**, and Mahmoud Melkemi. A Scalable Approach for Serial Data Fusion in Wireless Sensor Networks. ***Computer Networks***, 79:103–119, March 2015.

Conference Articles

- **Mohammed Amine Merzoug**, Azzedine Boukerche, and Ahmed Mostefaoui. Serial In-network Processing for Large Stationary Wireless Sensor Networks. In *Proceedings of the 20th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, **MSWiM'17**, Miami, Florida, USA, November 21-25, 2017*, pages 153–160.

- **Mohammed Amine Merzoug**, Ahmed Mostefaoui, and Samir Chouali. Distributed Collision-free Data Aggregation Approach for Wireless Sensor Networks. In *13th IEEE International Conference on Distributed Computing in Sensor Systems, **DCOSS'17**, Ottawa, Ontario, Canada, June 5-7, 2017*, pages 175–182.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF DEFINITIONS

LIST OF ACRONYMS

$L_{(i,\ j)}$	Communication link between node N_i and N_j	5
Q_j	Query intended for the whole WSN	75, 77, 79, 80
R	Communication range of nodes	5, 18, 19, 51, 53, 57, 63, 76
X	Left set of detected cycle	58–61
Y	Right set of detected cycle	58–61
Γ	Set of all currently visited nodes	49, 61
Ω	Set of all currently unvisited nodes	xiii, xiv, 25–31, 33, 35–38, 49, 51, 52, 54, 55, 57, 58, 60–62, 77, 78, 83, 84
$\mathcal{N}$	Set of all nodes in the network	5, 25, 36, 37, 49, 52, 61, 62, 64, 75
$\mathcal{V}_i$	Neighbors set of node N_i	5, 22, 78, 79, 81–83
$\mathcal{W}_i$	Active neighbors set of node N_i	78, 83
n	Number of nodes in the network	5, 37, 61, 73
$\mathcal{L}$	Set of all links in the network	5
ACN	Actual-Cut Node (Definition 16)	xix, 78–82, 84
BN	Boundary Node (Definition 3)	xix, 22, 23, 29

BRN	BRidge Node (Definition 6)	xix, 25–29, 34–38
BS	Back Set of detected cycle	31–33
BTI	Boundary Traversal Initiator	xiii, 18
CN	Cyclic Node (Definition 12)	xix, 38
CS	Curved-Stick (boundary traversal tool)	16–18
DBS	Disjoint-Boundary Scan Packet	xiv, 51, 58–61, 70, 71, 87
DFS	Depth-First Search	13, 39, 41, 43, 69, 70, 86–88
EPA	Enhanced Peeling Algorithm (enhanced version of first proposed approach)	6, 7, 21, 38, 39, 45, 46
FS	Front Set of detected cycle	30–33
GAR	Greedy Anti-void Routing	16–19
GBT	Greedy-Boundary Traversal	15, 48, 69, 70, 86, 87
GSS	Geometric Serial Search (third proposed approach)	6–8, 73–76, 79, 83–92, 94, 95
HCP	Hole Control Packet	xiii, 32–34
HGN	Hole Gate Node (Definition 10)	xiii, xix, 29–36
HLN	Hole Left Neighbor	29–34
HRN	Hole Right Neighbor	29, 30, 32, 33
IBS	Initial-Boundary Scan Packet	xiv, 50, 51, 55, 56, 59, 61, 71

LC	Link-Cut Packet	xiv, 51, 59, 60
LN	Linking Node (Definition 13)	xix, 52
NBN	Network Boundary Node (Definition 4)	xix, 23, 29, 34
PA	Peeling Algorithm (first proposed approach)	6–8, 21–23, 32, 35–39, 41, 43, 45, 46, 69, 70, 74, 86, 87, 94, 95
PBN	Potential Boundary Node (Definition 7)	xix, 27, 28, 34
PCN	Potential-Cut Node (Definition 17)	xv, xix, 78–84
PHBN	Potential Hole Boundary Node (Definition 9)	xix, 29, 30
PN	Portal Node (Definition 14)	xix, 55
PSB	Parallel Structure-Based	1–3, 6, 7, 11–13
PSF	Parallel Structure-Free	1–3, 6, 7, 13
QL	Query Launcher	xiv, 5, 13, 15, 23, 24, 39, 40, 49–51, 53–58, 62, 63, 65, 78
SA	Spreading Aggregation (second proposed approach)	xiv, 6–8, 47–50, 61, 63–68, 70, 71, 74, 75, 86, 87, 94, 95
SN	Starting Node	23–25

SSB	Serial Structure-Based	1–3, 6, 7, 13
SSF	Serial Structure-Free	1–3, 6, 7, 14, 15, 79, 84, 89–91
WQR	Window Query Region	95, 96
WSN	Wireless Sensor Network	1, 2, 6, 7, 9–11, 13–16, 21, 24, 39, 46–48, 64, 65, 70, 73, 84, 85, 88, 92–95

1 INTRODUCTION

1.1/ THESIS CONTEXT

Typically, a Wireless Sensor Network (WSN) is composed of hundreds to several thousands of smart autonomous sensor nodes which are randomly deployed in an area of interest. Using their sensing, computing and wireless communication capabilities, sensor nodes respond to the application/end-user needs by collecting and reporting the required environmental information (such as temperature, movement, etc.) to a specific more powerful node known as the base station or simply the sink [1, 2]. Actually, regardless of its inefficiency, raw data gathering from sensor nodes is practically not a purpose on its own. Rather than collecting all raw data at the sink, usually, the objective is to get a lower/upper value or to derive an estimate of a parameter of interest as fast and effectively as possible [3,4] (e.g., minimum, maximum, average reading [5], alive nodes count, target location [6–8], etc.). To achieve this end and attain time/energy efficiency, out-of-network data gathering (raw data collection) has been disregarded and much research has been focused on finding in-network approaches which involve sensor nodes in processing and consider the network as a distributed database [3, 9].

Given its interesting features and advantages, in-network processing has been established as a very efficient technique in WSNs, and numerous approaches have been proposed in this regard [9]. In fact, based on how the different tasks (such as processing, communications, etc.) are performed by sensor nodes, the in-network approaches proposed in the literature can be categorized into two major classes: *parallel concurrent* and *serial incremental*. In turn, actually, both parallel and serial approaches can also be divided into *structure-based* and *structure-free*. These four approaches, which we refer to in this manuscript respectively as PSB, PSF, SSB, and SSF are briefly presented in the following along with some of their advantages and drawbacks.

- **Parallel structure-based approaches (PSB)**, which are also known as *in-network centralized* approaches, operate in three separate phases: structure construction, query dissemination, and data processing [3, 9, 10]. For example, at first, a span-

ning tree rooted at the sink must be created. Once the whole network is covered, queries can then be spread throughout the tree ordering nodes to perform a certain processing on data. After query dissemination phase, data processing starts from leaf nodes and goes up towards the root/sink. In fact, before sending the result to the upper level, each node aggregates its children's data with its own reading.

Besides the fact that these approaches are not scalable and, consequently, are not suitable for large-scale deployments, they suffer from several other drawbacks among which we briefly cite: overuse of the network resources, mainly through communications. Generation of a high degree of collisions, especially in large-scale and dense networks. Creation of the energy depletion hole problem by relying the traffic on nodes that are close to the sink/root [11, 12]. Finally, low resilience to failures in nodes and links, as any link or node failure will require maintenance or complete reconstruction of the entire tree-structure, in particular, if the broken link or node is near the root. These drawbacks become very costly in large-scale deployments.

- **Parallel structure-free approaches (PSF):** actually, in large-scale WSNs, the two last limitations of PSB approaches cited above can have a deeply negative impact on the overall performance of information gathering. To overcome these drawbacks, *parallel structure-free* approaches, also known as *in-network distributed* approaches, were proposed as an interesting alternative [3, 9, 13–15]. In such approaches, each node maintains a local estimate of the unknown parameter. This estimate is *refined* successively (iteratively) through one-hop communications until convergence to the right value (which is attained when the difference between two consecutive estimates is less than the predefined convergence threshold). In these approaches, since all communications are one-hop, nodes do not need to have any knowledge about the current network topology. Furthermore, since at the end (after convergence) each node holds the desired estimate, there is no need for a central base station, as there is no need for any kind of routing. Finally, given their structure-free nature, these approaches are very robust against failures in links and nodes [16, 17].

  Even though PSF approaches are independent of any rooting structure, they remain dependent on the network topology. For instance, whether in large-scale or in sparse deployments (e.g., linear topologies), the convergence of the unknown parameter can require a lot of iterations, leading thus to a huge communication overhead. Knowing that communications are the most energy-consuming task in WSNs [1], this limitation of PSF approaches can seriously compromise the whole network lifetime. What is worse is the fact that concurrent communications between nodes not only increase collisions but also considerably augment the query response time [18].

- **Serial structure-based (SSB) and serial structure-free (SSF) approaches:** in WSNs, a serial in-network processing algorithm browses nodes one by one and can perform different tasks such as: creating a schedule among nodes, querying or gathering data from nodes, supplying nodes with data, etc. Actually, as is the case for parallel techniques, we distinguish two possible approaches: *serial structure-based* and *serial structure-free*. While in SSB approaches, the query is sequentially processed from node to node following a *preset* path (that crosses every node in the network), in SSF approaches, the visiting path is *gradually* built and

each visited node must be able of autonomously choosing the next hop. In both approaches, the information gathering process stops when all nodes in the network have been visited (i.e., contributed to the query), and the final node in the path holds the answer to the query [19, 20].

Similar to the PSB approaches case, using a predefined path contributes to the vulnerability of serial structure-based techniques to topology changes and failures in links/nodes. For instance, if at some point the next predefined hop is unavailable, the traversal process will inevitably stop at the currently explored node.

On the other hand, compared with PSB and PSF, or even with SSB approaches, serial structure-free algorithms achieve superior performances. In a nutshell, among others, the main features that differentiate serial structure-free algorithms from their counterparts are (1) compactness, (2) collision-freeness, and (3) structure-independency. First of all, one of the main reasons behind the bad performance of PSB approaches is their mode of operation. As previously mentioned, these approaches operate in three separate phases: structure construction, query dissemination, and data processing. Performing these three tasks separately not only increases energy consumption but also delays the response time. With regard to SSF algorithms, energy and time are considerably saved through the combination of the three previous phases into one step. While the path is being gradually laid out throughout the network, at the same time, the query is disseminated and data is processed. Second, in SSF algorithms, the desired task is executed sequentially by each node while the network is gradually traversed. In other words, only one node is allowed to communicate at any given moment in time. Hence, serial algorithms are inherently collision-free[1] and no elaborated MAC layer is needed in these algorithms. Consequently, a considerable improvement can be made in terms of responsiveness. As regards tree-based approaches, the network, in this case, is traversed in a parallel fashion. So, from a theoretical point of view, we can say that this feature would give an advantage to tree-based approaches and enhances their response time. In reality, however, the parallel traversal creates a lot of collisions, which considerably wastes time and dissipates energy, especially in large-dense networks. In fact, even the tree construction process is deeply affected by collisions. Finally, the main concern in structure-based approaches is topology changes because rebuilding or fixing a structure that covers a large dense network is a very time and energy-consuming task. Unlike structure-based approaches, SSF algorithms do not rely on any pre-established structure (no path is built in advance). Instead, each time a query is issued, a new path will be gradually built by each traversed node. This characteristic makes SSF algorithms more resistible to topology changes and links/nodes failures.

1.2/ MOTIVATIONS

Apart from the theoretical foundations of serial approaches (i.e., convergence proofs in the case of a parameter/function estimation) which have been provided and discussed in [19–21], several other practical issues of serial approaches still remain to be addressed and deserve additional research efforts, especially in large-scale and randomly deployed

[1] They do not generate any communication collisions because all communications in the network are serial.

networks. As a matter of fact, the operation of serial approaches requires the construction of a Hamiltonian path; that is, a path that passes through all nodes in the network and visits each one of them just once [20]. Yet, previous research has made implicit assumptions and supposed that such a path exists [20,22], which is not true for every configuration. In fact, even when a Hamiltonian path exists in a network of wireless nodes, finding it constitutes an NP-Complete problem [23]. Furthermore, constructing such a path gradually in a decentralized fashion while ensuring scalability can generate a prohibitive overhead, particularly in sparse and large-scale networks. For instance, to overcome this obstacle, the authors in [24] used space-filling curves to derive a path that is not necessarily Hamiltonian (nodes can be visited more than once). This serial approach, although it performs well in dense and regular networks, cannot, in fact, handle irregular network topologies, especially those with communication holes. In more specific words, this approach does not ensure that all connected nodes in the network will contribute to the query, hence, it does not ensure query completeness, which is very harmful to sensitive applications where query accuracy is an essential requirement.

To recapitulate, we can say that despite the fact that serial approaches outperform parallel ones in medium and large-scale network deployments, they still raise challenging research issues, primarily in the way the visiting path must be constructed. Actually, the performance of a serial approach benefits from a shorter visiting path, and *vice versa*. In this manuscript, the motivations that drive us is the design of novel serial structure-free algorithms that can draw shorter visiting paths (in comparison with state-of-the-art approaches), and exhibits better performance (i.e., less energy and time consumption) while satisfying the three following requirements:

- Robustness: using a pre-constructed path makes the serial approach very vulnerable to failures in links and nodes and completely unable to handle topology changes, as is the case for tree-based and itinerary-based approaches [24]. Actually, in such a scenario, when selecting the next hop, the currently traversed node has no choice but to follow the predefined path. Hence, for example, if the next predetermined hop is unresponsive, the node in question will have no alternative except to stop the aggregation/querying operation.

  In order for the proposed approach to support topology changes and be robust against link and node failures, it has to be structure-free, and the visiting path must be gradually constructed at each traversed node. In other words, instead of attributing a next hop to each node, the currently traversed node must be capable of autonomously selecting the next hop.

- Scalability: in order to be highly scalable and support a very large number of nodes (which is a fundamental requirement in large-scale deployments), the proposed approach has to be localized; that is, when selecting the next query hop, the currently traversed node must rely only on its local one-hop neighbors table, and no other additional information must be needed.

- Completeness: provided that the network is connected, the proposed approach has to be able to traverse any possible topology (e.g., with or without holes, regular or irregular topologies, etc.) and ensure that all nodes are queried (aggregate the data of all nodes). This requirement is fundamental in many practical applications, particularly sensitive ones where the aggregation result must involve all nodes of the network.

These requirements were not supported by the previous serial approaches. For instance, as mentioned above, the itinerary-based approach in which the visiting path is predefined through the use of space-filling curves [24] does not ensure the completeness requirement in some network deployments with holes (an example of such a case has been provided in [25]). In fact, even predefined paths with backtracking possibilities present a weak robustness in face of link and node failures [26]. On the other hand, as confirmed by the numerous performance evaluation studies we have conducted, other serial approaches like the one presented in [25] have the advantage of ensuring the aforementioned requirements, at the expense of drawing longer visiting paths.

1.3/ ASSUMPTIONS AND THESIS OBJECTIVE

We consider networks composed of a finite set of n connected stationary wireless nodes, denoted $\mathcal{N}$ such that $\mathcal{N} = \{N_1, N_2, \ldots, N_n\}$. All nodes in the network are assumed to have the same communication range[2], denoted R [27, 28]. The finite set of possible wireless non-oriented links between nodes is denoted by $\mathcal{L}$ such that $\mathcal{L} = \{L_{(i,\ j)} \mid i \neq j \wedge N_i, N_j \in \mathcal{N} \wedge distance(N_i, N_j) \leq R\}$. We assume that all links are bidirectional; that is to say, if link $L_{(i,\ j)}$ belongs to $\mathcal{L}$, then this implies that $L_{(j,\ i)}$ also belongs to $\mathcal{L}$. In other words, it is assumed that nodes located within the communication range of each others can communicate. We suppose that each node is aware of (1) its own location information via a positioning system such as GPS or by means of any other efficient localization technique [29–34] and (2) its direct (one-hop) neighbors and their corresponding locations. For each node N_i, the neighbors set is defined as $\mathcal{V}_i = \{N_j \mid L_{(i,\ j)} \in \mathcal{L}\}$.

We consider the query mode (also known as the *push mode* [1]), in which the end-user/application sends a query to the network and waits for a response. In this manuscript, we refer to the node that issues queries (triggers serial aggregation process) as the *Query Launcher* (QL), and we assume that this node can be located anywhere in the deployment field (i.e., it can be a boundary or non-boundary node).

Our objective is to start from any node and be able to traverse the entire network using one single packet. The latter must jump sequentially from node to node and browse the network while reducing communications to the extent possible. That is, minimizing or avoiding the visit of any node more than once. Also, in order to reduce communications, the next hop of the packet must be determined locally by each traversed node using only its local pre-collected one-hop neighbors table (no extra communications or collaboration between nodes should be required). In simple words, the problem that we are trying to solve can be boiled down to a distributed graph traversal. We recall that perfectly, a network with n connected nodes should be traversed using exactly $n-1$ communications. Thus, theoretically, the path must cross every node precisely once. But, realistically, not every graph or network contains such optimal Hamiltonian path [20], and even if it does, determining that path constitutes an NP-complete problem [23].

[2] In practice, the communication range of nodes can be set to a pessimistic value (e.g., the worst case communication range in the network).

1.4/ CONTRIBUTIONS

In this manuscript, we propose three efficient serial structure-free algorithms that fulfill all the requirements mentioned in Section 1.2 (i.e., robustness, scalability, and completeness). Actually, as for any distributed localized algorithm [35, 36], we have proven the correctness of the proposed algorithms. More precisely, we provide in this manuscript proofs which demonstrate that our proposed solutions (1) terminate and do not loop endlessly and (2) visit all connected nodes in the network (i.e., ensure query completeness). In addition to the theoretical proofs, we have conducted several series of simulations in order to assess/compare our proposed serial algorithms and highlight their good performances. The comparisons have been made with different state-of-the-art algorithms (PSB, PSF, SSB, and SSF approaches). The obtained simulation results confirm the efficiency of our three proposals and also validate their adequacy for large-scale network deployments.

The first proposed algorithm, called *Peeling Algorithm* (PA) [18], is a serial data fusion approach based on boundary traversal. In reality, the peeling term comes from the fact that the traversal must start from the external boundary of the network; then gradually, this external boundary (layer) of nodes is removed (marked as visited), hence revealing a new internal layer which will be peeled in its turn, and so on. More precisely, the sink node must first determine the starting point (node) of the peeling process, which is any node located on the external boundary of the network. Once found, the visit of nodes begins from this node through the use of a graph-free boundary traversal algorithm, called Curved-Stick [37, 38] (Section 2.3.2.1). Actually, we have also proposed an enhanced version of the peeling algorithm, called *Enhanced Peeling Algorithm* (EPA). This modified version has been specifically tailored for non-uniform network deployments.

The second proposed algorithm, called *Spreading Aggregation* (SA) [39, 40], is a serial data aggregation approach that starts information gathering by simply setting a rolling ball [41] (Section 2.3.2.2) and letting it sequentially explore the network. Unlike the Peeling technique [18], in SA, any node can issue queries without the need for external boundary determination. In fact, in this boundary-based SSF approach, even a non-boundary node can launch queries by simply creating and launching a shrunken rolling-ball. The latter gets gradually enlarged at each hop until gaining its optimal shape.

The third proposed approach, called *Geometric Serial Search* (GSS) [42, 43], is a serial processing technique that does not require any control packets. As a matter of fact, in this algorithm, just one data packet that can be issued by any node moves from node to node and traverses the entire network. As confirmed by the obtained simulation results, in most of the cases, GSS approximates the optimal traversal path, which means that GSS scales well in large networks and significantly saves energy and time. For example, for a network of 500 nodes, GSS requires approximately 510 hops to visit each and every node.

1.5/ THESIS STRUCTURE

This manuscript is divided into six chapters, as follows:

- Chapter 1: thesis context, motivations, thesis objective, contributions, and thesis structure.
- Chapter 2: query processing, boundary traversal, and generalities about WSNs.

- Chapter 3: peeling algorithm (first contribution).
- Chapter 4: spreading aggregation (second contribution).
- Chapter 5: geometric serial search (third contribution).
- Chapter 6: general conclusion and future works.

Chapter 1 (current chapter) discusses the considered problem, the motivations that drive this work, the considered assumptions (network and communication models, ...), thesis objective, and the main contributions proposed to solve the treated problem.

Chapter 2 introduces the necessary background and preliminaries required to understand the considered problem as well as its solving. More specifically, this chapter provides a brief introduction about Wireless Sensor Networks and their major challenges/issues. Furthermore, in addition to presenting and describing boundary traversal techniques in WSNs (which are the building block of our three proposed approaches), this chapter also gives the operation, strengths, and weaknesses of the four main types of aggregation techniques proposed in the literature, namely, SSF, SSB, PSF, and PSB approaches.

Chapters 3, 4, and 5 present respectively our three contributions (namely; Peeling algorithm, Spreading Aggregation and Geometric Serial Search) and details their principle of operation through algorithms and simple illustrative figures. These chapters also depict and interpret the obtained simulation results. More specifically:

- In Chapter 3, we present our novel serial approach, called *Peeling Algorithm* (PA), along with its second version, called *Enhanced Peeling Algorithm* (EPA), which has been specifically designed for non-uniform networks [18]. In this chapter, we also provide proofs that these two algorithms (1) terminate and do not loop indefinitely, and (2) ensure query completeness by visiting all connected nodes. In order to assess its performance, we have conducted several series of experiments and compared PA with state-of-the-art parallel structure-based, parallel structure-free, and serial approaches. The obtained results, presented in this chapter, confirm the efficiency of our peeling approach and also assert the effectiveness of EPA in non-uniform large-scale deployments.

- In Chapter 4, we present our second serial processing approach, called *Spreading Aggregation* (SA) [39, 40]. Actually, as is the case for PA, this approach has been also specifically designed to support very large WSNs. In addition to its structure-free and localized design, the proposed approach (i.e., SA) has been proven to fulfill the third design requirement cited in Section 1.2 (i.e., completeness). More precisely, in this chapter, as we did with the peeling approach, we formally prove the correctness of SA; i.e., it terminates and visits all connected nodes without falling into looping. We also provide the results of the conducted simulations. The latter highlight the very good scalability and efficiency of the proposed approach in terms of time and energy in comparison with other serial approaches, especially in very large-scale network deployments.

- In Chapter 5, we present our third scalable SSF approach that lays out very short visiting paths and reduces communications to the maximum extent possible [42, 43]. The proposed approach, called *Geometric Serial Search* (GSS), has been purposely designed to support very large, medium, or even small-scale WSNs. The proposed approach is totally localized and does not require any control packets. Actually, as the conducted evaluations (presented in this chapter) confirm, GSS

yields shorter visiting paths when compared with state-of-the-art approaches, and always approximates the optimal number of communications required to traverse a network of n nodes (i.e., $n - 1$ packets). Furthermore, similar to PA and SA, GSS combines path construction, query diffusion, and data fusion (while exploring the network, nodes are queried and their answers are simultaneously gathered). The evaluations studies we have conducted confirm that GSS is very scalable and has better performance in terms of energy and time reduction. The conducted evaluations also assert that GSS ensures aggregation completeness and query accuracy.

In the end, Chapter 6 concludes the manuscript and suggests some possible future directions.

2 WIRELESS SENSOR NETWORKS

2.1/ GENERALITIES

A wireless sensor network (WSN) is composed of several wireless sensor devices deployed in order to autonomously collect and transmit environmental information to one or more collection points called sinks or base stations. The self-configuration and operation features, which eliminate the need for human intervention, make WSNs a very interesting solution for information gathering applications, especially in harsh conditions where traditional approaches are very expensive or impossible. In this first section (2.1), which represents a brief introduction to wireless sensor networks, we will discuss the architecture, applications types, some areas of application, and some design constraints of these networks. In the next two sections (2.2 and 2.3), we give the operation, strengths, and weaknesses of the four main data processing techniques proposed in the literature for WSNs and present the concept of boundary traversal in WSNs which is the main ingredient of our proposed approaches.

2.1.1/ ARCHITECTURE

In a WSN, we distinguish two types of nodes: sensor nodes (data sources) and sink nodes (data destination). A typical sensor node is composed of four basic components [1, 2, 44]: (1) acquisition unit responsible for capturing physical quantities from the surrounding environment (temperature, humidity, pressure, vibrations, sound, image, etc.), (2) data processing and storage unit, (3) wireless communication unit and (4) energy unit. Depending on the application for which it was designed, a sensor node can also have additional units, such as a localization system and a unit responsible for moving the node. Sensor nodes are very limited in terms of resources, especially energy, so the main objective is to maximize their lifetime. As regards the sink, this node is usually much more powerful and has no energy constraints. Actually, in addition to its ordinary role of data gathering point, the sink has the ability to query information from sensor nodes.

2.1.2/ APPLICATIONS CLASSIFICATION

Based on how data is reported to the base station, WSN applications can fall into four categories: event-driven, time-driven, query-driven, or hybrid applications.

- In event-driven applications, sensor nodes send their data to the base station only if a specific event occurs (a threshold has been exceeded in sensor measurements). For instance, in a forest fire detection system, sensor nodes alarm the sink as soon as the temperature exceeds a certain threshold.
- In time-driven applications, sensor nodes *periodically* collect and send data to the base station. The acquisition period depends on the application and can vary from a few seconds to a few hours or even days. A good example of this class of applications is environmental data collection (agriculture, scientific experiments, etc.).
- In query-driven applications, sensor nodes send information only after an explicit request from the base station. The end-user can request information from certain specific regions (window queries) or from the whole network. In other words, in these applications, the network is seen as a distributed database that can be queried via the base station. Sensor nodes receive queries, execute them, and return the response to the base station.
- Hybrid applications combine any of the three modes described above. For example, a combination between an event-driven and time-driven applications.

2.1.3/ EXAMPLES OF APPLICATION AREAS

Among the areas where WSNs can be very useful are the military, security, domotics, environment, and health domains. In the following, we will succinctly present some examples of applications in these different areas.

- *Military applications*: the rapid deployment, self-organization and fault tolerance are features that make WSNs a very effective tool in the military field. More specifically, wireless sensors can be quickly deployed to help military units monitor strategic or hard-to-reach areas such as battlefields. They can provide information regarding the location, number, and movement of soldiers and vehicles. They can also detect chemical or biological agents.
- *Security applications*: deploying wireless motion sensors constitutes a very efficient distributed alarm system that can be used to detect intrusions into an area of interest. Since there is no critical point (single point of failure), disabling this system would not be easy.
- *Domotic applications*: wireless sensors can be used to monitor homes and contribute to their comfort by transforming them into intelligent environments whose parameters (temperature, humidity, brightness, etc.) can automatically adapt to the behavior of individuals.
- *Environment applications*: wireless sensor nodes allow, without disturbing, a better observation and tracking of wild animals life and movement. They can also efficiently detect natural disasters such as forest fires, storms, floods, volcanoes, etc.

For example, detecting a possible start of fire provides a faster and more effective extinguishing operation.

- *Health applications*: the use of wireless sensors can provide continuous patient monitoring. For instance, wireless sensors can detect abnormal behaviors (crying, falling, screaming, etc.) in the case of elderly or disabled people.

2.1.4/ MAIN DESIGN CONSTRAINTS

The design of WSNs face different constraints such as scalability, longevity, and robustness. The following points briefly describe each of these essential requirements.

- *Scalability*: the number of deployed sensor nodes can be of the order of hundreds or even thousands [1, 2]. Such a large number of sensor nodes generates a lot of transmissions and can create a lot of communication collisions. Thus, solutions and algorithms proposed for WSNs must be able to efficiently deal with any large number of nodes without overloading the network or exhausting its limited resources.

- *Longevity*: the definition of the lifetime of a WSN depends on its application. It can be defined as the duration until the first or last node dies. It can also be defined as the duration until a proportion of nodes die ($x\%$ of nodes has exhausted their batteries). For a WSN to remain alive for a long period of time without human intervention, energy consumption becomes a fundamental issue. Maximizing the life of sensor nodes means reducing their energy consumption.

- *Fault tolerance:* sensor nodes can fail due to physical damage (crushed by animals, during deployment, etc.), environmental interferences, or more frequently energy depletion. The failure of certain sensor nodes must not affect the functionality of the network. For instance, in critical hostile environments like battlefields, fault tolerance must be high because sensor nodes can be easily destroyed.

2.2/ QUERY PROCESSING IN WSNS

In this section, we present the principle of tree-based data aggregation, as well as some of the recent serial aggregation techniques proposed in the literature. In fact, in WSNs, information gathering from sensor nodes can be fulfilled using four different approaches; namely, parallel structure-based, parallel structure-free, serial structure-based, and serial structure-free. This section explains in detail the operation principles of each of these approaches, points out their strengths and weaknesses and provides examples of each type.

We mention that most of the approaches presented here in this section have been implemented and compared against our three proposed data aggregation approaches. In actual fact, the non-considered approaches have been mainly excluded because whether they do not ensure completeness (query accuracy) or because of their blatant inefficiency in terms of energy/time.

2.2.1/ PARALLEL STRUCTURE-BASED QUERYING

As their name clearly states, the operation of PSB approaches relies on structures (such as trees, clusters, etc.) that must have the sink as a root and must encompass all nodes in the network [3, 9, 10, 45–47]. As regards their mode of operation, these approaches are carried out in three distinct phases: *structure establishment*, *query diffusion*, and *data fusion*. Initially, the whole network must be covered by the desired structure; a spanning tree rooted at the sink for instance. Once this is done and the tree has been successfully built, queries can then be disseminated to all nodes of the tree ordering them to report the result of a certain processing that must be performed on their raw captured data. At last, once being interrogated, leaf nodes trigger data aggregation by simply forwarding their readings to their corresponding parents. From that point on, each non-leaf intermediate node waits to receive the result of each of its children nodes, fuses them with its own reading, and forwards the obtained result to its corresponding parent. This fusion operation is repeated until the root/sink is reached.

Besides the fact that PSB approaches are not scalable, and consequently not suitable for very large-scale dense deployments [18, 25], they suffer from several other limitations and drawbacks, among which we mention:

- **Structure construction/maintenance:** in these approaches, it is mandatory to build a structure that covers the whole network and provides each node with a path (next-hop) over which packets can be transmitted to the root. Constructing and repairing a distributed structure in a wireless collision-prone environment require a non-negligible time and energy. Several other issues need to be addressed. For instance, the tree must be balanced in terms of depth and in terms of node degrees. An unbalanced structure increases collisions and leads to an unbalanced energy consumption among nodes [48]. The energy depletion hole problem can also occur when nodes located near the root are overused to relay the traffic. In certain sensitive practical applications, the participation of all nodes is very crucial. Imagine a non-connected node with very sensitive data to report. Finally, in order to be reliable, the structure needs maintenance over time, which means more energy and time need to be spent.
- **Robustness:** PSB approaches are very vulnerable to topology changes and require maintenance or even complete re-construction in case of link/node failures. Actually, when a topology change occurs near the root, it leads to an important information loss, which can be very costly in terms of time and energy in large-scale networks.
- **Collisions and resources overuse:** given their concurrent nature, PSB approaches generate a lot of collisions during their three different phases, which considerably affects their performance in terms of energy conservation and delay time reduction, particularly in dense networks [18, 25]. In addition to collisions, operating in three separate phases also increases energy consumption and delays the response time. This becomes worse in very large deployments.
- **Query launcher singularity:** in these approaches, the root is the only point able to query the network. The creation of several trees over a resource-constrained network can quickly kill the latter. Besides, if the root changes its location whether intentionally or not, this can render the whole structure useless.

2.2.2/ PARALLEL STRUCTURE-FREE QUERYING

In these diffusion-based approaches, no routing is necessary [3, 9, 13–15]. In order to interrogate the network, the sink launches a query by broadcasting a packet to its one-hop neighbors. Upon receiving the query for the first time, each node rebroadcasts it to its immediate neighbors. After query spreading, each node exchanges raw data with its immediate one-hop neighbors. Actually, this last step depends on the used approach. In flooding approaches, each node exchanges data with all nodes in the network (via its immediate neighbors). In consensus-based approaches, each node keeps exchanging data in an iterative way with its one-hop neighbors until convergence (i.e., until reaching the desired result). In both approaches, in the end, each node in the network will acquire the answer of the query.

Due to their non-reliance on any routing structure and their use of one-hop diffusion among nodes, PSF approaches are very robust to topology changes [16, 17]. Nonetheless, these approaches are not suitable for WSNs because they excessively use the network resources and require a significant execution time [18, 25]. In parallel approaches, nodes perform their assigned tasks concurrently. So, from a theoretical perspective, this can be seen as an asset because it can presumably enhance the response time. Nevertheless, as shown by recent research [18, 25], even when a sophisticated MAC protocol is utilized [49], parallel information gathering creates a lot of collisions, which considerably dissipates energy and wastes time, especially in large-dense networks.

2.2.3/ SERIAL STRUCTURE-BASED QUERYING

As is the case for PSB approaches, using a predefined path contributes to the vulnerability of SSB approaches. For instance, if at some point the next designated hop is unavailable, the traversal process will inevitably stop at the currently explored node. In the following, we will briefly describe two examples of these approaches, namely, Space-Filling Curve-based and Depth-First Searches.

- **Depth-First Search (DFS)** [26]: the distributed version of this straightforward well-known approach expands the path, as far as possible, towards the unvisited nodes, and when it gets stuck at a certain hop (i.e., when all neighbors of the currently traversed node have been visited), it backtracks to the parent of that node and so forth. In fact, this basic technique constructs a path for each launched query and requires $2 * (n-1)$ communications to interrogate a network of n nodes and report the answer to the QL/sink. Among the main drawbacks of this technique, we cite the very poor robustness in front of node/link failures during the backtracking process.

- **Space-Filling Curve-based Search** [24]: this serial data aggregation technique, as its name implies, uses space-filling curves to lay out an itinerary through the network. When compared with DFS, this approach performs better in dense regular topologies and can traverse a sensor network more efficiently in terms of communications [24]. Nonetheless, besides its poor scalability and its weak robustness against topology changes (link and node failures, ...), it cannot handle irregular network topologies with communication holes. An example in which space-filling curve-based approach fails to ensure full network exploration and thus fails to aggregate all the data present in the network can be found in [25].

We mention that this approach was not considered (implemented) in the conducted simulations because it does not ensure aggregation completeness.

2.2.4/ SERIAL STRUCTURE-FREE QUERYING

In SSF approaches, a path passing through every node in the network must be gradually built, and each traversed node must be able of autonomously choosing the next hop. More precisely, the query launcher sends its reading to the next autonomously determined hop. The latter, first, fuses the received data with its own reading, second, autonomously determines the next hop, and finally sends the result to that node and so on. In the end, after complete network traversal, the result owned by the last node in the path is sent to the query launcher via an independent geographic routing.

SSF approaches have proven their effectiveness and shown interesting results in terms of avoiding collisions, reducing communications, and saving energy and time. Actually, recent research has confirmed the outperformance of SSF approaches in large-deployments and demonstrated that latency is not an intrinsic drawback as intuitively expected. The following points summarize the major advantages of SSF approaches:

- **Completeness:** SSF approaches can be proven to ensure the traversal of any possible topology (e.g., irregular hole-topologies, regular hole-free topologies, etc.). This feature makes SSF approaches very suitable for many practical sensitive applications in which the expected result must involve all nodes of the network.
- **Localizability**: SSF approaches are highly scalable and can support a very large number of nodes because they have a localized distributed nature. As previously mentioned, in these approaches, the next hop determination is done only through the use of the one-hop neighbors' table of the currently traversed node.
- **Structure-independency:** as indicated by their name, SSF approaches do not rely on any predefined itinerary; each time a query is launched, a new path is drawn. Creating the path on the fly gives SSF approaches huge advantages over the other approaches. First, it allows them to rapidly query networks. Second, it makes them maintenance-free, more robust against links/nodes failures and other topology changes, and thus very suitable for large dense WSNs and their specificities. Finally, queries can be issued from different spots and not just from the root/sink as is the case for other approaches. Actually, any node whatever its location can interrogate the network without wasting time and creating/fixing any structure. This characteristic is essential in multi-owner-multi-user WSNs in which many nodes can be in charge of distributing management commands and/or updating configuration parameters [50].
- **Compactness:** in serial approaches, queries are disseminated and processed at the same time. This merge significantly reduces the amount of sent packets, conserves energy and enhances query responsiveness. If we assume the existence of a path that passes exactly once by each node, then $n - 1$ packets are sufficient to query a network of n nodes and collect their answers.
- **Collision-freeness:** given the fact that only one node can transmit at any given instant in time, SSF approaches are collision-free. The total absence of collisions

facilitates communications, saves energy and improves query responsiveness. Actually, since SSF approaches do not witness any collisions, no sophisticated MAC layer is required in these approaches. This interesting feature can further improve the network traversal time.

In the remaining of this section, we will briefly present one of the recent SSF approaches proposed in the literature, namely *Greedy and Boundary Traversal* (GBT) [25]. This serial aggregation approach operates in two alternating phases: *greedy forwarding* and *boundary traversal*. In the beginning, during the greedy forwarding phase, the path is extended, as much as possible, towards the unvisited nodes. The idea here consists of starting from the sink and, at each step, adding the nearest unvisited node to the sink/QL to the path. When there are no more unvisited neighbors left at some hop (i.e., when the currently traversed node has no unvisited one-hop neighbors), the second boundary traversal phase begins looking for other possible unvisited nodes in the network. If a non-visited node has been found, the alternative greedy traversal phase will be resumed. This way, GBT switches between the two phases until visiting all nodes. In the end, after browsing the entire network, the boundary traversal phase will produce a cycle, indicating thus the end of the information gathering process.

2.3/ BOUNDARY TRAVERSAL IN WSNS

Since the key idea and essential ingredient of our serial data aggregation approaches is boundary traversal, the aim of this section is to provide an overview of this concept. More specifically, in this section, we explain the principle of operation of the boundary traversal algorithms used throughout the manuscript. In addition to this, we also cover the basic concepts of communication holes, boundaries, and boundary nodes in WSNs.

2.3.1/ BASIC CONCEPTS

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. 2.1 gives an illustrative example of holes and boundaries in a wireless network. For instance, the orange regions in the network deployment of Fig. 2.1 represent four holes. Note that the boundary of the network and that of holes 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.

We define a hole within a wireless network, as follows:

Definition 1: Hole

In a wireless connected network, 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. 2.1, hole 1 is delimited by the following non-intersecting links: $L_{(1,2)}$, $L_{(2,3)}$, $L_{(3,4)}$, and $L_{(4,1)}$.

According to the localization of nodes inside the network, we can define three categories of nodes: (a) boundary nodes, (b) network boundary nodes and (c) internal non-boundary nodes. In order to identify the boundary nodes in a wireless network, several definitions

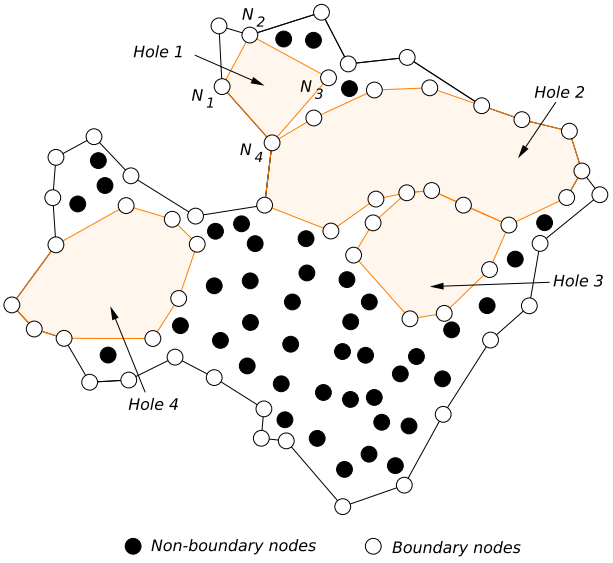


Figure 2.1: Example of holes and boundaries in a wireless sensor network.

have been proposed in the literature [18, 38, 41]. We underline that our three proposed serial approaches utilize different definitions of boundary nodes. To ease the manuscript reading, the details and definition utilized by each of our approaches will be provided in its corresponding chapter.

It is worth mentioning that the identification of network boundary nodes is not a process that can be done locally by each node as is the case for the identification of boundary nodes (where each node can rely only on its one-hop neighbors to perform this task).

2.3.2/ BOUNDARY TRAVERSAL ALGORITHMS

The main role of a boundary traversal algorithm is to sequentially visit all nodes of a boundary, one by one. In fact, originally, in WSNs, boundary traversal has primarily been studied in the context of data routing around communication holes and has been proposed as a solution for this void problem encountered in geographic routing [41]. For instance, as demonstrated in Fig. 2.2, in order to reach the destination N_d, node N_1 which is a local minimum[1] creates a virtual disc (rolling-ball) and spins it counterclockwise. The first touched neighbor, i.e., node N_2, in turn, spins the received disc and determines the next hop. This process is repeated at each visited node until the greedy routing is resumed or until the whole boundary is traversed.

According to the requirements set beforehand (Section 1.2), the boundary traversal solution to be utilized in our approaches must adhere to the following criteria. First, in order to ensure a complete network search, the considered boundary traversal algorithm must not skip any boundary node in its path, and must also maintain the traversal process on the same boundary. Second, in order to achieve time and energy efficiency, the used boundary traversal technique must be *localized* (i.e., does not require any additional knowledge except the pre-collected one-hop information).

To the best of our knowledge, in the literature, two major categories of boundary traver-

[1] N_1 is the nearest node to the destination N_d (among its one-hop neighbors).

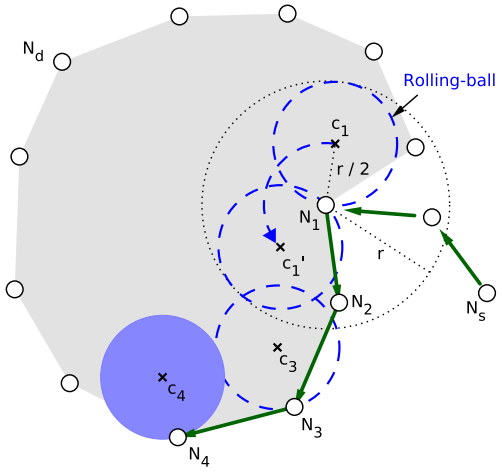


Figure 2.2: Anti-void routing.

sal algorithms have been proposed: (a) *graph-based approaches* such as GPSR [51], GOAFR [52], etc., and (b) *graph-free approaches* such as GAR which employs a Rolling-Ball [41], and CS which uses a Curved-Stick [38]. In order to traverse boundaries and avoid holes, graph-based approaches, as their name suggests, require the construction and maintenance of the underlying planar graph, and hence they require an extra overhead. To be precise, graph-based boundary traversal approaches require the construction of a planar graph that represents the same connectivity as the network graph (i.e., without altering the network connectivity, all crossing links must be removed). Actually, whilst graph-based approaches generate an important overhead due to the construction/maintenance of planar graphs, graph-free approaches are of a *localized memoryless*[2] nature (which fulfills our scalability requirement specified in Section 1.2). Furthermore, graph-free approaches have been *theoretically proven* to ensure boundary traversal whatever the boundary configuration [37, 38, 41] (which satisfies our completeness requirement specified in Section 1.2).

Based on what has been said, in this thesis, we totally disregard the first category of algorithms (obviously because of their graph construction/maintenance overhead) and adopt graph-free techniques (to implement our approaches) because they are much more suitable for our case; they fulfill our requirements, and serve our objective of (1) proposing a localized approach and (2) ensuring network traversal completeness. In other words, what attracts us to graph-free approaches is not just their localized nature but also the fact that they have been theoretically proven to ensure boundary traversal, which allows us to guarantee the exploration of all nodes.

To the extent of our knowledge, in the literature, only two distributed localized graph-free boundary traversal approaches have been proposed, namely the curved-stick (CS) [38] and the rolling-ball (GAR) [41]. In the following, we describe these two well-known algorithms.

[2]They do not rely on any local storage (any additional information) except the one-hop neighbors tables of the concerned nodes.

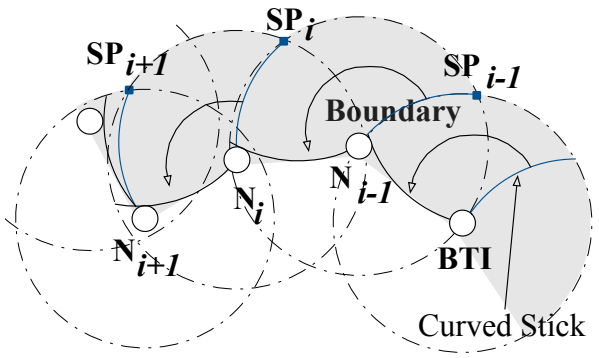


Figure 2.3: Curved-stick boundary traversal started by BTI.

2.3.2.1/ CURVED-STICK

The authors in [38] proposed a *curved stick boundary traversal algorithm* that is (as its name indicates) based on a *curved-stick* that can be swept clockwise or counterclockwise. More specifically, in this graph-free approach, the process of boundary traversal must always start from a boundary node, called BTI (*Boundary Traversal Initiator*), which is responsible for setting (initiating) the curved-stick (Fig. 2.3). Initially, as shown in Fig. 2.3, the curved stick (arc with radius R) is hinged at the BTI inside the boundary to be crossed. From this position, the BTI sweeps the curved stick counterclockwise until a neighbor is hit (node N_{i-1}). The latter which is selected as next hop (and hence will receive the traversal packet), in turn, must compute the new starting point of the curved-stick (point SP_{i-1}), sweep it to determine its next hop, and so forth. The same process is repeated by each traversed node until visiting the whole targeted boundary.

Note that each traversed node must start sweeping the curved-stick from its corresponding *starting point* defined as follows:

Definition 2: Curved stick starting point

The starting point SP_i of a node N_i is the intersection point between the two circles, of radius R, centered at N_i and N_{i-1} (previous hop of N_i), such that SP_i is located in the left side of the oriented line $\rightarrow N_iN_{i-1}$.

This localized boundary traversal technique has been shown to be very efficient in terms of saving energy and time, and deriving shorter routing paths around communication holes [37, 38]. More details about the curved-stick, in particular, the ones related to its proof of correctness, can be found in [37, 38].

2.3.2.2/ ROLLING-BALL

With regard to GAR [41], its operation is identical to that of CS [38] except a rolling-ball is used instead of the curved-stick. More precisely, this approach uses a virtual void[3] circle (of $R/2$ radius) that is hinged at a node (owned by one node) and must be empty of any other node. Fig. 2.2 shows a rolling-ball hinged at node N_1 with $c_1 \in \mathbb{R}^2$ as its center and $R/2$ as its radius (where R is the communication range of nodes [27, 28]). As depicted in

[3] It does not contain any node, neither initially, nor at any given time.

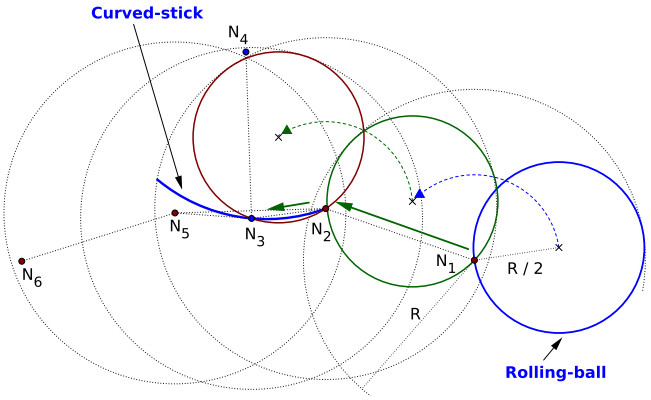


Figure 2.4: Rolling-ball boundary traversal triggered by node N_1.

Fig. 2.4, in order to traverse a given boundary, this circle, called *rolling-ball* and hinged at a boundary node (node N_1), is spun counterclockwise. The first hit neighbor (node N_2), which is chosen as next hop, repeats the same operation and so on. All nodes hit by either the curved-stick or rolling-ball are identified as *boundary nodes*. We also note that for both the curved-stick and rolling-ball, the clockwise direction can be used.

The rolling-ball radius value of $R/2$ is justified by the fact that when the current boundary node, holding the rolling-ball, spins the latter, it must be ensured that it hits neighbors. If the rolling-ball, hinged at the current node, does not touch any node, this necessarily means that the current node is isolated and does not have neighbors. This property has been used to formally demonstrate that GAR ensures boundary traversal. For the rest of this manuscript, we will use the term "*optimal*" when referring to this value of the rolling-ball radius (i.e., $R/2$). In the same context, we underline that the ball's emptiness is an essential requirement for a valid boundary traversal. In other terms, in order to guarantee a correct boundary detection/traversal, the rolling-ball must be, initially and all the time, empty of active nodes.

Finally, we recall that the operation of our proposed serial approaches has been built on *boundary traversal* in order to ensure their completeness (i.e., ensuring the traversal of all nodes in the network). Furthermore, we underline that in order to achieve a good performance, in particular, scalability, the distributed localized nature of the utilized boundary traversal algorithm must be maintained throughout the network exploration process. In the next chapter, we will start by presenting our first querying approach called the Peeling Algorithm.

3

PEELING ALGORITHM

3.1/ INTRODUCTION

In WSNs, serial data aggregation approaches, in which a parameter of interest is estimated through a serial set of communications between nodes, have shown their effectiveness over both, parallel structure-based and parallel structure-free approaches. Nevertheless, they still suffer from two major drawbacks: (i) they require the construction of a path that must cross every node in the network exactly once, which is known to be an NP-Complete problem [23] and (ii) they experience a poor scalability, which is an important concern in large-scale WSNs. In this chapter, we tackle these issues by proposing a novel serial localized approach, called Peeling Algorithm (PA) [18]. In this approach, a packet travels serially from node to node carrying with it the parameter estimate, and each visited node can locally determine the next hop of the packet while not needing to store any information about the network topology. This unique feature allows a very good scalability of PA. In this chapter, we also present a second algorithm, called Enhanced Peeling Algorithm (EPA) [18]. More specifically, we will discuss the implementation details of PA and EPA, provide their proofs of correctness and report their performance evaluation results. Actually, the extensive OMNeT++ simulation experiments, we have conducted, indicate clearly that our proposed algorithms outperform the existing ones.

Briefly, this chapter is structured in six sections, as follows. The next section (3.2) introduces the preliminaries and background required throughout the chapter. Section 3.3 presents and details the operation of the proposed localized approach through illustrative examples, while Section 3.4 provides its proof of correctness. In Section 3.5, we present and discuss the EPA approach. Section 3.6 presents and comments on the performance evaluation results of both PA and EPA. Finally, Section 3.7 concludes the chapter.

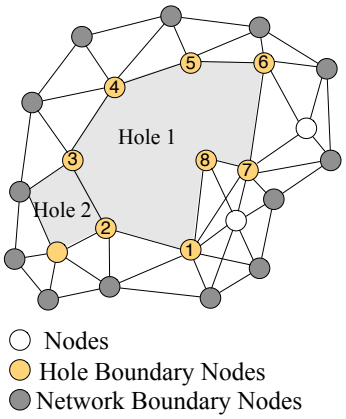


Figure 3.1: Boundary and non-boundary nodes in a wireless network.

3.2/ BACKGROUND AND GENERAL IDEA

This section presents the essential ingredients required for the proper operation/understanding of our peeling approach PA, namely the adopted definitions of boundary nodes and network boundary nodes. This section also succinctly presents the utilized boundary traversal algorithm (namely the curved-stick), and the general idea of the proposed peeling algorithm.

The localized distributed nature of PA imposes that each node, upon receiving the query, must execute a defined set of rules that are selected according to the localization of nodes inside the network. As previously mentioned in Section 2.3.1 (boundary traversal basic concepts), in a wireless network, three categories of nodes can be distinguished: (a) boundary nodes, (b) network boundary nodes and (c) the rest of nodes in the network, i.e., non-boundary nodes (Fig. 3.1). Before defining each category of nodes, let us first recall what is a hole in the world of wireless networks. Actually, a hole is an empty closed region delimited by the non-intersecting links of at least four nodes (see Section 2.3.1 and Definition 1 for more details). For example, the grey regions in the network of Fig. 3.1 represent two holes. The boundary of Hole 1 is formed by nodes N_1, N_2, N_3, N_4, N_5, N_6, N_7, and N_8.

In the same context, we recall that in order to identify boundary nodes, several definitions can be applied [18, 38, 41]. The definition that we propose in PA is the following:

Definition 3: Boundary Node (BN)

A node N is said to be a boundary node in the direction $\angle N_i N N_j$, iff it has at least two angularly adjacent neighbors N_i and N_j, so that N_i cannot communicate with N_j or the angle $\angle N_i N N_j \geq \pi$.

We mention that we assume that all one-hop neighbors ($\mathcal{V}_i$ set) of a node N_i are ordered counterclockwise. Hence, two consecutive one-hop neighbors of N_i are said to be *angularly adjacent*. For instance, in Fig. 3.1, node N_1 has two angularly adjacent neighbors; N_8 and N_2, that cannot communicate, so N_1 is facing a hole in this direction. Similarly, the two angularly adjacent neighbors of node N_8; N_7 and N_1 are forming an angle greater than π, hence node N_8 is facing a hole in this direction, and this, despite the fact that these two neighbors can communicate. We note that all (grey and yellow) shaded nodes

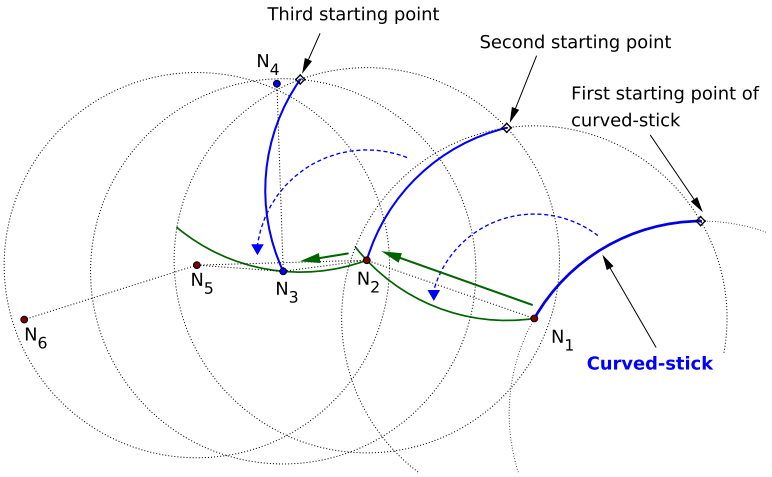


Figure 3.2: Curved-stick network traversal started by node N_1.

in Fig. 3.1 are boundary nodes.

Definition 4: Network Boundary Node (NBN)

In a wireless connected network, the network boundary nodes are defined as the set of BNs such that the polygon formed from the corresponding set of links, considered as straight non-intersecting line segments, contains all nodes of the network.

For example, the NBN set of Fig. 3.1 is composed of all the dark shaded nodes. In our proposed approach (detailed in Section 3.3), the serial visiting (i.e., peeling) process must begin from a NBN node. However, the identification of such a node is not straightforward and cannot be done locally by each node as is the case for boundary nodes. In Section 3.3.3, we present the distributed localized algorithm that we propose to determine whether a node is NBN or not. Once the starting point of the peeling has been determined, the network can then be peeled layer-by-layer starting from that point (Section 3.3).

We mention that as a network traversal tool, PA makes use of the boundary traversal algorithm described in Section 2.3.2.1. Actually, as shown in Fig. 3.2, this localized distributed algorithm [38] utilizes a *curved-stick* that hinged at a node (N_1) and swept counterclockwise until a neighbor is hit (N_2). The latter, which is selected as next hop, in turn, computes the new starting point of the curved-stick, sweeps it to determine its next hop, and so forth. A more detailed description of the curved stick can be found in Section 2.3.2.

3.3/ PROPOSED APPROACH: PEELING ALGORITHM

In this section, we present our proposed serial approach, named Peeling Algorithm (PA). We first begin by sketching the overall peeling behavior through illustrative examples in different network topologies. We then provide details of the mechanism utilized to select the peeling starting point/node.

The proposed approach operates in three steps, as follows:

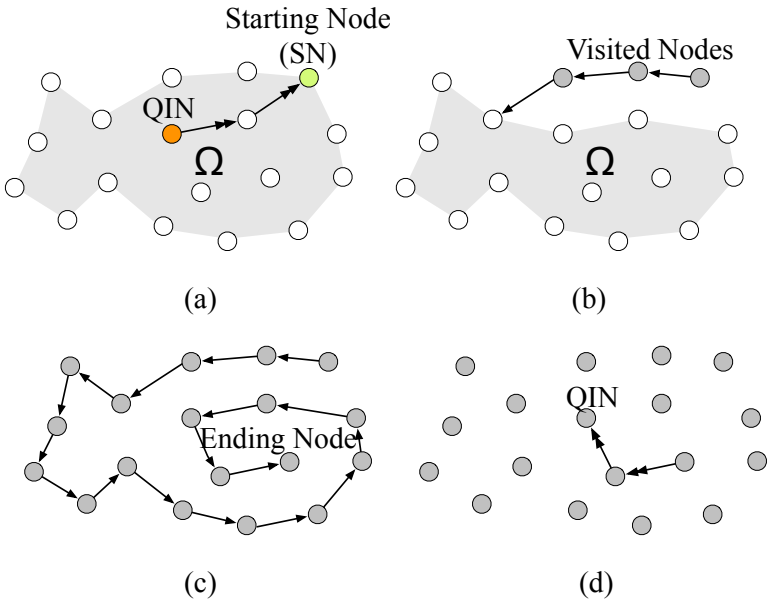


Figure 3.3: Peeling process.

- First, the *Query Launcher* (QL), called also the *Query Initiator Node* (QIN) sends the query packet to a node, named *Starting Node* (SN), which is located on the external boundary of the network (Fig. 3.3 (a)). The determination of the starting node of the peeling will be described in details in Section 3.3.3.
- Second, from this node (i.e., SN), the boundary traversal of the unvisited nodes starts, leading at each step to visit a node located on the external boundary of the network, which is changing over time (shaded region in Fig. 3.3 (b)). The process stops when the immediate one-hop neighbors of the currently traversed node have all been visited. (Fig. 3.3 (c)).
- Finally, by means of any efficient geographic routing mechanism [1], the query result is sent from the last node in the path to the QL ((Fig. 3.3 (d)).

The key idea of the peeling approach is to serially visit the network, node by node, using the curved-stick [38]. The latter guarantees that whenever the traversal process starts from a boundary node, it remains on the same boundary.

In order to find the next appropriate hop, every node must maintain a flag for each of its neighbors. This flag indicates whether the corresponding neighbor has been visited by the query process or not. Thus, each node, upon receiving the query, will know which of its neighbors have been visited, and hence will not consider them when it selects the query next hop.

We note that thanks to the broadcasting nature of wireless communications, the process of updating the flags does not require any communications or overhead other than the query packet itself. In fact, in this communication model of WSNs [1], when a node sends a packet, all nodes within its transmission range will hear it and hence can receive it. This interesting feature is used in our approach to update, without any additional cost, the visiting flags of each node.

The initial boundary from which the proposed peeling algorithm starts is the boundary of

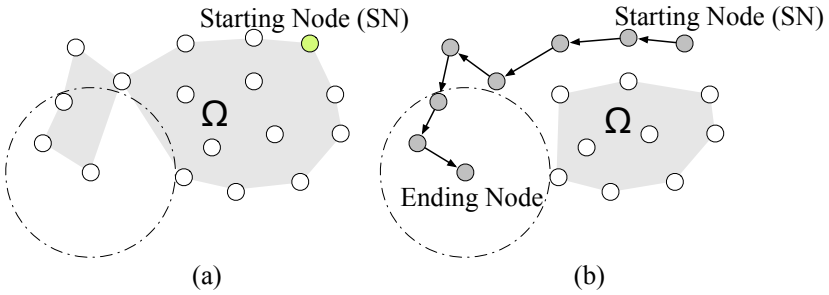


Figure 3.4: Peeling disconnectivity issue. (a) The node in the middle ensures the connectivity of Ω. (b) If this node marks itself as visited (i.e., no longer participates in the traversal process), Ω will be partitioned, leading to missed unvisited nodes.

the unvisited sub-network, defined as follows:

Definition 5: Unvisited sub-network

We define the current unvisited sub-network, denoted by Ω, as the set of all currently unvisited nodes.

3.3.1/ HOLE-FREE TOPOLOGIES

Initially, we have $\Omega = \mathcal{N}$ and the starting node SN is located on the external boundary of Ω. Then, at each step of the process, the currently traversed node marks itself as *visited* (i.e., removed from Ω), and in turn selects the next unvisited neighbor located on the boundary of Ω. Consequently, the external boundary of Ω changes in each step of the peeling process[1]. This rule is reapplied by each node receiving the query until visiting all nodes in the network.

We underline that although the clockwise direction can be used, in this chapter, as illustrated in Fig. 3.3, we adopt only the counterclockwise (trigonometric) direction of the peeling.

The network traversal process described above, whereas it behaves very efficiently in dense and hole-free topologies, does not nevertheless ensure query completeness for any network topology. A simple example in which this process fails is when a node that connects two parts of the network marks itself as visited (Fig. 3.4 (a)). Actually, once this node is being marked as visited, the unvisited region connectivity will no longer be ensured, leading thus to missed unvisited nodes as shown in Fig. 3.4 (b).

To prevent such a scenario from happening and preserve the connectivity of Ω, certain nodes have to be maintained *alive* (involved in the traversal) despite the fact that they have been visited. To this end, the notion of *bridge nodes* has been introduced.

[1]The term "*peeling*" comes from the fact that, at each step, the visited node is removed from the external boundary of Ω.

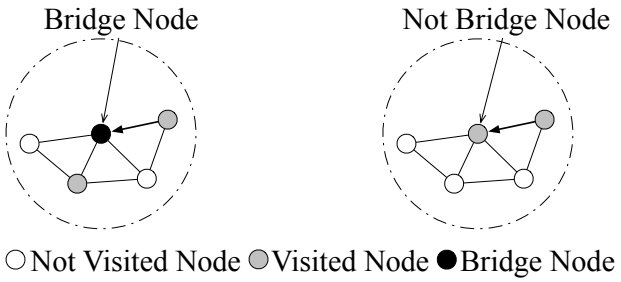


Figure 3.5: Bridge and non-bridge nodes.

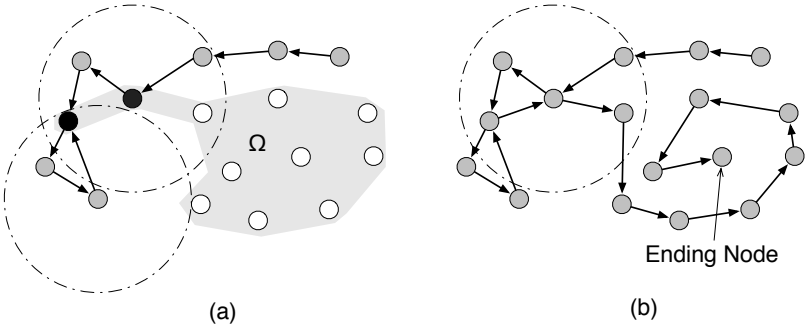


Figure 3.6: Peeling connectivity maintenance. (a) Bridge nodes maintain the connectivity of Ω. (b) The process continues, at each moment, on the boundary of Ω.

Definition 6: BRidge Node (BRN)

A node is called bridge node when it satisfies the two following conditions:
1/ *At least two of its unvisited or BRN neighbors cannot communicate, and*
2/ *None of the other unvisited or BRN neighbors are able to connect these two neighbors.*

In other words, a bridge node is the only node able to ensure the connectivity of at least two of its neighbors. An illustration of bridge and non-bridge nodes is given in Fig. 3.5. We note that according to the considered network model presented in Section 1.3, each node can locally[2] decide whether or not it is a BRN. Nevertheless, when the connectivity of two nodes does not depend only on the distance between them (e.g., the presence of obstacles), the node in question has to further be aware of the effective communication links between its immediate neighbors.

To maintain the connectivity of Ω and consequently ensure query completeness/accuracy, bridge nodes must not be removed from Ω and must be considered by the currently traversed node when selecting the next hop. This means that the next hop of the currently traversed node can be either an unvisited or BRN node. Fig. 3.6 shows how the peeling algorithm behaves in the presence of BRN nodes. Actually, when a node detects locally that it is a BRN (black nodes in Fig. 3.6 (a)), it continues the process and will not be removed from Ω (it ensures its connectivity). When a BRN node receives the peeling packet once again, it changes its status according to the ones of its immediate neighbors and lets the process continues on the boundary of Ω (Fig. 3.6 (b)).

Property [1]. *Bridge nodes ensure the connectivity of the unvisited sub-network Ω.*

[2]based only on its neighboring information.

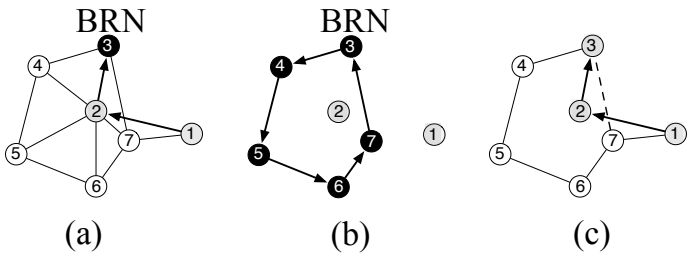


Figure 3.7: Example of artificial holes. (a) Once node N_2 is marked as visited, it creates an artificial hole. (b) Artificial holes lead to looping. (c) Links crossing the boundary and causing artificial holes (dashed line) must be deactivated.

Proof. We prove this property by contradiction. We assume that removing the currently traversed node N_i from Ω breaks its connectivity, and we assume that node N_i is not a BRN. Initially, thanks to the network connectivity assumption made in Section 1.3, we know that Ω is connected. According to Definition 6, being not a BRN means that among each pair of neighbors of node N_i, there exists a link connecting them without passing by N_i. Consequently, Ω is connected, which contradicts the fact that removing N_i breaks the connectivity of Ω. □

Although bridge nodes resolve the disconnectivity problem, they, however, bring another problem to the surface that we call the *artificial holes* problem. Fig. 3.7 (a) illustrates the formation of such holes. In fact, when node N_2 receives the peeling packet from node N_1, it considers itself as visited, because all its unvisited neighbors (N_3, N_4, N_5, N_6, and N_7) can communicate with each other without its help. As a matter of fact, removing node N_2 from Ω will create an artificial hole between the unvisited nodes N_3, N_4, N_5, N_6, and N_7. Afterward, when node N_3 receives the peeling packet, it considers itself as a BRN because its two unvisited neighbors, i.e., N_7 and N_4, cannot communicate with each other. From that point on, according to Definition 6, all the other nodes will be considered as BRNs, leading hence to a looping situation (Fig. 3.7 (b).)

The creation of artificial holes is due to the conjunction of two facts. First, node N_2 is not a boundary node in the sense that all its unvisited neighbors can communicate with each other. Removing node N_2 (i.e., marking it as visited) will create a hole between its unvisited neighbors. Second, some links, which are crossing the boundary of Ω, are still considered, in particular, within the BRN decision. For instance, in our example, link $L_{(7,3)}$ is crossing the boundary of Ω when the peeling packet is sent from N_1 to N_2 to N_3. When node N_3 receives the packet, it takes into account link $L_{(7,3)}$, which is actually behind its decision of being a BRN.

To alleviate this problem, links which are behind the creation of artificial holes have to be deactivated. Before diving into the details of how to identify these links, we first introduce the notion of *potential boundary nodes*.

Definition 7: Potential Boundary Node (PBN)

A node N_i is said to be a potential boundary node iff when considering only its alive neighbors (i.e., unvisited or BRNs), it is a boundary node per Definition 3.

In this definition, the goal is to capture whether or not the currently traversed node is located on the actual external boundary of Ω. To this end, the currently traversed node

has to consider only its alive neighbors and see, according to Definition 3, whether it is located on a boundary (i.e., external boundary of Ω). Consequently, if a node is a PBN, it cannot create an artificial hole. On the opposite case, i.e., when a node is not a PBN, there is a risk of artificial hole formation.

Moreover, some links may cross the current boundary of Ω, as is the case of link $L_{(7,3)}$ in Fig. 3.7 (a). To handle this situation, we introduce the following definition:

Definition 8: Alive Neighborhood

We denote by $V_i(\Omega) = \{N_j \mid N_j \in V_i \wedge N_j \in \Omega\}$ *the current alive neighborhood of node* N_i.

Based on $V_i(\Omega)$, the currently traversed node N_i can determine which links among $V_i(\Omega)$'s links are crossing the boundary of Ω. For instance, in our example of Fig. 3.7, node N_2 can easily determine that link $L_{(7,3)}$ is crossing the current boundary of Ω. Hence, when a node N_i receives the peeling packet from the previous hop N_{prev}, and find out that it is not a PBN and that there exist some links crossing the current boundary of Ω, it executes the two following steps:

- Construct two sets. The first one contains all nodes involved in the crossing links that are on the right side of the segment $[N_i,\ N_{\text{prev}}]$. The second set is formed by the other nodes. In our example, the two sets are: $\{N_7\}$ and $\{N_3\}$.
- Send the peeling packet with this additional information about the two sets.

Upon receiving the peeling packet, the neighbors of the currently traversed node must perform the following step. Each node checks first to which set it belongs. If it belongs to one of these two sets, then, it deactivates (deletes) its links with the nodes belonging to the other set. For example, as shown in Figure 3.7 (c), node N_3 will deactivate its link with N_7, and vice versa. Consequently, node N_3 will not identify itself as a BRN, avoiding thus the previously observed looping situation. We underline that this link deactivation concerns only the current executed query.

Property 2. *PBN rules do not raise artificial holes.*

Proof. Let us suppose that the currently traversed node is N_i, the previous hop is N_{prev}, and that an artificial hole has been created. We note that this assumption implies that N_{prev} has necessarily been marked as visited, otherwise an artificial hole cannot be created. In more details, we have two cases to consider; N_{prev} is a PBN or a non-PBN.

- N_{prev} is a PBN: according to Definition 7, this implies that nodes in $V_{N_{\text{prev}}}(\Omega)$ do not form a closed region, which contradicts the assumption of existence of a hole (cf. Definition 1).
- N_{prev} is not a PBN: also according to Definition 7, this implies the existence of a path connecting all alive neighbors of N_{prev} without passing through it. However, N_{prev} was located on the boundary of Ω and consequently, there exists a link belonging to the created hole and crossing this boundary. This is in contradiction with PBN rules because such a link would have necessarily been deactivated. Consequently, a hole cannot exist, which is in contradiction with the initial assumption.

Therefore, in all cases, an artificial hole cannot exist. □

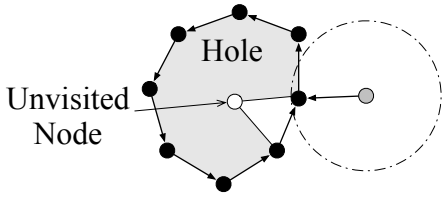


Figure 3.8: Example of peeling misbehavior (caused due to the use of bridge nodes concept).

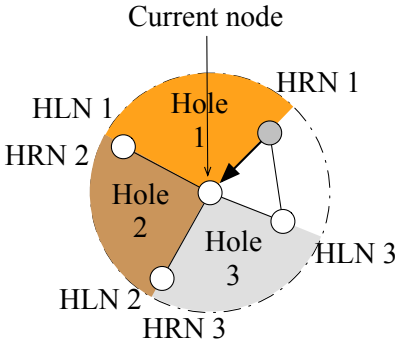


Figure 3.9: Example of hole selection by HGN.

3.3.2/ HOLE TOPOLOGIES

Network topologies with holes are usually much more complex to handle with distributed and localized algorithms. If we apply the peeling process described above on network topologies with holes, then, as depicted in the example of Fig. 3.8, it will fall into a hole looping situation and, consequently, will not be able to visit nor the potential nodes located inside that hole nor the rest of nodes in the network. For instance, in the given example, the node located inside the hole cannot be accessed/traversed because bridge nodes (black nodes) are creating a looping situation around the hole and preventing thus the peeling process from going inside. Essentially, the problem comes from Definition 6, because according to this latter, all nodes, which are located on the boundary of a hole, will be marked as BRNs.

To overcome this looping issue and render the peeling process able to visit all nodes, the nodes responsible for such a misbehavior have to be identified, and the rules associated with these nodes have to be defined. Of course, the connectivity of Ω must be maintained all the time.

Definition 9: Potential Hole Boundary Node (PHBN)

Every Boundary Node (BN) which is not a Network Boundary node (NBN) is defined to be a potential hole boundary node.

In other words, as previously explained, each PHBN is facing a hole that can create a looping case. We mention that, as shown in Fig. 3.9, each PHBN node has two hole neighbors: *Hole Right Neighbor* (HRN) and *Hole Left Neighbor* (HLN).

To avoid looping and ensure that all nodes in the network will be visited, the idea is to break holes (cycles) and allow the peeling process to go inside. We emphasize that the peeling must also be able to leave holes (avoid inner looping). These goals are not

obvious to meet, especially with a localized serial algorithm. To solve this issue, we introduce the following definition:

Definition 10: Hole Gate Node (HGN)

We define the Hole Gate Node of a hole as the first PHBN node of this hole to receive/possess the peeling packet.

The HGN notion has a twofold role: (a) it allows the peeling process to discover holes (and visit any potential inner nodes), and (b) it guarantees that the peeling can leave holes. To meet these two requirements, while still using a serial boundary traversal algorithm, some particular links in Ω have to be deactivated. In other terms, the boundary of Ω must be modified without altering its connectivity.

3.3.2.1/ HOLE GATE NODES IDENTIFICATION

According to Definition 10, two steps are necessary for a node to identify itself as HGN:

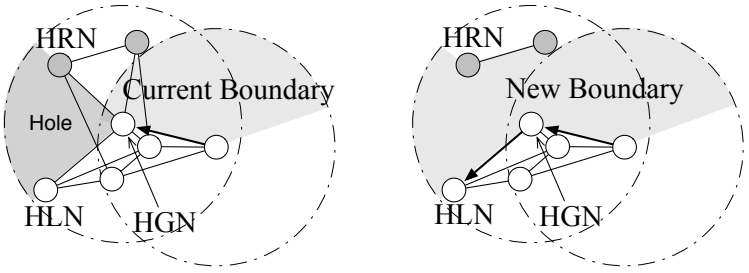
- First, it must determine whether it is facing a hole or not (Definition 3). For instance, the node of Fig. 3.9 is facing three holes. We underline that each node can locally perform this step (thanks to its neighboring information).
- Second, once this node is aware that it is located on the boundary of a hole, it can determine if it is the HGN of this hole when it receives the peeling packet. More precisely, if the peeling packet arrives at the currently traversed node from a node that does not belong to the same hole, the currently traversed node is certainly the first visited node of the hole and consequently, it is the HGN of this latter. Otherwise, the hole has already got an HGN. We note that this second step can be performed locally and does not require any extra-information other than the one-hop information.

In the case where the currently traversed node is facing more than one hole, the selection of the hole to be explored is made according to the counterclockwise direction. For instance, in Fig. 3.9, the peeling packet came to the currently traversed node from hole 1, which means that hole 1 has already got an HGN and consequently, hole 2 will be the one to be explored next.

3.3.2.2/ HOLE GATE NODES RULES

Upon receiving the peeling packet from the previous hop N_{prev}, if node N_i detects that it is an HGN, it executes the four following steps:

1. Creates two sets from its immediate neighbors.
   - The first set, called Front Set (FS), contains all neighbors located in the sector defined by the angle $\angle N_{\text{prev}} HGNHLN$. In the example of Fig. 3.10, all grey nodes belong to this set, including the HRN.



(a) (b)

Figure 3.10: Rules applied by HGNs. (a) Before performing HGN rules. (b) After changing the boundary of Ω.

- The second set called Back Set (BS) is composed of the rest of neighbors, including the HGN itself and its HLN. In Fig. 3.10, all white nodes belong to this set.

2. Virtually deletes links with every node of FS. This means that, for the currently executed query, the HGN will not consider these nodes as its neighbors. Formally speaking, the set $\mathcal{D} = \{L_{(\text{HGN, i})} \mid N_i \in FS(HGN)\}$ is removed from $\mathcal{L}$ for this query.

3. Initializes a packet which contains: (1) its identity as the HGN of this hole, and (2) the two previously constructed sets $FS(HGN)$ and $BS(HGN)$.

4. Broadcasts this packet to neighbors.

Upon receiving the HGN packet, each neighbor must perform which follows:

1. Determines to which set it belongs; $FS(HGN)$ or $BS(HGN)$.

2. Based on membership, virtually deletes its links with every node belonging to the other set (as previously done by the HGN).

Fig. 3.10 gives an illustrative example of the rules applied by HGNs. Actually, when a HGN receives the query packet (Fig. 3.10 (a)), it performs the above-mentioned steps and broadcasts a packet to its neighbors. Upon receiving this packet, each neighbor of the HGN, in turn, performs the corresponding steps (Fig. 3.10 (b)). Afterward, the query process continues in the direction of the HLN.

Property 3. *The rules applied by HGNs maintain the boundary of Ω. That is; when exploring (opening) a hole, no link remains that crosses the newly created boundary.*

Proof. We denote by N_{prev} the previous node that delivers the peeling packet to the HGN. We also denote by $\sigma = \{V_{HGN} \cap V_{Prev}\}$ the common neighbors of the HGN and N_{prev}. Initially, we know that the HGN and N_{prev} belong to the boundary of Ω (see Fig. 3.10). This means by construction that there cannot exist any node, in the direction of the boundary and outside the communication range of the HGN and N_{prev} (i.e., it does not communicate with both of them), that can communicate with any node belonging to σ. Formally speaking, $\nexists N_i \mid L_{\{i,\ HGN\}} \notin \mathcal{L} \ \wedge \ L_{\{i,\ prev\}} \notin \mathcal{L} \ \wedge \ L_{\{i,\ n\}} \in \mathcal{L} \wedge \angle N_{\text{prev}} N_i HGN \leq \pi$, where $N_n \in \sigma$. On the other hand, thanks to the second step performed by the HGN and all its neighbors

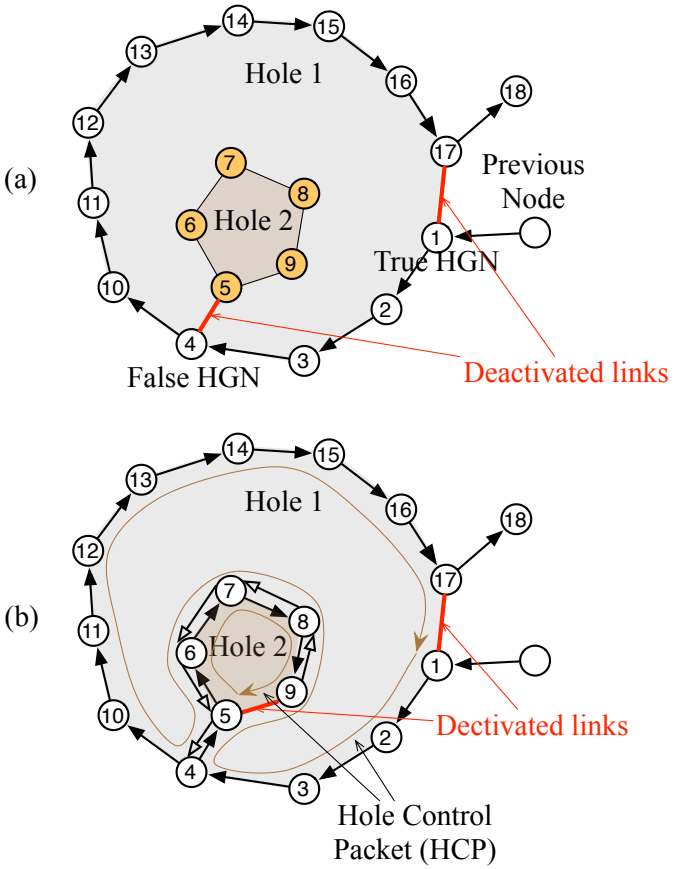


Figure 3.11: Nested holes. (a) Peeling the network without using HCP packets. (b) Peeling the network using HCP packets.

(i.e., links deletion), there cannot exist any link between the FS and BS sets. In addition, by construction, there cannot exit a link between the HLN and HRN (otherwise, a hole cannot exit). Finally, there cannot exist a link crossing the new boundary including N_{prev}, HGN, and HLN. □

The peeling approach described above has ensured query completeness on the numerous network configurations that have been randomly generated in our performance evaluation study (cf. Section 3.6). All alive connected nodes were visited. Nevertheless, from a theoretical point of view, we have constructed by hand, some particular topologies in which this version of PA fails to ensure query completeness. Fig. 3.11 (a) illustrates an example of such topologies. In this figure, when node N_1 receives the query, it considers itself as the HGN of hole 1 and applies the corresponding rules. That is, link $L_{(1,\ 17)}$ is deleted and the query is oriented towards node N_2. Similarly, when node N_4 receives the query, it considers itself as the HGN of the hole located to the left, deletes link $L_{(4,\ 5)}$, and then sends the query to node N_{10}. Consequently, all nodes located on the boundary of hole 2 will be lost and can never be visited.

This false HGN detection comes from the fact that node N_4 treats a single hole (*hole 1*) as being two different holes. To solve this problem, an additional rule must be executed by the HGNs. Actually, before performing steps 2, 3 and 4 described above, the HGN sends a control packet, named *Hole Control Packet* (HCP), which travels along the hole

in question, and visits all nodes located on its boundary. Eventually, due to its nature, the HCP will get back to the HGN (Fig. 3.11 (b)). In fact, the HCP packet plays a twofold role:

- First, it informs all nodes located on the boundary of the hole that this latter has an HGN. Consequently, these nodes will not consider this hole later on when they receive the query packet.
- Second, the HCP packet allows nodes to detect if their local stored holes constitute the same hole. For instance, node N_4 will know that its left and right holes are actually one single hole that has already got an HGN. This way, N_4 will not consider itself as the HGN of this hole and will allow the peeling process to continue after contributing to it.

  In the same context, later on, when the query packet gets to node N_5, it will identify itself as the HGN of hole 2, and consequently will apply the corresponding rules. That is, sending a HCP packet inside this hole and deleting the link with its right neighbor (Fig. 3.11 (b)).

Property 4. *The rules applied by HGNs do not partition the set of unvisited nodes Ω.*

Proof. Let us assume that the deleted links partition Ω and let us try to arrive at a contradiction. Further, let us suppose that there exists only one link between FS and BS sets (i.e., link connecting HGN and HRN). Actually, in this case, our initial assumption means that after links deletion, no path can connect the nodes of FS and BS sets. Which, thanks to the HCP packets sent by the HGN of the hole contradicts the existence of another path connecting HGN with HRN. Also, by construction, each hole has one and only one HGN. Consequently, there exists a path, passing through the HLN, connecting the HGN to every node in the hole, in particular, the HRN. In other words, the HGN and the HRN are connected through the hole. In addition to that, based on (1) the fact that the BS and the FS are internally connected (i.e., nodes inside them are connected), and (2) the fact that HGN and HRN belong respectively to the BS and FS, we can deduce that the FS and BS are connected. This contradicts the initial assumption. □

The overall peeling algorithm, executed by each node receiving the query packet, is depicted in Algorithm 1.

3.3.3/ STARTING NODE DETERMINATION

As previously mentioned, the peeling approach must start from a node that is located on the external boundary of the network. To this end, in this section, we propose a distributed localized technique able to efficiently identify the network boundary nodes. Actually, this approach is inspired from the work presented in [37, 38], and its key idea is to try to reach a virtual node located outside the network area using a geographic greedy routing (Fig. 3.12).

In more details, a packet aiming to reach the virtual node is initiated by the sink/query launcher. Actually, at first, the query launcher N_i sets the location of the virtual node and sends the created packet to its one-hop neighbor N_j which is the closest to the set virtual node (among N_i itself and all its neighbors). Knowing the virtual node location (from the received packet), every node in the path repeats this process until the packet

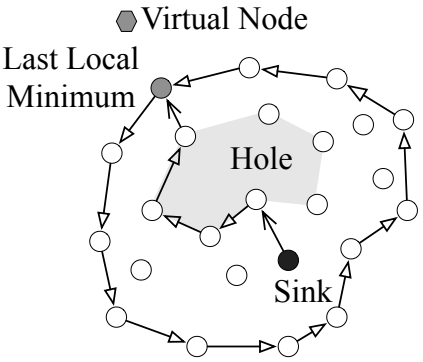


Figure 3.12: Network boundary nodes determination.

Algorithm 1 Peeling Algorithm.

```
Require: Receive peeling packet from previous hop.
 1: if (current node not visited for this query) then
 2:    Contribute to this query;
 3:    Mark itself as visited for this query;
 4:    if (current node is HGN) then
 5:       // If current node is located on more than one unexplored hole (Fig. 3.9)
 6:       Select the first counterclockwise hole among local unexplored holes;
 7:       Send an HCP packet inside the selected hole;
 8:       Upon receiving the HCP packet back, select the HLN as next hop;
 9:    else
10:       if (current node is not PBN and crossing links exist) then
11:          Add necessary information to the peeling packet; // cf. artificial holes.
12:       end if
13:       Select next hop among unvisited and BRN neighbors;
14:    end if
15: else
16:    Select next hop among unvisited and BRN neighbors;
17: end if
18: Determine the current node status; // BRN or not?
19: Send peeling packet to next hop;
```

falls into a local minimum situation (i.e., the packet cannot be delivered to the next hop as the current node is the closest one to the virtual node). From this node, also called the *local minimum node*, the recovery process is launched using the curved-stick boundary traversal approach [38] (Section 2.3.2.1). This process is repeated until the currently selected node is closer to the virtual node than the local minimum node. From this moment, the greedy forwarding is used again, and so on. For instance, in Fig. 3.12, plain arrows represent greedy communications and hollow ones represent boundary traversal (curved-stick) communications.

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

Property 5. *The last local minimum node N_l is a network boundary node (NBN).*

Proof. Initially, all nodes in the network are connected (connectivity assumption made in Section 1.3). Also, given the fact that (1) the curved-stick ensures boundary traversal [38], and (2) the greedy rule selects the closest node to the virtual node, we conclude that the

last local minimum node N_l has the shortest distance to the virtual node among all nodes in the network. Hence, N_l is located on the network boundary. □

Once a node has been identified as a network boundary node (NBN); e.g., the grey node in Fig. 3.12, the process of identifying all the other NBNs in the network can be triggered from this node using the curved-stick boundary traversal algorithm. In fact, upon receiving the curved-stick packet, each node (which is necessarily a NBN) must locally update its list of local holes (i.e., mark the external boundary of the network as *explored*), and this in order to avoid considering the external boundary of the network as a hole boundary.

It is worth emphasizing that the process of NBNs determination must be done only once and it is not necessary to repeat it each time the network is queried unless of course, the network topology has changed.

3.4/ PROOF OF CORRECTNESS

In this section, we provide proofs that demonstrate the peeling algorithm correctness. More precisely, we first prove that our approach terminates and does not fall into looping. In other words, we prove that PA generates a finite set of hops. Second, we prove that PA ensures query completeness (i.e., visits all nodes in the network).

The main facts on which the proof of correctness of PA is built are: (a) iteratively shrinking Ω while maintaining its connectivity (Property 1), (b) avoiding looping situations that can be caused by the creation of artificial holes (Property 2), and (c) transforming Ω (using the HGN rules) from a connected sub-network with holes (cycles) to a one without holes (Properties 3 and 4).

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

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

where N_v is the first traversed node which is not a BRN during the scan of Ω_i. For instance, in Fig. 3.11(b), if we denote the sub-network of nodes N_1 to N_{18} by Ω_k, then:

$$\begin{aligned} \Omega_{k+1} &= \Omega_k - \{N_1\}, \\ \Omega_{k+2} &= \Omega_{k+1} - \{N_2\}, \\ \Omega_{k+3} &= \Omega_{k+2} - \{N_3\}, \\ \Omega_{k+4} &= \Omega_{k+3} - \{N_9\}, \\ \Omega_{k+5} &= \Omega_{k+4} - \{N_8\}, \\ &\dots \end{aligned}$$

Lemma 1:

At each iteration i, PA finds a non-BRN node N_v.

Proof. Let us suppose that at step i, node N_v does not exist. In fact, since the curved-stick ensures boundary traversal, this assumption means that all the traversed nodes are BRNs. In other words, the traversed nodes form a hole. This, however, contradicts the non-existence of holes (cycles) guaranteed by the HGN rules. Therefore, we conclude that at each step i, N_v exists. □

Based on the previous lemma, we have:

Theorem 1:

PA generates a finite sequence of hops $\mathcal{H} = \bigcup_{i=0}^{k} N_i$ (i.e., PA terminates).

Proof. Initially, PA starts with a finite set $\Omega_0 = \mathcal{N}$. Then, at each iteration i, Lemma 1 ensures that a non-BRN node N_v exists. This means that the number of nodes in Ω is strictly decreasing at each iteration:

$$\forall \mid \mathcal{N} \mid \geqslant i > j \geqslant 0, \mid \Omega_i \mid < \mid \Omega_j \mid$$

Since PA starts with a finite set of nodes, then, it exits a k such that $\Omega_{k+1} = \emptyset$. Hence, we prove that $\mathcal{H}$ is finite and PA terminates. □

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

Definition 11: Boundary of Ω_i

The boundary of Ω_i is a cyclic sequence of unvisited or BRN 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, the next visited node N_v belongs to $B(\Omega_i)$. That is, N_v is located on the boundary of the current sub-network Ω_i.

Proof. We know that, initially, the peeling algorithm starts from a node that is located on the boundary of Ω_0. Based on this and given the fact that the curved-stick is a boundary traversal tool [38], each traversed non-BRN node N_v will be certainly located on the boundary of Ω_i. This can be proven as follows. In fact, initially, the peeling algorithm keeps scanning the nodes located on the boundary of Ω_0 until it hits a non-BRN node N_v. That is, initially, N_v is located on the boundary of Ω_0 and the property is true for $i = 0$. Now, let us suppose that the property is true for i and let us try to show that it remains as such for $i + 1$. First, since the property is true for i, this means that the currently removed node N_v is located on the boundary of Ω_i. Second, as previously mentioned, using the curved-stick as a traversal algorithm guarantees that the next hop N_w of N_v is located on the boundary of Ω_i. Consequently, it is clear that this next hop N_w is also located on the boundary of Ω_{i+1}. In fact, if N_w is not a bridge node (non-BRN) then it can be removed from Ω_{i+1} (marked as visited). Otherwise, starting from N_w, PA will keep scanning the boundary of Ω_{i+1} until it finds a non-bridge node that can be marked as visited. Hence, we conclude that the property is also true for $i + 1$. □

Theorem 2:

The sequence of hops $\mathcal{H}$, generated by PA, contains all nodes of the network (i.e., $\mathcal{H} = \mathcal{N}$).

Proof. To prove that PA visits all nodes in the network, let us assume that $\mathcal{H}$ (set generated at the end of network traversal) is different than $\mathcal{N}$ (set of all connected nodes in the network) and then let us seek to derive a contradiction. First, let us denote by N_v, each traversed non-BRN node that will be removed from Ω without altering its connectivity. Second, let us denote by N_m, one of the unvisited nodes that might be missed by PA ($N_m \neq N_v$, $\forall\ v$). Third, let us consider an unsigned integer j ($0 < j < n$) such that N_m belongs to Ω_j and does not belong to Ω_{j+1}. That is, N_m has been missed at iteration j. Actually, since at iteration j, PA hits only nodes that are located on the boundary of Ω_j and does not miss any one of them (Lemma 2 and Theorem 1); therefore, N_m cannot, in any case, be located on the boundary of Ω_j. Thus, N_m belongs to Ω_{j+1}. This, however, contradicts the initial assumption which says that N_m does not belong to Ω_{j+1} ($N_m \notin \Omega_{j+1}$). Consequently, we conclude that PA guarantees network traversal completeness in spite of the considered topology. □

To recapitulate, thanks to Theorems 1 and 2, we have the guarantee that, regardless of the considered network topology (irregular or regular, large-scale or sparse, with or without holes,etc.), the proposed peeling algorithm terminates (does not fall into looping) and ensures query completeness/accuracy.

3.5/ PEELING EFFICIENCY AND ROBUSTNESS ENHANCEMENT

As confirmed by the simulation results presented in the next section, the proposed peeling approach performs very efficiently in uniformly dense networks. Nevertheless, we have noted that in sparse networks with different densities, some nodes/links (because of their positions as BRNs) participate more often in the query evaluation than others. An example of such topologies is given in Fig. 3.13 (a), which illustrates one peeling round (grey nodes have been marked as *visited*, black nodes are *bridges*, and white ones have not been visited yet). Actually, in this example, it is clear that bridge nodes will be accessed more frequently than the others. For instance, if the peeling process continues until the end, the bridge black nodes and their related links will be accessed/used five times. Such a situation occurs because the peeling algorithm does not consider the visited nodes (which are not bridges). In other words, PA keeps only one alive path of bridges and uses it all the time.

As a matter of fact, this path linearity phenomenon brings with it several drawbacks and issues. First, it increases the communications required to accomplish a query. Second, and more importantly, it contributes somehow to the vulnerability of the peeling process. For instance, when a link in such a linear path of bridges is broken, the unvisited region will be partitioned despite the existence of other paths that can be formed by the already visited nodes. Actually, in such a scenario, since the current version of the peeling algorithm does not consider this solution or any other one, the peeling packet will be sent back to the previous hop and a false peeling termination will be detected.

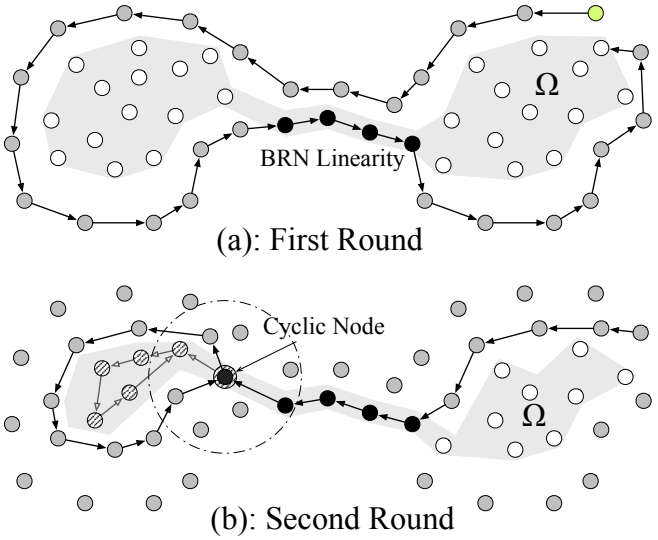


Figure 3.13: Peeling linearity phenomenon.

Repairing the connectivity of Ω (when it is partitioned due to the unavailability of a path) is a very challenging issue that deserves much research efforts. We mention that in the present section, our goal is just alleviating the impact of path linearity phenomenon on the peeling process. To this end, we have added a new rule that must be performed by certain specific nodes defined, as follows:

Definition 12: Cyclic Node (CN)

A node N_i is said to be cyclic, iff it conforms to the following two conditions: (a) it is a BRN (bridge node) and (b) it has received the peeling packet from two different directions.

In other words, as illustrated in Fig. 3.13 (b), the peeling process must pass, at least twice, by a cyclic bridge node N_i. Actually, when the peeling process gets back to a cyclic node (i.e., has accomplished a cycle), it defines a closed unvisited region (Fig. 3.13 (b)). The objective of the new rule that must be performed only by cyclic nodes is to peel this closed region before continuing the process somewhere else. This, as shown by Fig. 3.13 (b), means that each cyclic node must re-orient the peeling process towards the closed unvisited region. This way, the new applied rule ensures that the peeling process remains inside the region in question until visiting all its nodes. Eventually, once the desired region has been completely visited, the cyclic node in question will change its status from bridge to visited node and forwards the peeling packet.

Clearly, from a theoretical standpoint, the new applied rule noticeably reduces the serial path length, which considerably enhances the peeling performance. Yet, to confirm this point, in our experiments, in addition to PA, we have also implemented this improvement, calling it Enhanced Peeling Algorithm (EPA). For instance, applying the new rule to the network of Fig. 3.13 leads to more than 8% of communications reduction. Further, as it will be shown in the next section (3.6), this reduction is more substantial in large-scale network deployments.

3.6/ PEELING PERFORMANCE ASSESSMENT

In order to evaluate the performance of our proposed peeling algorithms, we have conducted several series of simulations using OMNeT++ [53] and its WSNs framework Castalia [54]. For comparison purposes, in addition to PA and EPA, we have also implemented the three following approaches:

1/ Tree-based querying: in this approach, called also *centralized in-network* querying [3], a tree rooted at the sink must be constructed. In fact, in the resulting tree, each node can have one or several children but must have only one parent, except for leaf nodes which will have no children and the sink which must have no parent. Once the tree is created and all nodes in the network are connected to it, query processing can be performed following two phases. The first phase is dedicated to query dissemination through the tree (starting from the sink). More specifically, upon receiving the query, each intermediate node forwards the query to its children. When the query arrives at leaf nodes, the latter trigger the second phase by sending their readings to their corresponding parents. From this point on, upon receiving data from its children, each intermediate node aggregates these measurements with its own reading and sends the result to its parent, and so on. This process stops when all data is received by the sink.

2/ Iterative querying: in this approach, called also *distributed in-network* querying [14, 15], the QL can launch a query by simply broadcasting a packet to its immediate neighbors. Upon receiving the query, each node re-broadcasts it and starts exchanging its own estimate with those of its immediate neighbors. More precisely, at each iteration of the process, the local estimate of each node is updated with the weighted data received during the previous iteration, and so on. This process terminates when each node detects that the difference in values between two consecutive terms of the estimated parameter is smaller than a certain predetermined threshold.

3/ Depth-First Search: this well-known serial technique [26] constructs for each query a path that stems from the QL and extends, as much as possible, toward the unvisited nodes. In other words, DFS explores the network as far as it can, and when it gets stuck, it backtracks (looking for other possible unvisited nodes). To be able to do so, each node must be aware of its unvisited neighbors as well as the identity of its parent. We recall that DFS requires $2*(n-1)$ hops to explore a network of n connected nodes and get back to the sink/QL.

3.6.1/ EVALUATION METRICS

Note that, regardless of the considered network configurations, all the implemented approaches have been proven to ensure query completeness (all nodes contribute to the query). In the same context, note also that, except DFS, we have not considered any other serial approaches such as space-filling curves algorithm [24], because they do not satisfy this requirement (they do not guarantee query accuracy).

For a single query, in addition to completeness, the other considered comparison metrics are:

- Required communications: total number of packets (transmissions) used to query the network and aggregate data (i.e., control and data packets).

Table 3.1: Peeling simulation configuration and parameters

Parameter	**Value(s)**
Field dimensions (m^2)	1000 x 1000
Nodes number	100, 150, 200, . . . , 500
Nodes deployment	Uniform
Nodes transmission range (m)	150
Nodes initial energy (J)	100
Query launcher location	Random
Query packet size ($byte$)	50
Radio type	CC2420
Radio data rate ($kbps$)	250
MAC protocol	CSMA/CA
Sensed values interval	[0, 0.4]
Convergence threshold (for Iterative approach)	10^{-2}

- Response time: time that elapses between the instant when the QL issues a query and the instant when it receives the corresponding answer.
- Consumed energy: total energy consumed by the network during the querying process. In fact, in order to estimate the energy required by each node to transmit and receive packets, we have used the energy consumption model proposed in [55]. The radio in this model dissipates $\mathrm{E_{elec}} = 50\ \mathrm{nJ/bit}$ to run the transmitter or receiver circuitry and $\epsilon_{\mathrm{amp}} = 100\ \mathrm{pJ/bit/m^2}$ for the transmitter amplifier. Thus, to transmit a k-bit packet a certain distance d using this model, the radio consumes:

$$E_{TX}(k, d) = k * E_{\mathrm{elec}} + k * d^2 * \epsilon_{\mathrm{amp}}$$

and to receive this same packet, the radio consumes:

$$E_{RX}(k) = k * E_{\mathrm{elec.}}$$

In addition to these metrics, we consider the number of queries a network can support before one of its nodes runs out of energy. This metric represents the network lifetime in terms of supported queries.

3.6.2/ SIMULATION PARAMETERS

The considered simulation parameters are summarized in Table 3.1. Note that the sensed values interval, which impacts only the iterative querying approach (described above), has been set to $[0, 0.4]$. Actually, as explained below in Section 3.6.3.1, the length of this interval has been purposely set to a small value in order to not penalize the iterative approach. In the same context, we mention that the sensed values, taken from the chosen interval, are normalized uniform random values.

As regards Table 3.2, it shows the average degree of nodes in the different deployments; i.e., the average number of neighbors of each node. More precisely, in this table, the first

Table 3.2: Peeling average nodes' degree.

100	**150**	**200**	**250**	**300**	**350**	**400**	**450**	**500**
8	12	16	20	25	29	33	37	42

line represents the number of nodes in the network, whereas the second one represents the corresponding average degree.

3.6.3/ SIMULATION RESULTS

For each query configuration (i.e., network deployment and query launcher location), we have run the studied approaches and recorded the obtained results. Actually, the results presented below represent the mean values of 30 runs on each configuration.

3.6.3.1/ SINGLE QUERY PERFORMANCE

Figures 3.14, 3.15, and 3.16 summarize the obtained single query evaluation results. These curves clearly confirm the effectiveness of serial approaches over centralized (i.e., parallel tree-based) and iterative (i.e., parallel structure-free) ones. Although the improvement is minor in sparse network configurations, it becomes more noticeable when the network density increases. For instance, in terms of communications, PA is about 6 times better than the iterative approach and 4 times better than the centralized one. As a matter of fact, even though the centralized approach allows concurrent transmissions, the generated collisions (during tree construction, query answering, etc.) prevent this approach from exhibiting better performance. The obtained results also reveal that the iterative approach (because of its intrinsic iterative behavior) is more costly in comparison with the other approaches. Actually, we have concluded (from other results not reported in this manuscript) that when the standard deviation between sensed data is important and the network is large, the iterative approach will need more iterations to reach convergence. In fact, this happens particularly when nodes with important differences in sensed data are located far away from each other.

We note that, in all the studied approaches, the three considered metrics increase with the increase of the number of nodes in the network. This trend, however, is less pronounced in serial approaches, which means that they are more scalable than the centralized and iterative techniques.

Now, if we consider only the studied serial approaches (PA versus DFS), the obtained results clearly display the outperformance of our algorithm in terms of communications, time, and energy consumption. This improvement, while minor in sparse configurations (from 100 to 150 nodes), becomes more significant in dense networks (from 70% to 90%). Actually, this is primarily due to the fact that our approach is localized and structure-free (only one-hop information is required and no predetermined visiting path needs to be followed). As regards DFS, in fact, its bad performance comes mainly from its costly mandatory backtracking process.

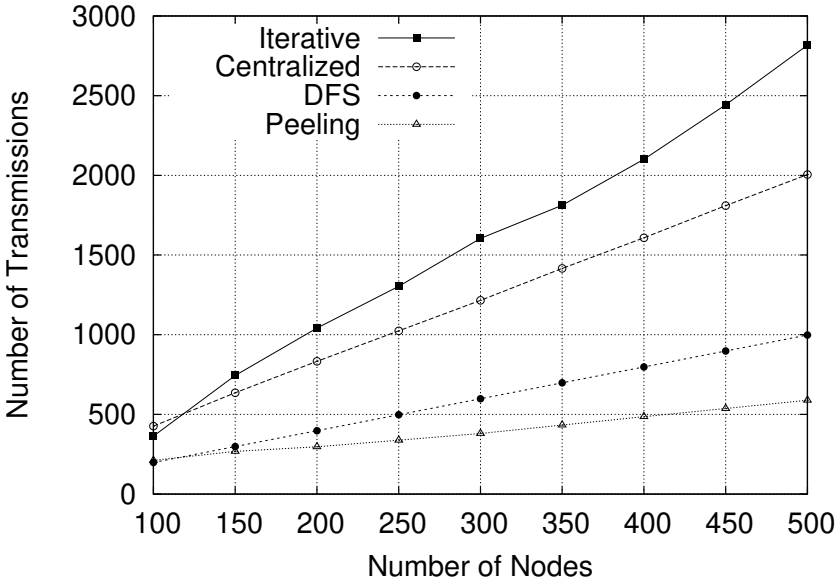


Figure 3.14: Peeling one query performance: number of transmissions.

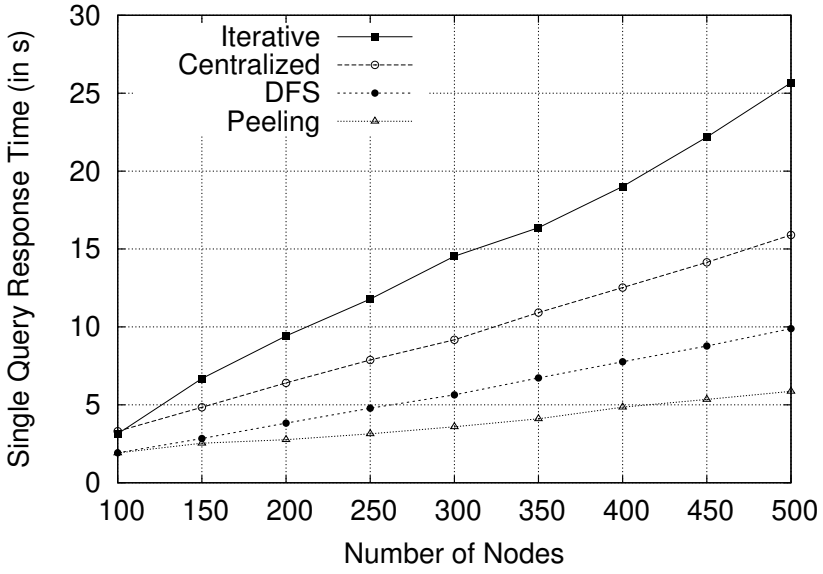


Figure 3.15: Peeling one query performance: average response time.

3.6.3.2/ NETWORK LIFETIME

In these series of simulations, we measure the effect of the different studied approaches on the overall network lifetime. To this end, prior to deployment, we have provided each node with the same initial energy provision of $5J$. Then, we have run the considered approaches by launching a query every 2 minutes. The network is assumed to be dead when any of its nodes (i.e., the first node) runs out of energy. Fig. 3.17 plots the recorded number of supported queries. In fact, as this figure shows, the obtained results confirm the effectiveness of our peeling approach in terms of prolonging the network lifetime and supporting more queries compared with the other approaches. More specifically, the

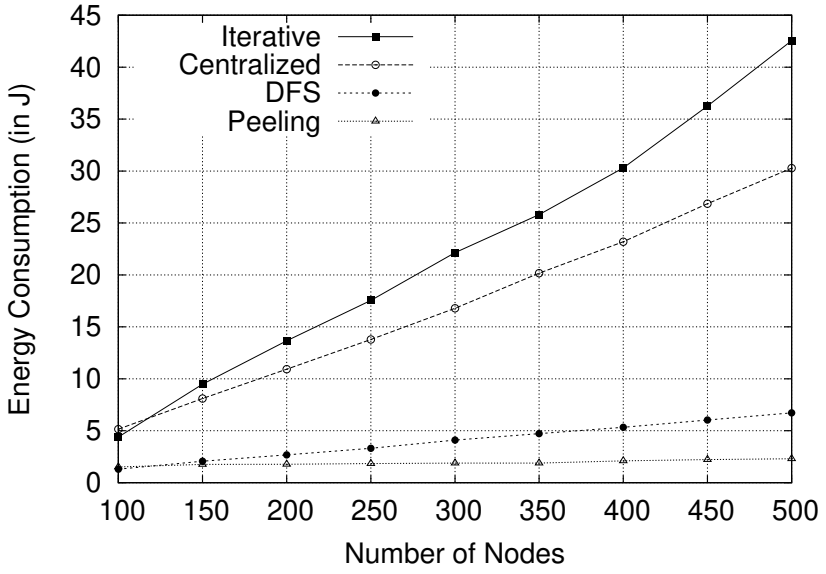


Figure 3.16: Peeling one query performance: total energy consumption.

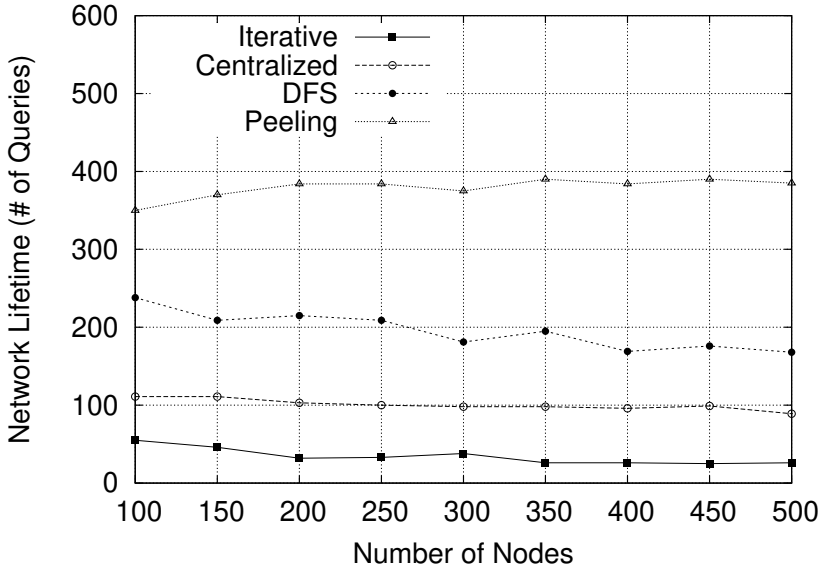


Figure 3.17: Peeling performance: network lifetime.

improvements are nearly 500%, 300% and 50% in comparison, respectively, with the iterative, centralized, and DFS approaches.

3.6.3.3/ COLLISIONS IMPACT

As previously mentioned, one of the main interesting features of serial approaches is that they do not generate collisions. Actually, in these approaches, all transmissions are sequential and at any moment in time, only one node is allowed to send data. To quantify the effect of this collision-free feature on query responsiveness, in this section, we compare the studied serial approaches (i.e., PA and DFS) while enabling and disabling

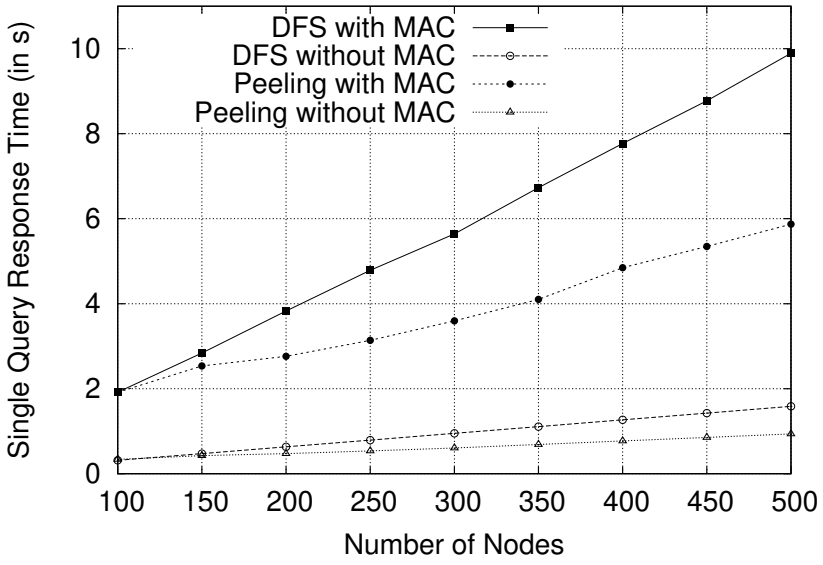


Figure 3.18: Peeling query response time while enabling and disabling MAC functions.

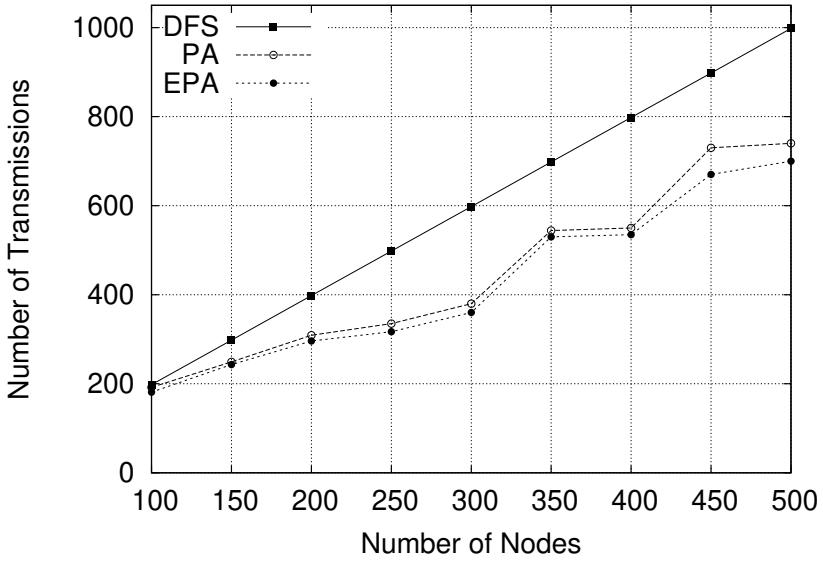


Figure 3.19: PA and EPA one query performance: number of transmissions.

the MAC layer. We recall that the considered MAC is the CSMA/CA protocol provided by Castalia [54] (Table 3.1).

The obtained results confirm that disabling the MAC layer does not affect either the required number of communications nor the consumed energy, but actually, impacts the query response time (Fig. 3.18). More specifically, a clear time improvement can be attained when no MAC layer is utilized, especially in dense networks (about six times of improvement). The reality of this enhancement being more noticeable in large-scale deployments validates the appropriateness of serial approaches for this kind of networks. However, the only downside here is that the MAC layer can be disabled only in the case where just one query is present in the network.

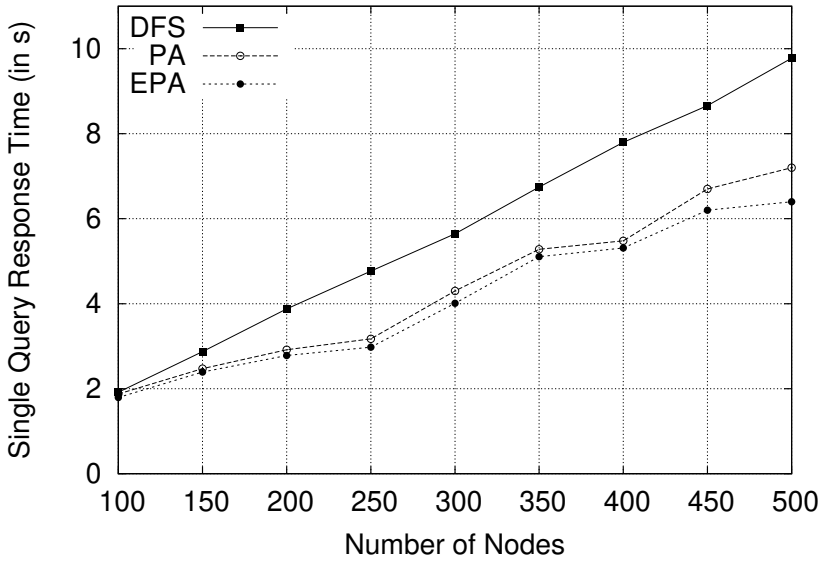


Figure 3.20: PA and EPA one query performance: average response time.

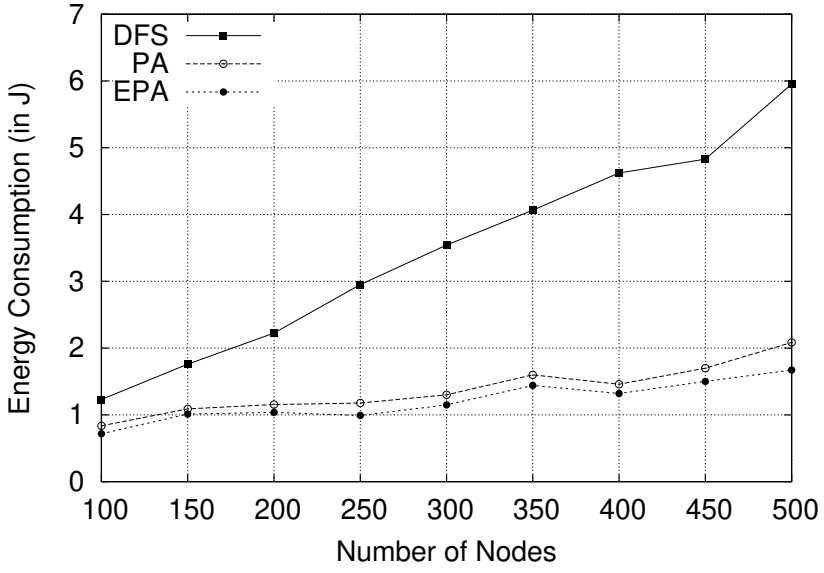


Figure 3.21: PA and EPA one query performance: total energy consumption.

3.6.3.4/ PA VERSUS EPA

The objective of this section is to compare the performance of PA and EPA in non-uniform networks. To this end, the field dimensions and nodes deployment have been set, as follows. At first, the area of interest has been split into three horizontally adjacent regions. Then, the obtained left and right regions were given the same network densities (i.e., uniform distribution) and the same size (300 x 1000 m^2). As regards the middle region, it was given a different density and size (400 x 1000 m^2).

The obtained single query evaluation results are depicted in figures 3.19, 3.20, and 3.21. Exactly as expected, EPA outperforms PA in non-uniform networks. The reason behind

this (as previously explained) is that EPA uses the notion of cyclic nodes to optimize the visiting peeling path. The improvement is about 4% in sparse configurations and about 10% in dense ones, which validates the effectiveness of EPA in large-scale and non-uniform networks.

We mention that the obtained results (not shown here in this manuscript) also confirm that PA and EPA have almost the same performance when the network deployment is uniform.

3.7/ CONCLUSION

In WSNs, serial query processing approaches have shown their effectiveness in terms of reducing communications and energy consumption. Nevertheless, besides the fact that query completeness was not ensured (i.e., visiting all nodes in the network) [24], previous research works have not explicitly addressed the key issue of constructing the visiting path [20,22]. In this chapter, we have tackled these important issues by proposing a serial localized approach, named the Peeling Algorithm (PA) [18]. The main advantage of this proposed approach is that it does not require any additional knowledge other than what is traditionally available in WSNs. That is to say, PA is a one-hop approach that does not require any information about the network topology. In this chapter, we have formally proven that PA ensures query completeness and is free of looping. In addition to this, the obtained simulation results have clearly highlighted the effectiveness of PA in terms of reducing communications and energy/time consumption.

Despite its good performance and efficiency, PA still raises several research challenges that deserve further investigations. Among these issues, we mention that the performance of PA can be further enhanced by reducing the traversal path length. In fact, this is precisely the aim of the next chapter.

4

SPREADING AGGREGATION

4.1/ INTRODUCTION

Recently, numerous works have shown that serial aggregation in large WSNs is scalable and very efficient, in terms of avoiding collisions and conserving energy and, more importantly, in terms of reducing response time [18, 25]. In this chapter, a novel serial data aggregation approach, called Spreading Aggregation (SA), is proposed with the aim of shortening the traversal path and further reducing communications [39, 40]. First, given the fact that it is not based on a pre-established itinerary, SA is maintenance-free and does not require any communications in this regard. Each time an aggregation process is launched, a new path is built, which decreases vulnerability to failure in links and nodes and allows the approach to handle topology changes. Second, SA is a localized approach that relies only on the one-hop neighbors' table of each node to gradually construct the path, which makes it very scalable. A third interesting feature of SA is the merge of path construction and data processing. While the path is progressively constructed, data is simultaneously aggregated, saving a considerable amount of time and energy. In addition to all that, SA also saves energy and time due to its collision-free nature. In fact, in SA, only one packet is present in the entire network at any given time.

In this chapter, we formally prove the correctness of SA (i.e., free of looping and ensures the traversal of all connected nodes). Furthermore, the extensive OMNeT++ simulations, we performed, confirm that the proposed approach reduces communications, scales well in large networks, and conserves time and energy. The obtained results also show that SA outperforms state-of-the-art serial approaches.

Briefly, this chapter is structured in five sections, as follows. The next section (4.2) introduces the preliminaries and background required throughout the chapter. Section 4.3 presents and details the operation of the proposed approach, while Section 4.4 provides its proof of correctness. Section 4.5 presents and comments on the evaluation results. Finally, Section 4.6 concludes the chapter.

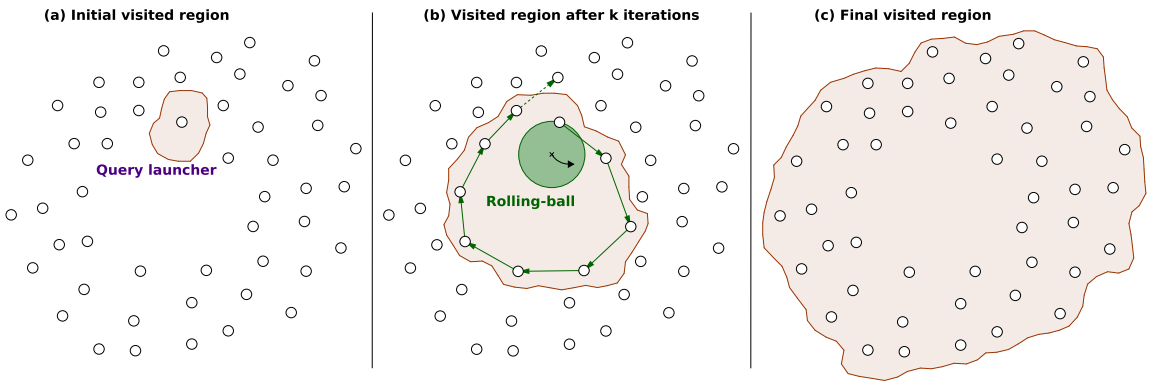


Figure 4.1: Illustration of SA: starting from the launcher node, the traversal mechanism extends the visited region as a *"stain"* until reaching the entire network (and this independently of the topology as shown in the rest of the chapter).

4.2/ BACKGROUND AND GENERAL IDEA

Indeed, different from previous serial approaches, our second proposal does not rely on (1) any initialization phase as is the case in the peeling algorithm [18] (i.e., finding and marking the external boundary of the network) or (2) any intermediate initialization phases as is the case in GBT [25] (i.e., searching for unvisited nodes). Rather, it can immediately commence query processing from any node in the network whatever its location. As a matter of fact, as shown in Fig. 4.1, the main intuition behind our new proposal is to start from the query launcher (considered as the initial visited region) and then, at each step, extend this region by exploring the unvisited nodes located on its boundary. In fact, as explained later in Section 4.3.2, at each step, each explored node is added to the visited region, except some particular nodes that are mandatory to preserve the network connectivity. The process of expanding the visited region is gradually repeated until reaching all nodes in the network (i.e., process stopping criterion). In other terms, our new approach, by means of a boundary traversal tool, serially traverses the network and visits nodes, hence creating a *virtual boundary* between *visited* and *unvisited* nodes. This boundary is progressively enlarged, at each node traversal, until all nodes in the network are visited.

SA relies on two fundamental concepts: (1) the use of a boundary traversal algorithm capable of keeping the visiting process always on the boundary of unvisited nodes, and (2) the available information about the status of the neighbors of each node (i.e., visited or not). First, to maintain the status of nodes, a flag has been added to each neighbor showing its current status. The process of updating this flag is done thanks to the broadcasting nature of communications in WSNs. That is, when a node sends a packet, all its neighbors can overhear and receive it. Consequently, this process does not incur any additional communication cost [56,57]. Second, as for the considered boundary traversal algorithm, we recall that it has to fulfill two essential requirements:

- In order to ensure complete network traversal, the most important requirement that must be fulfilled is *correctness*. That is to say, the algorithm must not miss any boundary node. A good example of such misbehavior has been illustrated in [37].
- In order to propose a time and energy-efficient serial approach and to ensure scala-

bility for large-scale deployments, the other requirement that must be fulfilled by the used boundary traversal algorithm is *localizability*. This means that the boundary traversal algorithm must rely only on the local information available at each node (node's position and that of its one-hop neighbors).

In this chapter, the rolling-ball [41] has been selected as a network traversal tool because, as it has been demonstrated in [37, 38], it can visit more nodes than the curved-stick. Actually, while this is a drawback in data routing context[1], it can be seen as an advantage in the case of data aggregation since the prime objective in this context is to visit/query nodes as quickly as possible. The rolling-ball details can be found in Section 2.3.2.

The proposed spreading approach because of its localized nature on one hand and the requirements mentioned in Section 1.2 (particularly completeness) on the other hand, raises several research issues that this chapter covers in details. For instance, the main problem in our case was how to start query processing (or more precisely network traversal) from a non-boundary or inner node (i.e., a node that cannot hold an empty valid rolling-ball)? To this end, as shown later in the chapter, we have developed an *adaptive rolling-ball mechanism* that is able to cope with this difficulty (Section 4.3.3). Furthermore, starting from any node in the network, the proposed network traversal mechanism has to be designed to be able to effectively deal with all kind of network configurations; i.e., handle all possible topologies that a random deployment can generate including those with communication holes.

4.3/ PROPOSED APPROACH: SPREADING AGGREGATION

In this section, we present our second serial data fusion approach, called Spreading Aggregation (SA), and explain its operation through algorithms and illustrative examples.

At first, all nodes in the network are considered *unvisited*. That is, the set of unvisited nodes, denoted Ω, is equal to the set of nodes ($\Omega = \mathcal{N}$) and the set of visited nodes, denoted Γ, is empty ($\Gamma = \emptyset$). To start the serial aggregation process, the QL sets a rolling-ball on one of its boundaries and lets it serially visit the network. As illustrated in Fig.4.2, the rolling-ball (aggregation packet) launched by N_0 moves in a localized fashion from node to node, while marking them as *visited*. To find the next appropriate hop, each node must be aware of the status of its neighbors; visited or not (e.g., node N_{21}). As a matter of fact, as previously mentioned, the broadcast nature of wireless communications renders this operation effortless; i.e., all of a node's neighbors can clearly hear the communication when a rolling-ball is delivered to the next hop; consequently, they will know the status of that node. We underline that because nodes have to listen for a packet that is not addressed to them (idle listening), a low power listening mechanism (LPL) can be utilized [56, 57]. Actually, since only one packet is present in the entire network at any given point in time, no sophisticated MAC protocol is required.

In the end, once the rolling-ball has successfully browsed the entire network, the last node in the path can send the obtained result (aggregated data) to the QL using any efficient geographic routing.

[1] The curved-stick outperforms the rolling-ball in the context of anti-void data routing and derives shorter paths around holes.

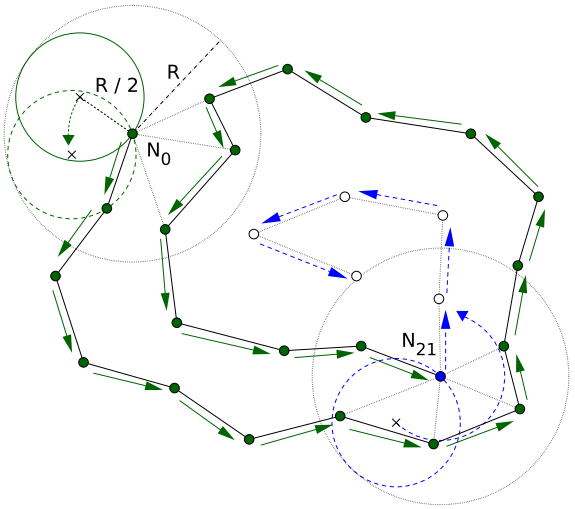


Figure 4.2: Rolling-ball network traversal launched by node N_0.

Although the approach described above is localized, it faces, however, two major issues: (i) ensuring the connectivity of unvisited nodes while traversing the network, and (ii) launching the aggregation from a non-boundary QL (i.e., a node on which a valid empty rolling-ball cannot be hinged). We examine these two issues respectively in subsection 4.3.2 and 4.3.3. In the last subsection 4.3.4, we present the technique proposed to solve the looping issue created by the connectivity maintenance solution proposed in subsection 4.3.2. Before getting into the details of our approach, and in order to ease the chapter understanding, in the next subsection, we give a quick overview of SA.

4.3.1/ BRIEF DESCRIPTION OF SA

In the beginning, all nodes including the QL must be aware/create their lists of boundaries[2]. These lists are used later in the algorithm to detect and remove cycles. In order to launch a query, the node in question, called QL, checks whether it is a boundary node or not[3], and acts accordingly. First, let us begin by briefly describing the steps performed starting from a boundary QL, then, the ones carried out starting from a non-boundary QL.

A boundary QL starts by placing a rolling-ball on one of its boundaries and after that sending an IBS (Initial-Boundary Scan) packet to mark this initial boundary. The aim behind using the rolling-ball to traverse and mark the initial boundary, node by node, is to help find any possible cycles in the network. Once this step is done, the QL launches query processing by again spinning the rolling-ball counterclockwise (inside the initial boundary) and selecting the first hit one-hop neighbor as next hop. Upon receiving the aggregation packet, each node in the network repeats the same process of spinning the received ball and selecting a next recipient. Actually, in addition to these two steps, each node determines whether it can leave the querying process, or must remain involved in it if it is necessary for the network connectivity (Section 4.3.2). Furthermore, to prevent looping around communication holes (Section 4.3.4), each node upon receiving the ball

[2]A boundary is defined by a right neighbor, a left neighbor, and a Boolean indicating whether the boundary has been traversed by the rolling-ball.

[3]A boundary node is able of holding at least one valid empty rolling-ball and its list of boundaries must contain at least one item.

checks if it has an unmarked disjoint boundary (that is, a boundary that has not been explored by the IBS packet sent by the QL). If this is the case, a cycle has been detected and the current owner of the rolling-ball (called in this situation *portal node*) must start cycle-removal by first sending a DBS (Disjoint-Boundary Scan) packet responsible for marking the disjoint boundary. We mention that the role of both IBS and DBS packet is to allow nodes to detect cycles. In fact, after receiving the rolling-ball DBS packet back, the portal node removes the found cycle by (1) cutting some specific links with its one-hop neighbors, and (2) sending an LC (Link-Cut) packet to those neighbors ordering them to do the same.

Since a non-boundary QL is unable of initiating/launching an empty optimal rolling-ball with radius $R/2$, it sets a smaller empty rolling-ball and launches it in the network (Section 4.3.3). This shrunken non-optimal rolling-ball performs like the optimal one by gradually spreading the visited region. The only difference is that the non-optimal ball gets enlarged if possible at each hop. As a matter of fact, in order to guarantee the correctness of our approach, the objective is to reach the optimal value (i.e., create a boundary) as quickly as possible. Once the optimal radius of the rolling-ball is reached, the traversal process continues on the boundary of the unvisited region as previously described. The algorithm ends at a node when the neighbors of the latter have all left the traversal process.

4.3.2/ CONNECTIVITY MAINTENANCE DURING NETWORK TRAVERSAL

The distributed network traversal algorithm, shown in Fig. 4.2, performs efficiently in dense deployments but fails to visit all nodes in the case of sparse networks. More specifically, this serial aggregation technique fails when the unvisited nodes, which are mandatory for the connectivity of other unvisited nodes, mark themselves as visited and no longer participate in the traversal. For example, in Fig. 4.3, nodes N_4 and N_5 are mandatory for the connectivity of Ω. If they consider themselves as *visited*, data aggregation will not be complete. We recall that the traversal termination is detected at a node when the neighbors of this latter have all been marked as visited (e.g., node N_6 in Fig. 4.3). To solve the partitioning problem, a new status has been introduced, namely *linking* nodes. A linking node is defined as follows:

Definition 13: Linking-Node (LN)

A node is said to be a Linking-Node if at least two of its one-hop unvisited or linking neighbors cannot communicate without its help.

In other words, a linking node is the only connection point or path between at least two of its immediate unvisited or linking neighbors. In Fig. 4.3 nodes N_4, N_5, and N_7 are linking nodes, while the others are not. We note that by considering the network model described in Section 1.3, a node can locally (based only on its one-hop neighborhood information) decide whether or not it is a linking node.

To ensure the connectivity of Ω and consequently guarantee aggregation completeness, linking nodes must remain involved in the traversal. As Fig. 4.4 shows, when the owner of the rolling-ball detects locally that it is a linking node, it marks itself as such and selects the next hop (linking or unvisited neighbor). When a linking node receives the aggregation packet again, it does not contribute to the query but changes its status to *visited* or remains as a linking node and so forth.

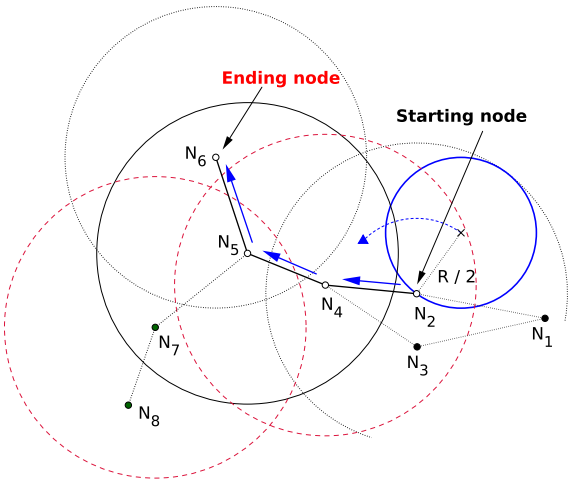


Figure 4.3: Disconnectivity of Ω during network traversal. Nodes N_1, N_3, N_7, and N_8 are left unvisited.

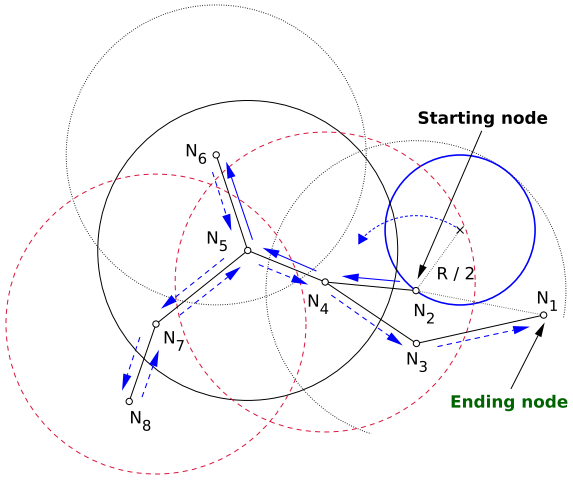


Figure 4.4: Network traversal with the use of the linking nodes concept.

Lemma 3:

Linking nodes ensure the connectivity of Ω.

Proof. This lemma can be proven by contradiction, as follows. First, at the beginning, we have $\Omega = \mathcal{N}$ and thanks to the network connectivity assumption made in Section 1.3, we know that Ω is connected. Second, let us assume that the removal of the currently explored node N_i from the traversal process disconnects Ω. Third, let us suppose that this node N_i is not a linking node. In fact, according to Definition 13, being a non-linking node means that for each pair of neighbors of N_i, there is a path of nodes that can connect them without passing by N_i. Therefore, we conclude that Ω can be connected without considering node N_i, which contradicts the hypothesis that assumes that excluding N_i from the traversal process disconnects Ω. □

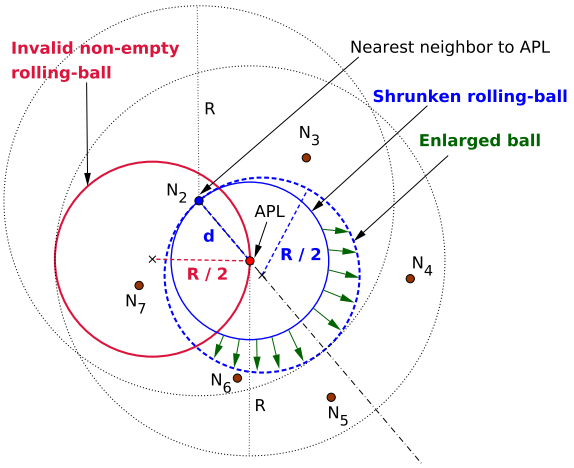


Figure 4.5: Shrunken rolling-ball setting and enlargement.

Algorithm 2 `setShrunkenRollingBall()`

```
1: // set shrunken ball's center
2: center = currentNode.location;
3: // set shrunken ball's radius
4: radius = distance(currentNode, nearestNeighbor);
5: // check obtained ball
6: if (radius > optimalValue){
7:    adjust rolling_ball to optimal shape while keeping it attached to
      nearestNeighbor;
8: }
```

4.3.3/ AGGREGATION LAUNCH BY NON-BOUNDARY NODES

On one side, in order to ensure a correct boundary traversal, the main requirement of the rolling-ball is to be empty of unvisited and linking nodes both initially, and all the time. On the other side, one of the interesting features of the proposed approach is that it does not make any restriction or assumptions about the QL's location (Section 1.3). Provided that the network is connected, the QL can be any node in the network. As a consequence of these two points, being non-boundary means that the QL cannot set a valid rolling-ball, and hence cannot launch the aggregation process. For instance, as Fig. 4.5 demonstrates, the red rolling-ball set by the non-boundary QL is invalid (contains nodes N_2 and N_7).

To overcome this issue, we proposed the following solution. The non-boundary QL creates a *shrunken rolling-ball*. As shown in Fig. 4.5 and Algorithm 2, this ball is centered at the QL's location, and its radius d is equal to the distance between the QL and its nearest neighbor. Setting the ball this way respects its emptiness requirement (i.e., it does not contain unvisited or linking nodes). In fact, we have the guarantee that the only node located inside the shrunken rolling-ball is the QL which is a visited node and could not be a linking node since it is a non-boundary node. It should be noted that sometimes the distance between the QL and its nearest neighbor can be larger than the optimal radius of the ball (i.e., $R/2$). In this case, the QL has to adjust the rolling-ball to its optimal shape (i.e., $d = R/2$) while keeping it hinged to the nearest neighbor (Algorithm 2).

Algorithm 3 `checkReceivedRollingBall()`

```
1: if (rolling_ball is optimal) { // radius == R/2
2:   call Algorithm 7;
3: }
4: else {
5:   // rolling-ball is shrunken (radius < R/2)
6:   enlarge rolling_ball;
7:   if (rolling_ball is optimal) {
8:    call Algorithm 7;
9:   }
10:  else {
11:   // rolling-ball is still shrunken
12:   change status to VISITED or LINKING_NODE;
13:   spin rolling_ball;
14:   sendAggregationPacket(next_hop);
15:  }
16: }
```

Lemma 4:

A non-boundary QL cannot be a linking-node.

Proof. We recall that being non-boundary means that node N_i cannot hold an optimal rolling-ball. To illustrate our point, let us first assume that the only nodes present in the network are node N_i and its one-hop neighbors. Second, let us imagine that this non-boundary node N_i has set an optimal rolling-ball on the external network boundary. More specifically, the rolling-ball is owned by node N_i but is hinged at its farthest (most distant) neighbor (to ensure its emptiness of any other neighbor). Under those circumstances, being non-boundary means that if node N_i rolls the ball starting from its farthest neighbor and ending at this same neighbor, then node N_i will not be hit by the ball. This last statement also means that a path of one-hop neighbors exists surrounding node N_i, preventing the rolling-ball from touching it. That is, for each pair of neighbors of N_i, a path exists that connects them without passing by N_i. Consequently, according to Definition 13, we conclude that a non-boundary QL can never be a linking-node. □

Once the shrunken (or optimal) rolling-ball has been set, the non-boundary QL marks itself as *visited* (Lemma 4) and forwards the aggregation packet to its nearest neighbor (Algorithm 4). This latter, upon receiving the packet (Algorithm 6), checks the received ball (Algorithm 3). In the case of an optimal rolling-ball, the neighbor simply continues the aggregation process by calling Algorithm 7 (explained later). Otherwise, as shown in Fig. 4.5, first, the nearest neighbor of the QL enlarges the received ball as much as possible. Afterward, it again checks the rolling-ball. If it became optimal, it executes Algorithm 7, otherwise, it (1) changes its status to visited or linking node, (2) spins the shrunken ball, and (3) delivers the packet to the first hit unvisited neighbor (node N_3 in Fig. 4.5). 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. It is worth highlighting that in order to remain in the same boundary and ensure network traversal, the received shrunken-ball must be enlarged as much as possible by its owner before being spun to choose the next hop.

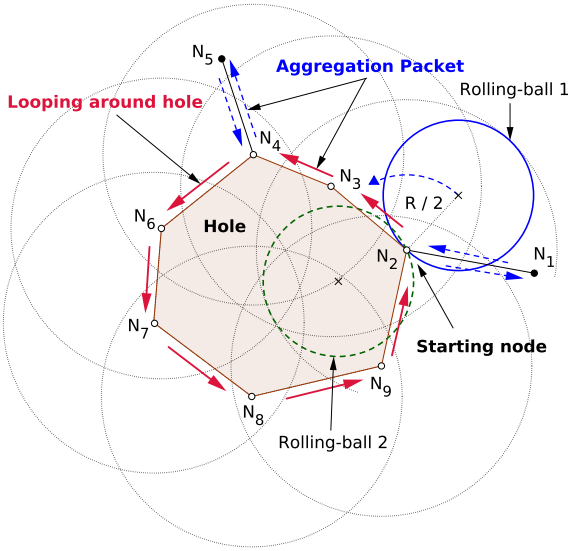


Figure 4.6: Example of cycles. Looping due to the use of linking nodes concept.

4.3.4/ CYCLES DETECTION AND REMOVAL

The notion of linking nodes (Definition 13) ensures the connectivity of Ω (Lemma 3) and respects the design requirement of proposing a localized aggregation approach. Nevertheless, due to its reliance on the local limited knowledge of nodes, it creates a risk of looping. In the present section, we exhibit the looping problem along with the proposed solution.

The most important condition for a node to be truly a linking node necessary for the connectivity of Ω is not to be contained in a cycle. For example, node N_3 in Fig. 4.6 is not required to maintain the connectivity of Ω, but given its limited knowledge (i.e., its one-hop neighbors and their locations), this node believes the opposite, thus creating a looping scenario. The same applies for nodes N_6, N_7, N_8, and N_9. Actually, as Fig. 4.6 clearly shows, looping situations occur in the presence of disjointed boundaries in the network (e.g., the external boundary of the network and the boundary of a hole)[4]. In these cases, due to the use of linking nodes concept, the rolling-ball will get stuck (loop) inside the currently traversed boundary and will not be able to leave it.

In order to ensure a proper identification of disjoint boundaries (cycles) in the network and to avoid looping, each node must first be aware of its boundaries. From the local limited perspective of a node, a boundary is defined by a right side, a left side, and a Boolean indicating whether the boundary has been explored. For instance, node N_2 of Fig. 4.6 has three boundaries, whereas N_6 has two (the first defined respectively by N_7 and N_4 as its right and left sides, while the second has N_4 and N_7 as its respective right and left sides). Initially, all boundaries must be marked as unexplored. The last step that must be taken to verify the existence of disjoint boundaries is to mark as *explored* the initial boundary from which the aggregation process will be launched. To do so, as Algorithm 4 depicts, the boundary-QL issues a rolling-ball control packet, named IBS (*Initial-Boundary Scan*) Packet.

As shown in Fig. 4.7 and Algorithm 5, the IBS packet traverses the initial boundary node-

[4]Note that the number of cycles in a network is equal to its number of disjoint boundaries, minus one.

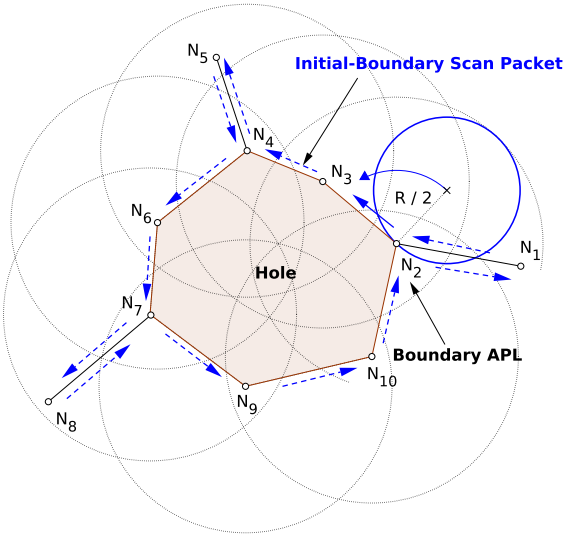


Figure 4.7: Initial boundary marking via IBS Packet issued by boundary-QL (node N_2).

Algorithm 4 - Spreading Aggregation Initialization.

```
1: To trigger serial data aggregation, the QL calls the following method:
2: void initializeAggregation(){
3:   if (currentNode.isBoundaryNode()){
4:     set a rolling_ball on any boundary;
5:     send_IBS_Packet(); // to mark initial boundary
6:   }
7:   else {
8:     // current node is not a boundary node
9:     setShrunkenRollingBall(); // Algorithm 2
10:    currentNode.status = VISITED; // Lemma 4
11:    sendAggregationPacket(nearestNeighbor);
12:  }
13: }
```

by-node and orders them to locally mark the boundary in question as *explored*. Once the initial boundary has been successfully marked, the determination of disjoint boundaries can be done locally by each node. Simply, if the currently traversed node (including the QL) has an unexplored boundary, then this latter is certainly a disjoint boundary (cycle) that must be opened, and the node in question is a *portal node* responsible for this cycle removal.

Definition 14: Portal Node (PN)

A node is said to be a Portal Node if it has at least two disjoint boundaries.

In the case of a non-boundary QL as described earlier in Section 4.3.3 (aggregation launch by non-boundary nodes) and shown here in this section in Algorithm 4, instead of issuing an IBS packet, the non-boundary QL sets a shrunken rolling-ball, marks itself as *visited*, and delivers the ball to its nearest neighbor by broadcasting an aggregation packet. Upon receiving the aggregation packet, each neighbor of the QL executes the steps described in Algorithm 6. Note that no initial boundary needs to be marked in the case of non-boundary QL in Algorithm 4. In fact, in this case, a new boundary will be

Algorithm 5 `receive_IBS_Packet()`

```
1: mark locally the currently traversed boundary as explored;
2: if (currentNode is not the QL){
3:    forward IBS_Packet to next_hop;
4: }
5: else {
6:    // current node is the QL
7:    if (previous_hop is not the expected node){
8:       // initial boundary not totally explored
9:       forward IBS_Packet to next_hop;
10:   }
11:   else {
12:      // initial boundary has been totally explored
13:      call Algorithm 7;
14:   }
15: }
```

Algorithm 6 `receiveAggregationPacket()`

```
1: if (currentNode is not destination){
2:    // overhear aggregation packet
3:    updates locally the status of source node according to the received packet
      (i.e., mark it as VISITED or LINKING_NODE);
4: }
5: else {
6:    // current node is the destination
7:    update locally the status of source node according to the received packet
      (i.e., mark it as VISITED or LINKING_NODE);
8:    aggregate_data(); // if visited for the first time
9:    checkRollingBall(); // Algorithm 3
10: }
```

created in the network. Therefore, during the aggregation process, any other encountered unexplored boundary is disjoint and must be opened by the rolling-ball owner.

We underline that due to the limited knowledge of nodes, and given the radius of the optimal rolling-ball ($R/2$, where R is the communication range of nodes), the owner of this latter cannot be aware of all nodes surrounding the ball, and hence, cannot locally detect the existence of cycles. However, this is not the case for the shrunken rolling-ball. In fact, because of its small size, its owner is well aware of all nodes that surround it, along with their corresponding statuses (visited or linking nodes).

Lemma 5:

Shrunken rolling-ball does not loop.

Proof. This lemma means that at each iteration, the shrunken rolling-ball finds a non-linking node N_k that can be removed from Ω. To prove this point, let us assume that the shrunken ball is owned by node N_i. Actually, given the fact that the shrunken radius "d" is smaller than the optimal one ($d < R/2$), the farthest distance between each pair of neighbors of N_i which surround the shrunken ball and prevent it from being optimal is smaller than the communication range of nodes R ($2 * d < R$, where $2 * d$ is the shrunken ball diameter). Consequently, we conclude that each pair of neighbors of N_i surrounding

Algorithm 7 `continueAggregationProcess()`

```
1: spin rolling_ball;
2: if (next_hop is undefined){
3:   // aggregation ends here; no other nodes to visit
4:   currentNode.status = VISITED;
5:   send result to QL; // using geographic routing
6:   return;
7: }
8:
9: if (not currentNode.isPortalNode()){
10:   // current node has no disjoint boundaries
11:   change status to VISITED or LINKING_NODE;
12:   sendAggregationPacket(next_hop);
13: }
14: else {
15:   // current node is a portal node
16:   create right and left sets;
17:   send_DBS_Packet(); // to mark disjoint boundary
18: }
```

the shrunken ball can directly communicate with each other. Hence, even if we suppose that a cycle can occur in this case, determining and removing that cycle can be done locally by each node without recourse to control packets. □

In the case of an optimal rolling-ball, the idea of avoiding looping around cycles consists of *opening* any encountered disjoint boundary by portal nodes. More specifically, as shown in Algorithm 7, in the case where the currently traversed node is not *portal* (i.e., has no disjoint boundaries), this node changes its status to *visited* or *linking* node, and forwards the aggregation packet. On the other hand, a *portal* node must execute the three following steps (upon receiving the aggregation packet) in order to break the cycle:

Step 1 - Creation of the left and right sets. The portal node N_i divides the set of its immediate neighbors into two subsets: left and right sets.

- **Left set** ($X \subset \Omega$): this set contains all neighbors located in the sector defined by the angle $\angle N_{\text{left}}, N_i, N_{\text{prev}}$ (see Fig. 4.8). Where N_{left} is the left side of the unexplored disjoint boundary, and N_{prev} is the previous hop from which node N_i has received the aggregation packet. In Fig. 4.8, all orange nodes belong to the left set X. Note that N_{prev} (if it has not been visited), as well as N_i, and N_{left} belong to X.
- **Right set** ($Y \subset \Omega$): this second set is composed of the rest of neighbors of node N_i including the right side of the unexplored boundary. In Fig. 4.8, all blue nodes belong to Y.

Step 2 - Disjoint boundary scan: after having created the left and right sets, the portal node issues a rolling-ball control packet named DBS (*Disjoint-Boundary Scan*) packet, which traverses the unexplored boundary node by node (see Fig. 4.9 and Algorithm 8). Similar to the IBS, DBS packet marks the disjoint boundary as explored and allows nodes to determine whether their local stored boundaries constitute the same boundary. For instance, in Fig. 4.9, DBS packet informs node N_x that the boundary defined by N_y as its

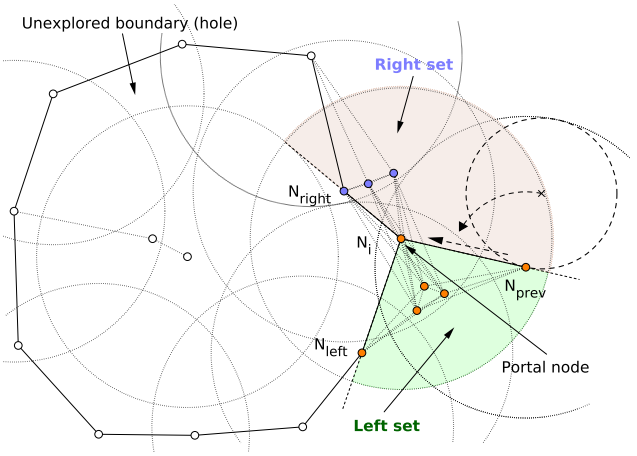


Figure 4.8: Left and right sets creation by portal node N_i.

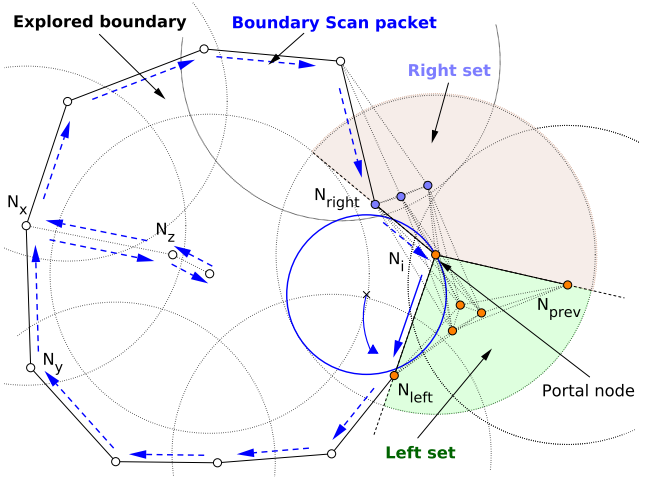


Figure 4.9: Disjoint boundary scan using DBS Packet issued by portal node N_i.

right side, and the one defined by N_z as its right side, are actually the same boundary. Hence, when node N_x later receives the aggregation packet, it will not consider these two boundaries as disjoint.

Step 3 - Cycle removal: after receiving the DBS packet back, the portal node (which belongs to the left set X) virtually deletes its links with every node of the right set Y (Algorithm 8). In other words, 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 $\mathcal{D} = \{L_{(i,\,j)} \mid N_j \in Y(N_i)\}$. Once the appropriate links have been removed, the portal node changes its status to *visited* or *linking* node, and broadcasts an LC (*Link-Cut*) packet to its immediate neighbors. To finish the cycle break process, each neighbor of the portal node performs the following steps upon receiving the LC packet (Algorithm 9). First, it determines the set to which it belongs (X or Y). Then, it virtually deletes its links with every node of the other set (whenever such a link exists). Fig. 4.10 gives an example of disjoint boundary opening (cycle removal). Once the cycle has been broken, the aggregation process continues from the left side of the opened boundary (i.e., N_{left}).

Property 6. *Portal nodes ensure cycles removal from* Ω.

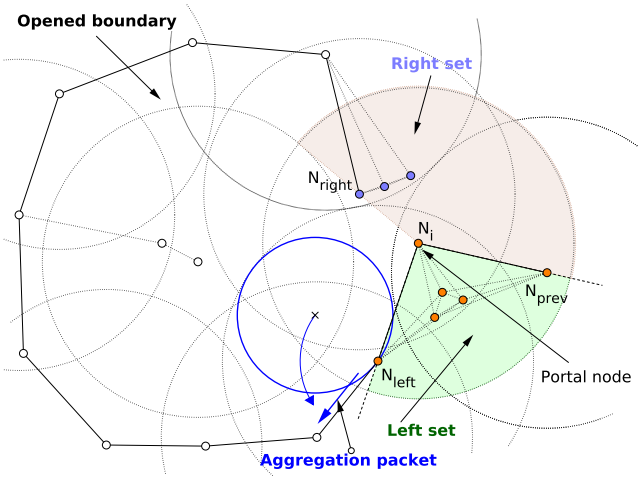


Figure 4.10: Cycle removal using LC packet broadcasted by portal node N_i.

Algorithm 8 `receive_DBS_Packet()`

```
1: locally mark the currently traversed boundary as explored;
2: if (currentNode is not the portal node){
3:    forward DBS_Packet to next_hop;
4: }
5: else {
6:    // current node is the portal node
7:    if (previous_hop is not the expected node){
8:       // disjoint boundary not totally explored
9:       forward DBS_Packet to next_hop;
10:   }
11:   else {
12:      // disjoint boundary has been totally explored
13:      cut links with nodes of right_set;
14:      change status to VISITED or LINKING_NODE;
15:      send_LC_Packet();
16:   }
17: }
```

Algorithm 9 `receive_LC_Packet()`

```
1: update locally the status of source node according to the received packet
   (i.e., mark it as VISITED or LINKING_NODE);
2: determine membership set; // right or left
3: cut links with nodes of the other set;
4: if (currentNode is destination){
5:    aggregate_data(); // if visited for the first time
6:    call Algorithm 7;
7: }
```

Proof. Let us assume that N_i, which is a portal node, has detected the existence of a cycle C_1 in Ω. Before removing C_1, initially thanks to the issued DBS packet we are sure that two paths exist connecting the left set X and the right set Y of C_1. The first path is comprised of direct communication links between the nodes of X and those of Y, while the second is composed of N_{left} and each node located on the disjoint boundary until N_{right} (i.e., nodes visited by DBS packet). After removing C_1 from Ω thanks to the cycle

removal process performed by portal node N_i and its corresponding neighbors (three steps described above), no direct communication links can exist between the left and right set of C_1. Consequently, the only path that remains connecting X with Y is the DBS packet path. In addition to that, it is worth mentioning that using the rolling-ball guarantees that portal node N_i and its previous-hop N_{prev} belong to the boundary of Ω. This formally means that: $\nexists N_j \mid L_{\{j,i\}} \notin \mathcal{L} \ \wedge \ L_{\{j,prev\}} \notin \mathcal{L} \ \wedge \ L_{\{j,k\}} \in \mathcal{L}$ such that $N_k \in V_i \ \cap \ V_{prev}$ (i.e., N_k belongs to the set of common neighbors of N_i and N_{prev}). Thus, by construction, there cannot exist a node N_j such as (1) N_j cannot communicate with N_i and N_{prev}, but (2) can communicate with N_k. □

Property 7. *Portal nodes do not partition Ω.*

Proof. This property can be proven by contradiction. First, let us assume that cutting the direct communication links between X and Y during cycle removal process disconnects Ω. Second, let us assume that only one direct communication link exists that connects X and Y, which is $L_{(i,\ right)}$ (the link between portal node N_i and right side of disjoint boundary N_{right}). In such a scenario, the first assumption means that there is no other path that connects X and Y except $L_{(i,\ right)}$, which thanks to the DBS packet sent beforehand contradicts the existence of another path connecting N_i with N_{right}. Given the fact that (1) nodes belonging to X or those belonging to Y are internally connected (thanks to Lemma 3 and the connectivity assumption made in Section 1.3) and (2) N_i and N_{right} belong respectively to X and Y, we deduce that X and Y can be connected thanks to the DBS packet path, and hence without requiring the deleted direct links between them. This contradicts the assumption that suggests that links cut, during cycle removal process, disconnects Ω. In addition to what has been said, thanks to the use of IBS and DBS packets, we have the guarantee that each disjoint boundary (cycle) has only one portal node. □

4.4/ PROOF OF CORRECTNESS

In this section, we prove the correctness of the proposed algorithm. More specifically, we prove that SA visits all connected nodes in the network, without falling into looping. Proving that SA visits all nodes is based on (1) the use of a proven boundary traversal algorithm and (2) the fact that the network is initially connected ($\Omega = \mathcal{N}$) and its connectivity is maintained throughout the data aggregation process (Lemmas 3 and 4, and Property 7). In contrast, the proof that SA does not loop and terminates is based on the fact that all possible cycles in Ω (which can be generated because of the use of linking nodes notion) are detected and properly removed (Lemmas 4 and 5, and Property 6).

Initially, all nodes in the network are marked as *unvisited*. Therefore, we have the set of unvisited nodes Ω equal to the set of nodes $\mathcal{N}$ and the set of visited nodes Γ is empty ($\Omega = \mathcal{N}$ and $\Gamma = \emptyset$). To accomplish data aggregation, SA iteratively visits nodes by gradually filling up Γ and emptying Ω. The traversal of the network can be expressed formally via the following recurrent formula:

$$\begin{cases} \Gamma_0 = \emptyset \ \ and \ \ \Omega_n = \mathcal{N}, \\ \Gamma_{i+1} = \Gamma_i + \{N_k\} \ \ and \ \ \Omega_{j-1} = \Omega_j - \{N_k\} : if \ \ i < n \ and \ j > 0 \end{cases}$$

where n is the number of nodes in the network and N_k is the first traversed non-linking node that can be removed from the current set of unvisited/linking nodes Ω_j without alter-

ing its connectivity. For example, in Fig. 4.4, initially, we have:

$\Gamma_0 = \emptyset$ and $\Omega_8 = \mathcal{N} = \{N_1, N_2, N_3, N_4, N_5, N_6, N_7, N_8\}$. The traversal of $\mathcal{N}$ is done through the following respective iterative filling of Γ_0 and emptying of Ω_8:

$\Gamma_1 = \Gamma_0 + \{N_2\}$ and $\Omega_7 = \Omega_8 - \{N_2\}$
$\Gamma_2 = \Gamma_1 + \{N_6\}$ and $\Omega_6 = \Omega_7 - \{N_6\}$
$\Gamma_3 = \Gamma_2 + \{N_8\}$ and $\Omega_5 = \Omega_6 - \{N_8\}$
$\Gamma_4 = \Gamma_3 + \{N_7\}$ and $\Omega_4 = \Omega_5 - \{N_7\}$
$\Gamma_5 = \Gamma_4 + \{N_5\}$ and $\Omega_3 = \Omega_4 - \{N_5\}$
$\Gamma_6 = \Gamma_5 + \{N_4\}$ and $\Omega_2 = \Omega_3 - \{N_4\}$
$\Gamma_7 = \Gamma_6 + \{N_3\}$ and $\Omega_1 = \Omega_2 - \{N_3\}$
Finally, $\Gamma_8 = \Gamma_7 + \{N_1\} = \mathcal{N}$ and $\Omega_0 = \Omega_1 - \{N_1\} = \emptyset$.

To prove that the proposed algorithm always generates, at its termination, a finite set of visited nodes Γ_n that is equal to the set of nodes in the network $\mathcal{N}$, and at the same time it also produces an empty set of unvisited nodes Ω_0, we will proceed in the following manner. First, we will consider the case of non-boundary QL. More precisely, we consider the case of shrunken rolling-ball. Next, we will move to the optimal rolling-ball, which of course includes the case of boundary-QL.

As mentioned before, we initially have a set of unvisited nodes Ω_n composed of all nodes of the network, and we have an empty set of visited nodes Γ_0. Thanks to Lemma 4, the non-boundary QL cannot, in any case, be a linking-node, and hence it is not necessary for the connectivity of Ω_n. Consequently, in all cases, regardless of its location in the network, the non-boundary QL can be removed from Ω_n and added to Γ_0:

$\Gamma_1 = \Gamma_0 + \{QL\}$ and $\Omega_{n-1} = \Omega_n - \{QL\}$

Given the fact that the non-boundary QL is always removed from Ω_n and added to Γ_0 ensures the main requirement of the rolling-ball (which is the permanent emptiness from any node) and guarantees a correct boundary traversal process. Once the non-boundary QL has been excluded from the traversal process (Algorithm 4), the rolling-ball will be delivered to its nearest neighbor. Hence, this latter will have a new boundary in which the received rolling-ball can be released. We emphasize that the ball given to the nearest neighbor of the non-boundary QL can be shrunken or optimal (Algorithms 6 and 3). As previously mentioned, we will consider only the first case. The optimal rolling-ball scenario will be treated later on in this section.

When the shrunken rolling-ball leaves the non-boundary QL, it moves in a localized fashion from node to node, and if possible, it becomes enlarged at each hop. We underline that the traversed nodes by the shrunken rolling-ball can be marked as visited or linking nodes, and as previously shown, the existence of linking nodes in Ω means the potential presence of cycles. Nonetheless, thanks to Lemma 5, we have the assurance that the shrunken rolling-ball does not generate looping situations.

Theorem 3:

After a certain number of iterations ($i \geq 1$), shrunken rolling-ball gains its shape and becomes optimal.

Proof. Let us suppose that while traversing the network with the use of a shrunken rolling-

ball, the latter cannot be enlarged. In such a case, the fact that the shrunken-ball (like the optimal one) traverses the boundary of Ω_{n-i} signifies that all nodes that have been explored are linking nodes that form a cycle. According to Lemma 5, this is not true; at each iteration, a non-linking node N_k exists. This means that, at each iteration, the number of nodes in Ω_j strictly decreases by one, and the number of nodes in Γ_i strictly increases by one:

$| \Omega_{j-1} | < | \Omega_j |$ and $| \Gamma_{i+1} | > | \Gamma_i |$ such as $j > 0$ and $i < n$.

Therefore, we deduce that, at each iteration, the shrunken-ball creates more space for itself by finding a non-linking node N_k that can be removed from Ω_j. Given the fact that (1) the shrunken-ball does not generate looping cases, and (2) it can be enlarged, in the end, after a certain number of iterations i, the shrunken-ball can gain its shape and becomes optimal. □

After having treated the case of shrunken rolling-ball, and proved that it does not loop and produces a finite set of hops, we move to the case of optimal-ball to prove that it also does not loop. Next, we will prove that SA ensures the exploration of all nodes. But before this, it is worth mentioning that the shrunken rolling-ball, like the optimal one, ensures boundary traversal and does not miss any non-linking node in its way. The formal proofs demonstrating that the rolling-ball ensures boundary traversal can be found in [41].

Lemma 6:

Optimal rolling-ball does not loop.

Proof. To prove this by contradiction, let us assume that at some point during network traversal, the optimal-ball cannot find a non-linking node N_k. This means that the rolling-ball has become stuck between at least two disjoint boundaries, and is jumping (looping) from a linking node to another inside one of the boundaries. This assumption can be simply denied because it contradicts the fact that portal nodes guarantee a proper detection/removal of cycles (disjoint boundaries) from Ω_{n-i}. More precisely, according to Property 6, we deduce that, at each iteration i, the optimal-ball finds a non-linking node N_k. That is, after each iteration, the number of nodes in Ω_{n-i} will be strictly decreased by one, and the number of nodes in Γ_i will be strictly increased by one:

$| \Omega_{n-(i+1)} | < | \Omega_{n-i} |$ and $| \Gamma_{i+1} | > | \Gamma_i |$ such as $i < n$. □

Theorem 4:

SA does not loop.

Proof. First, thanks to Lemma 5 and Theorem 3, we have the assurance that the shrunken-ball does not loop. Actually, regardless of the considered scenario, the shrunken-ball progressively gets enlarged and will eventually gain its optimal form. Second, if we consider the rolling-ball in its optimal shape, then, according to Lemma 6, we have the guarantee that it does not loop and at each iteration, it finds a non-linking node. Based on what has been said and given the fact that in SA, the ball can be either shrunken or optimal, and its radius can never be larger than $R/2$, we conclude that SA does not loop and generates a finite set of hops. □

Lemma 7:

At each iteration $i \geq 1$, the currently visited node N_k belongs to the boundary of Ω_{n-i}.

Proof. In the case of a shrunken-ball, the non-boundary QL (located inside the ball) cannot be in any case a linking node (Lemma 4), hence creating a boundary for its nearest neighbor to exploit (launch the ball inside it). Therefore, in both cases (optimal and shrunken-ball), the algorithm is triggered by a boundary node. Based on this and given the fact that the rolling-ball is a boundary traversal tool [41], each visited non-linking node N_k will be certainly located on the boundary of Ω_{n-i}. To prove this, let us assume that this property is true for i and let us demonstrate that it remains as such for $i+1$. First, saying that the property is true for i means that the currently traversed node N_k is located on the boundary of Ω_{n-i}. Second, as previously mentioned, the used rolling-ball boundary traversal algorithm guarantees that the next hop N_{succ} of N_k is located on the boundary of Ω_{n-i} [41]. Consequently, it is clear that this next hop N_{succ} is also located on the boundary of $\Omega_{n-(i+1)}$. More precisely, if N_{succ} is not a linking node (i.e., can be marked as visited) then:

$$\Gamma_{i+1} = \Gamma_i + \{N_{succ}\} \quad \text{and} \quad \Omega_{n-(i+1)} = \Omega_{n-i} - \{N_{succ}\}$$

Otherwise, starting from N_{succ}, the rolling-ball has to continue its boundary browsing of $\Omega_{n-(i+1)}$ until it finds a non-linking node that can be marked as visited. Hence, we conclude that the property is also true for $i+1$. □

Theorem 5:

SA visits all nodes in the network.

Proof. First, thanks to the assumption made in Section 1.3, we have the guarantee that the initial network is connected ($\Omega_n = \mathcal{N}$). Second, thanks to Lemmas 3 and 4, we have the guarantee that for both shrunken and optimal-ball, the notion of linking-nodes ensures the connectivity of Ω_j. Third, according to Property 7, we know that the cycle removal process (which takes place only in the case of optimal-ball) does not partition Ω_j. To prove that SA visits all nodes in the network, let us assume that the set Γ_i generated at the end of network traversal is different than the set of nodes $\mathcal{N}$ (i.e., $i \neq n$) and then let us seek to find a contradiction. First, let us denote by N_k the first traversed non-linking node that can be removed from Ω_j without altering its connectivity (Lemmas 3 and 4, and Property 7). Second, let us denote by N_m, one of the unvisited nodes that might be missed by SA (i.e., $N_m \neq N_k$, $\forall\, k$). Third, let us consider an unsigned integer j ($0 < j < n$) such that N_m belongs to Ω_j and does not belong to Ω_{j-1}. That is, N_m has been missed at iteration j. In actual fact, since at iteration j, the rolling-ball touches only nodes that are located on the boundary of Ω_j and does not miss any one of them (Lemma 7 and Theorem 4), N_m cannot, in any case, be located on the boundary of Ω_j. In other words, given the fact that the use of the rolling-ball guarantees that there cannot exist a node N_m that can be missed, N_m must surely belong to Ω_{j-1}. However, this contradicts the initial assumption, which says that N_m does not belong to Ω_{j-1} ($N_m \notin \Omega_{j-1}$). Consequently, we conclude that SA guarantees network traversal completeness in spite of the considered topology. □

To recapitulate, according to Theorems 4 and 5 and regardless of the network topology (irregular or regular, large-scale or sparse, with or without holes, etc.), we conclude that SA ensures query completeness/accuracy and terminates without looping indefinitely.

4.5/ SPREADING PERFORMANCE ASSESSMENT

As it has been previously mentioned, in this manuscript, OMNeT++ and its WSNs framework Castalia [53, 54] have been opted for as a performance evaluation tool. This choice is motivated, among others, by the fact that Castalia offers a realistic wireless channel (e.g., interference, mobility of nodes, path loss, etc.) and radio models (e.g., realistic modeling of RSSI and carrier sensing, probability of reception based on SINR, packet size, modulation type, etc.), and also a realistic node behavior, especially pertaining to radio access. Besides the proposed spreading approach, the following aggregation techniques have been implemented: tree-based aggregation [3], Depth-First Search [26], Peeling Algorithm [18], and Greedy-Boundary Traversal [25]. We note that all the considered serial techniques have been formally proven to ensure complete exploration of nodes, regardless of the network topology.

4.5.1/ EVALUATION METRICS

In addition to the completeness criterion, the three other metrics with which all parallel and serial approaches cited above have been evaluated are energy, time and communication efficiency.

- ***Energy:*** this metric, which represents the most precious resource in WSNs, measures the total energy spent by the network during data aggregation. To allow each node to estimate the energy required to transmit and receive packets, the model proposed by Heinzelman et al. [55] has been considered. In this energy consumption model, in order to send a packet of a k-bit size a certain distance x, the radio consumes $E_{TX}(k,\ x) = E_{\text{elec}} * k + \epsilon_{\text{amp}} * k * x^2$. And to receive this same packet, it consumes $E_{RX}(k) = E_{\text{elec}} * k$. Where $\mathrm{E_{elec}} = 50\ \mathrm{nJ/bit}$ is the energy required to run the transmitter/receiver circuitry, and $\epsilon_{\text{amp}} = 100\ \mathrm{pJ/bit/m^2}$ is the energy consumed by the transmitter amplifier.
- ***Time:*** this evaluation metric measures the time required to aggregate all the data present in the network. That is, the time that passes between the moment data aggregation is triggered and the moment the desired result is delivered to the sink/QL. For instance, in the conducted simulations, nodes have been ordered to find the average temperature of the whole network.
- ***Transmissions:*** this last metric represents the total number of packets issued by the network in order to aggregate data (including control and data packets).

4.5.2/ SIMULATION PARAMETERS

To evaluate the scalability of SA and the other considered techniques, simulations have been conducted in different network size and density scenarios. More specifically, the de-

Table 4.1: Spreading simulation configuration and parameters

Parameter	**Value(s)**
Field dimensions (m^2)	1000 x 1000
Nodes number	100, 150, 200, . . . , 500
Nodes deployment	Uniform
Nodes transmission range (m)	150
Nodes initial energy (J)	100
Query launcher location	Random
Query packet size ($byte$)	50
Radio type	CC2420
Radio data rate ($kbps$)	250
MAC protocol	CSMA/CA

Table 4.2: Spreading average nodes' degree.

100	**150**	**200**	**250**	**300**	**350**	**400**	**450**	**500**
6	9	12	15	18	21	25	27	31

ployment field has been fixed to 1000 x 1000 meters throughout simulations. As regards the network, initially, 100 sensor nodes have been randomly deployed, and then, 50 other sensor nodes were progressively added, to a maximum of 500 nodes. Table 4.1 summarizes the simulation parameters and Table 4.2 shows the average degree of nodes (i.e., the average number of neighbors of each node). To be more specific, the first line of Table 4.2 depicts the number of nodes in the network, whereas the second one represents the corresponding average degree.

4.5.3/ SIMULATION RESULTS

In this section, the obtained simulation results are compared and evaluated. Actually, for the sake of clarity, and given the obvious difference between the results of serial and parallel approaches (particularly in terms of energy consumption), we first start by plotting and comparing the results of our spreading algorithm with those of the implemented tree-based approach [3]. Next, we will plot the same results of our approach and compare them with the implemented serial techniques, namely: Depth-First Search [26], Peeling Algorithm [18], and Greedy-Boundary Traversal [25]. We mention that in both cases, all results, whether those relating to required transmissions, time or energy, are plotted against the number of nodes specified in Table 4.1.

4.5.3.1/ SPREADING ALGORITHM VERSUS TREE-BASED AGGREGATION

Fig. 4.11 plots the number of packets that our proposed serial technique of Spreading Aggregation (SA), and the tree-based approach [3] (denoted here as T-BA), have both issued to accomplish data aggregation. Concerning Fig. 4.12 and Fig. 4.13, they respectively plot the time and energy that each of these two approaches has taken in order to aggregate data and deliver the response to the sink.

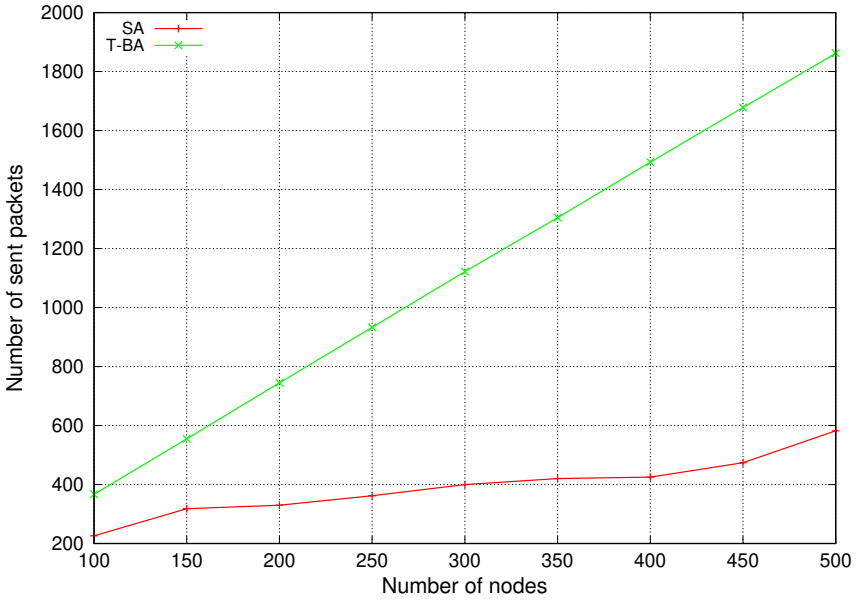


Figure 4.11: Transmissions required by Spreading and Tree-based approaches to aggregate data.

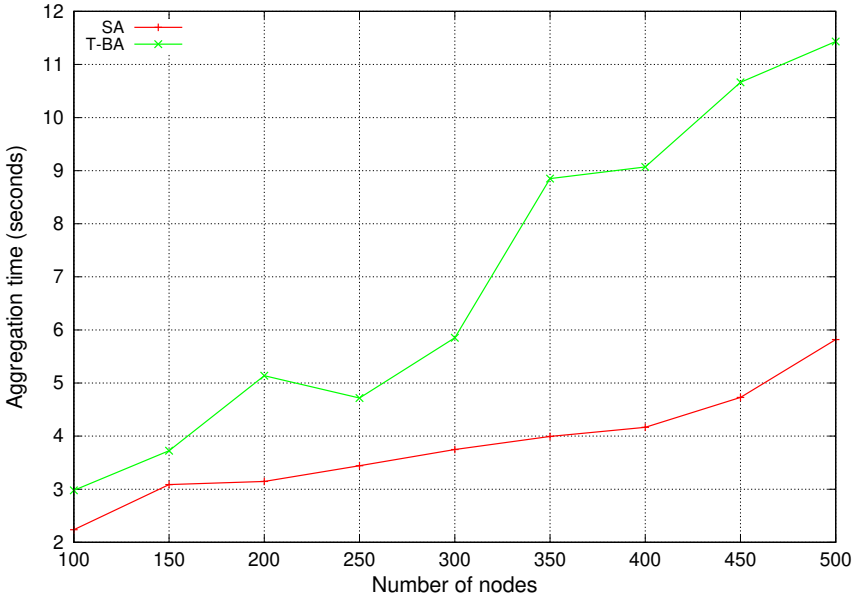


Figure 4.12: Time required by Spreading and Tree-based approaches to aggregate data.

In actual fact, for both approaches (SA and T-BA), the considered transmissions include (1) transmissions used to construct the tree or respectively traverse the network, (2) transmissions used to query the network, and (3) transmissions used to deliver the processed data to the sink. As previously mentioned, structure construction, query diffusion and data processing in tree-based approaches are three independent phases. First, the network must be properly covered with a spanning tree (to ensure query accuracy all nodes must be attached to the tree). Then, after being queried via the constructed tree, sensor nodes start data processing and response delivery to the sink/root. Conversely, in our serial approach, these three phases are all merged into one unique phase. At the same time,

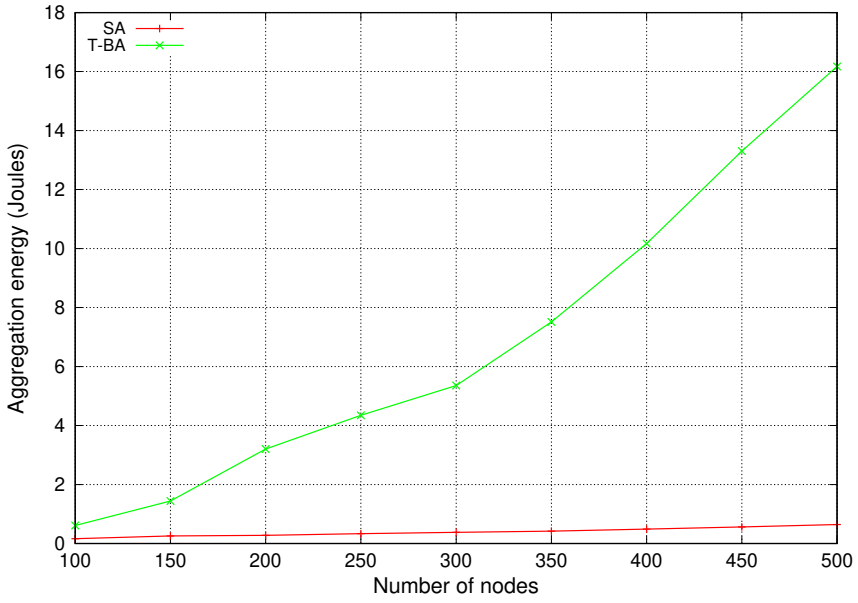


Figure 4.13: Energy required by Spreading and Tree-based approaches to aggregate data.

while gradually traversing the network, sensor nodes are queried and the proper treatment is applied to data. This compact performance considerably minimizes the number of packets that need to be issued, conserves energy, and improves processing time.

As Fig. 4.13 confirms, there is a big gap between T-BA and SA in terms of energy consumption. Actually, given their parallel and non-collision-free nature, tree-based approaches do not consume energy only to construct/maintain the tree, disseminate queries and report the processed data to the sink, but also to re-transmit collided packets. With respect to aggregation time, due to their parallel concurrent operation, tree-based approaches should defeat serial algorithms, even with a large number of issued packets. Nonetheless, as Fig. 4.12 demonstrates, because of collisions essentially among other reasons, this is not the case; in fact, SA presents better results than T-BA in terms of aggregation time reduction.

To illustrate the difference between serial and tree-based approaches in terms of required transmissions, let us consider a sensor network composed of n wireless nodes. If we assume that each node in tree-based approaches is involved in the query diffusion phase (i.e., it forwards the received query), this means that n packets will be broadcast. On the other hand, during the data or query processing phase, n additional packets will be sent, because obviously during this phase each node is obliged to forward a packet. Therefore, without taking into account the packets used to construct or probably maintain the tree, and without counting the retransmissions of collided packets, we can say that in total, $2*n$ packets need to be issued to accomplish data aggregation in tree-based approaches. We emphasize that building and fixing a distributed structure in a wireless environment that is prone to errors and interference, is not a straightforward task, and non-negligible time and overhead must certainly be spent to accomplish this process. Regarding serial approaches, if we consider the same network composed of n wireless nodes, and if we assume that these approaches can lay out an itinerary that crosses each node once at any given moment in time, then, without generating any collisions or requiring a separate

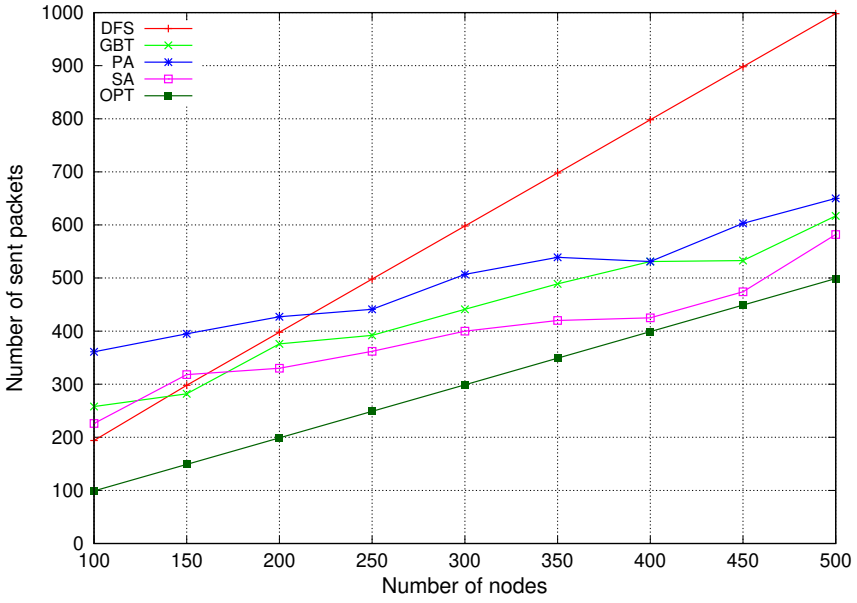


Figure 4.14: Spreading Aggregation versus serial approaches. Comparison in terms of required transmissions for data aggregation.

structure construction phase or maintenance, only $n - 1$ packets need to be issued to diffuse a query and process data.

To recapitulate, besides their bad scalability, their inadequacy for dense large-scale networks, and their robustness and imbalance issues, tree-based approaches suffer primarily from collisions. First, due to their parallel concurrent mode of operation, tree-based approaches undergo a lot of communication collisions, especially in dense deployments. The retransmissions of collided useless packets waste a noticeable amount of energy and time, and negatively affect these approaches. Second, apart from collisions, tree-based approaches also suffer from the problem of unbalanced energy consumption. This issue takes place due to the creation of an unbalanced spanning tree in the first place, or because sensor nodes that are located near the sink/root are excessively used to pass the traffic [12]. Third, a distributed tree is a set of pre-established itineraries that are fragile and very sensitive to node and link failures. Therefore, the implementation of a tree maintenance operation is a requirement that can fix the problem from one side, but can be very expensive in terms of energy and time from the other side, especially in dense large-scale networks.

4.5.3.2/ SPREADING ALGORITHM VERSUS SERIAL APPROACHES

Fig. 4.14 shows the number of packets issued to accomplish data aggregation by the proposed approach, Depth-First Search (DFS) [26], Peeling Algorithm (PA) [18], and Greedy-Boundary Traversal (GBT) [25]. We mention that for all four serial approaches, the counted transmissions include path construction, query diffusion, and result delivery to the sink. Fig. 4.14 also plots the number of packets that an optimal theoretical serial algorithm (denoted as OPT) would have issued to aggregate data if it existed. Actually, the number of packets required by both DFS and the optimal theoretical algorithm OPT have been plotted to measure the effectiveness of the evaluated serial algorithms in terms of

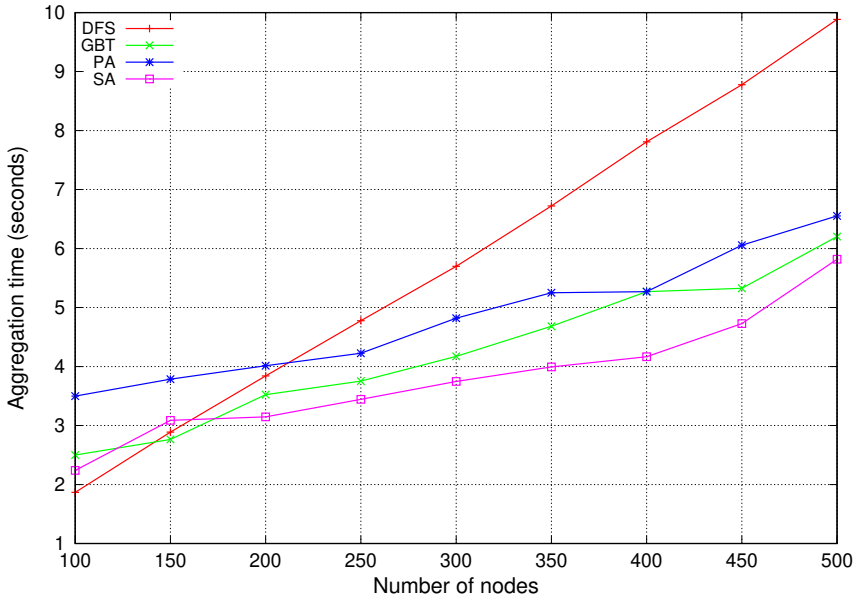


Figure 4.15: Spreading Aggregation versus serial approaches. Comparison in terms of required time for data aggregation.

transmissions. The curve of a good serial approach must be located between those of OPT and DFS.

We recall that the optimal algorithm OPT necessitates $n-1$ packets to traverse and aggregate data from a network of n connected nodes. Nevertheless, as previously mentioned, such an optimal Hamiltonian path does not exist in every network; even if it does, finding this path constitutes an NP-complete problem [20]. DFS, because of its backtracking nature, requires double the number of OPT, i.e., $2*(n-1)$ packets to aggregate and deliver the result to the sink. The number of transmissions required by the other serial approaches (i.e., SA, PA, and GBT) cannot be expressed using a mathematical expression.

As demonstrated by Fig. 4.14, SA outperforms all the other serial approaches and necessitates a number of transmissions that is closer to the optimal number of OPT. However, in sparse low-density networks, DFS slightly outperforms SA. In reality, this is due to the fact that in low-density deployments, it is more likely that the network contains more disjoint boundaries (i.e., more cycles). In such a scenario, the bad performance of SA stems from the excessive use of DBS packets sent to scan the disjoint boundaries. So, the presence of disjoint boundaries in the network results in a greater expenditure of control packets by SA. But, in contrast, a denser network will contain fewer holes and disjoint boundaries, and will improve the performance of SA. Note that when the number of nodes increases, SA approximates the optimal number of transmissions, and the difference between SA and DFS becomes more and more noticeable. For instance, in the case of 500 nodes, the improvement is over 40%.

Fig. 4.15 and Fig. 4.16 respectively show the time and energy required to aggregate data by all the considered serial approaches. If these latter are compared together in terms of data delivery time, one can say that, due to their sequent behavior, a serial data aggregation algorithm will need more time to finish when it necessitates a larger number of transmissions, and vice versa (provided that the issued packets have approximate or identical sizes). As depicted in Fig. 4.15, since SA necessitates the smallest number of

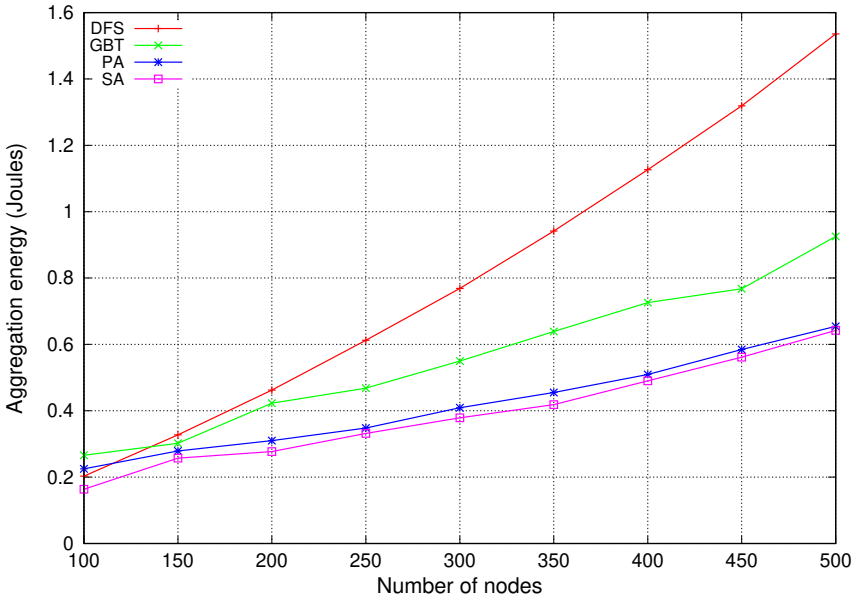


Figure 4.16: Spreading Aggregation versus serial approaches. Comparison in terms of required energy for data aggregation.

transmissions, it performs better than all the other serial algorithms and aggregates data more rapidly.

As shown in Fig. 4.16, SA also performs better than the other serial approaches in terms of energy consumption. Actually, in WSNs, energy is consumed by the various tasks carried out by nodes (chiefly, sensing, computing, and communicating). So, if one considers simple sensing systems and plain aggregation queries, communications dominate and become the most energy consuming task [1], and thus, the energy consumed by serial algorithms will be mainly related to the number of issued transmissions. That is, the more a serial algorithm requires packets, the more it will consume energy (provided that the issued packets have identical or approximate sizes). Since SA necessitates the lowest number of packets compared with the other serial algorithms, it is superior from the point of view of energy consumption.

4.6/ CONCLUSION

In this chapter, we presented a new serial algorithm called *Spreading Aggregation* (SA). The attractiveness of this in-network data processing technique comes from several factors. First, SA is maintenance-free and collision-free; hence, it does not require any transmissions in these regards. Second, SA is a path-free localized approach that relies only on the limited knowledge of each node to progressively traverse the network. In other terms, for each query, a new path will be laid out, which decreases the vulnerability to topology changes and link/node failures. Third, as in any other serial approach, SA fuses path construction, query dissemination, and data processing. That is to say; while crossing the network, sensor nodes are simultaneously queried, and their answers are collected. The conducted OMNeT++ simulations confirm that SA is scalable and very efficient in terms of energy conservation and response time reduction. The simulation results also assert that SA ensures aggregation completeness, and therefore query ac-

curacy. In addition to simulations, the proposed approach has been theoretically proven to visit all connected nodes in the network and to terminate without looping indefinitely.

Despite the efficiency of the proposed approach compared with tree-based aggregation and state-of-the-art serial algorithms, some other issues still remain to be addressed, such as robustness against node/link failures, imprecision of nodes' locations, etc. In addition to this, the performance of the proposed approach can be further enhanced by eliminating the use of control packets (i.e., IBS and DBS packets). This will be the object of the next chapter.

5 GEOMETRIC SERIAL SEARCH

5.1/ INTRODUCTION

Considering large WSNs and comparing data aggregation approaches in terms of scalability, robustness, completeness, and time/energy effectiveness, recent research has asserted the superiority of serial structure-free approaches over serial structure-based ones, and over both parallel structure-free and parallel structure-based techniques [18, 25, 39, 40]. But, in spite of the fact that serial structure-free approaches excel in large and medium-scale networks, their underlying path-construction algorithms are not optimal and can be improved by reducing the involved communications and further shortening the visiting path. To respond to this need, this chapter presents *Geometric Serial Search* (GSS); a new serial structure-free algorithm specifically designed to efficiently gather information from large wireless resource-constrained networks [42]. In addition to its completeness (i.e., visiting all nodes), collision-free nature, high scalability, energy/time efficiency, and robustness against topology changes (failures in links/nodes, ...), the main advantage distinguishing GSS is that it considerably reduces communications and always approaches the optimal number of hops (i.e., $n-1$). More precisely, in GSS, no control packets or complex data structures are required, instead, one packet hops from node to node and explores the entire network. While gradually finding its way through the network, this packet interrogates nodes and collects their responses at the same time. The followed path is not established in advance, can stem from any node in the network, and requires only the one-hop neighborhood information of each traversed node to be gradually drawn. The obtained OMNeT++ simulation results presented in this chapter demonstrate the efficiency of GSS and confirm all the previously cited claims.

This chapter is sectioned into four parts, as follows. Section 5.2 introduces the necessary preliminaries required throughout the chapter (to understand the considered problem as well as its solving). Section 5.3 details the proposed solution (i.e., GSS) through algorithms and simple illustrative figures. Section 5.4 depicts and interprets the obtained OMNeT++ simulation results. In the end, Section 5.5 concludes the chapter.

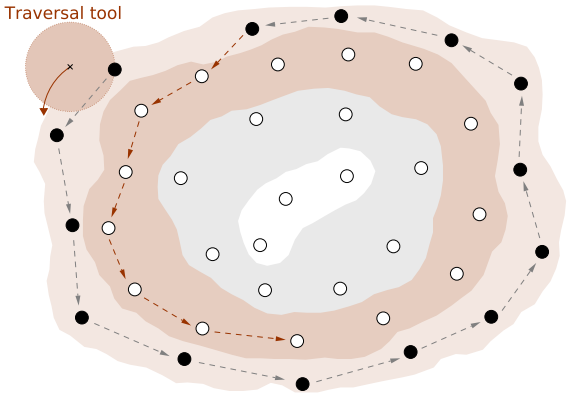


Figure 5.1: Boundary-first network traversal.

5.2/ BACKGROUND AND GENERAL IDEA

In brief, as demonstrated in Fig. 5.1, in order to efficiently traverse any randomly deployed wireless network, our approach sees it as a set of layers (boundaries). Once all nodes of the initial boundary have been explored, the latter (i.e., initial boundary) will be supplanted by a new boundary, and the same process will be reapplied to this new boundary and so forth. Thus, the network will be traversed layer by layer (boundary by boundary). In the end, the estimated parameter (query response) owned by the last node in the path can be sent to the query launcher via any desired efficient geographic routing.

Similar to SA, the whole operation of our third proposal is based on a geometric shape called *rolling-ball* (or *rolling-disc*) [41], which is simply a virtual circle that is hinged at a node and must, all the time, be empty of any other nodes. More details about this geometric concept can be found in Section 2.3.2. We mention that, aside from the rolling-ball, any other efficient boundary traversal tool can be utilized in our approach, provided that it adheres to the following. In order to ensure completeness and scalability, the utilized boundary traversal tool must (1) not miss any boundary node in its path, (2) keep the traversal process on the boundary of unvisited nodes, and (3) operate while relying only on the local information available at each node. Our choice has been specifically made on the rolling ball because in addition to the fact that this traversal tool is localized[1] and guarantees the visit of all boundary nodes in its path [41], it, actually, can visit more nodes than the curved-stick [38]. As a matter of fact, while this can be seen as a drawback in the case of anti-void routing (i.e., when data is routed around communication holes), it is in fact an advantage in the context of query processing (and data fusion), because the main objective in those circumstances is to query nodes and get their responses as quickly as possible.

Although the general idea of GSS is somehow similar to that of the previously proposed approaches (i.e., PA and SA), its uniqueness and outperformance come from its time and energy efficient technique with which it keeps the network connected while gradually traversing it using the rolling-ball (Section 5.3.2). In addition to the difficulty of ensuring the exploration of all nodes while maintaining the connectivity and reducing the required energy/time, the other encountered problem in our approach was how to begin query processing from an internal non-boundary node (i.e., a node that is unable of holding an

[1] Other than the one-hop neighbors table of its owner, it does not require any other information to move to the next node.

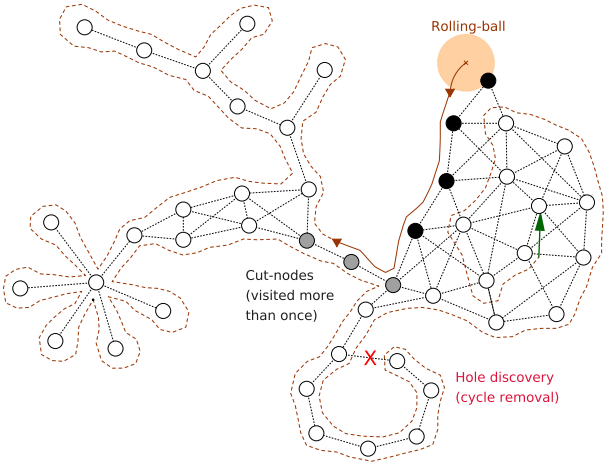


Figure 5.2: Optimal rolling-ball network traversal.

empty valid rolling-ball)? As shown later in this chapter (Section 5.3.1), we have proposed an efficient distributed technique that is able to effectively deal with this difficulty.

5.3/ PROPOSED APPROACH: GEOMETRIC SERIAL SEARCH

In all our three proposed query processing techniques, initially, for any query Q_j to be issued by node N_i, all nodes are considered as unmarked (did not contribute to Q_j yet). Formally speaking, the initial set of unmarked nodes for Q_j is equal to the set of all nodes in the network and its initial set of marked nodes is empty:

$$\Omega_n(i,j) = \mathcal{N} \quad \wedge \quad \Gamma_0(i,j) = \phi.$$

In fact, similar to SA, in GSS, in order to trigger network querying, node N_i marks itself and launches a rolling-ball:

$$\Omega_{n-1}(i,j) = \Omega_n(i,j) - \{N_i\} \quad \wedge \quad \Gamma_1(i,j) = \Gamma_0(i,j) + \{N_i\}$$

The launched ball rolls in a localized fashion over nodes and marks each one of them (Fig. 5.2). The details of the difference between GSS and SA (and which one is the most effective) will be revealed throughout this chapter.

Note that, here also, as it has been already mentioned in the previous chapter, in order to autonomously determine the ball's next-hop, each traversed node must be aware of the status of its neighbors (marked or not). To do so, when the rolling-ball is delivered (broadcast) by the current node, all of its neighbors can "hear" this notification and can thus update the status of their neighbors accordingly. This interesting feature of wireless communications preserves the localized nature of GSS and does not incur any overhead. Note also that the existence of one query in the entire network does not require any elaborated MAC layer. However, since nodes are obliged to overhear the communications of each other, a low power listening (LPL) MAC protocol must be considered [56, 57].

In GSS, in the end, once the rolling-ball has successfully explored the whole network, the last visited node forwards the obtained response to the query launcher via any efficient

geographic routing. We point out that the ball stops rolling when the currently traversed node has no unmarked neighbors. In other words, the traversal's ending point is a node of which all the neighbors have been marked (Fig. 5.2).

In the following, we will provide details of our third approach. First, in Section 5.3.1, we show how GSS is able to launch queries from non-boundary nodes (i.e., nodes unable of holding a valid empty rolling-ball). Second, in Section 5.3.2, we present the new efficient solution we propose to preserve the connectivity during network search. Third, In Section 5.3.3, we summarize the operation of GSS. Finally, in Section 5.3.4, we explain the mechanism we propose to detect and remove the looping situations created by the (localized) connectivity maintenance technique proposed in Section 5.3.2.

5.3.1/ INTERNALITY OF NODES

The fact that from one side the query launcher can be any node, and from the other side, the rolling-ball must be initially empty[2] [41] means that nodes can be divided into two categories: *internal* and *external*.

Definition 15: Internal and external Nodes

Given a set of wireless nodes with R as their communication range, a node N_i is said to be external (resp. internal), if at least one rolling-ball (resp. no rolling-ball) with radius $R/2$ can be attached to it.

In simple words, external nodes can hold a rolling-ball (and hence can launch queries), while internal ones cannot. To solve this issue and give internal nodes the ability to launch queries, we propose a straightforward technique that consists of finding another starting point for Q_j. The objective is to rapidly find the closest external node to the internal query launcher N_i without compromising the localizability (fewer transmissions means less time/energy consumption). Several techniques can be utilized (see Sections 3.3.3 and 4.3.3). In this chapter, we propose a new approach that aims the external network boundary to find an external node (potential starting point for Q_j). To explain this point, let us suppose that all nodes are aware of the fact that their deployment area is a rectangle defined by two points. In this scenario, to find the shortest path to the external boundary, internal node N_i starts by determining its closest perpendicular projection (Fig. 5.3). Once found, node N_i sends an initialization packet to the nearest neighbor to this point. Upon receiving this packet, each node N_j checks whether it is external or internal, and acts accordingly (Algorithm 10). Note that the initialization packet stops traveling towards the external network boundary as soon as it finds an external node (e.g., communication hole depicted in Fig. 5.3).

5.3.2/ DISCONNECTIVITY OF UNMARKED NODES

As it has been previously demonstrated in Chapter 4, combining the rolling-ball traversal with the concept of excluding (marking) each visited node is not an effective solution. Indeed, when cut-nodes mark themselves and leave the traversal, they lead to the disconnectivity of $\Omega_k(i, j)$ and thus to the incompleteness (inaccuracy) of Q_j. For instance,

[2] In order to guarantee a correct boundary detection, the rolling-ball must be empty of active nodes both initially and all the time.

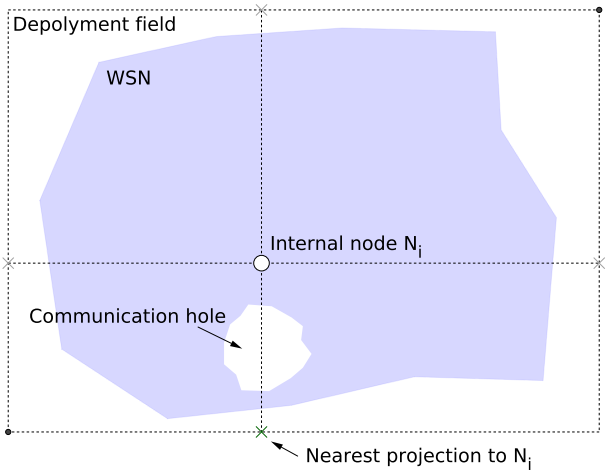


Figure 5.3: Query launch by internal non-boundary nodes.

Algorithm 10 GSS Setup

To launch a query, node N_i executes the following if-else statement:
if (`currentNode.isExternal()`) **then**
  `call Algorithm 11;`
else
  // currentNode is internal
  `find nearest projection P` (Fig. 5.3);
  `find nearest neighbor` N_j `to P;`
  `sendInitPacket(`N_j`);`
end if

`Upon receiving initPacket, node` N_j `calls Algorithm 10;`

in Fig. 5.4, once N_3 is marked, it leads to data aggregation stopping at N_4 (the unique neighbor of N_4, which is N_3, has been removed).

To avoid the partitioning of Ω and eliminate any early termination of queries, in addition to the concept of *unmarked* and *marked* nodes, two new states must be introduced, namely the *potential* and *actual-cuts*. Whilst marked nodes are inactive and do not collaborate in any way, the rest of the nodes (i.e., unmarked, potential and actual-cuts) are seen as active. To facilitate the understanding, in the following, we begin by defining actual-cuts and explaining the reason behind their use. Later on, we will explain what are potential-cuts and what is their role.

Definition 16: Actual-cut node (ACN)

An actual-cut N_i in a connected network is a node whose removal (marking) disconnects the set of all alive active nodes $\Omega_k(i, j)$ into two or more sub-sets.

For example, node N_{10} of Fig. 5.5 is an actual-cut that must remain involved in the process given its importance for the connectivity of Ω. Once no longer needed (i.e., after the visit of N_9 or N_{11}), this actual-cut marks itself to indicate its exclusion. The rolling-ball traversal combined with the concept of actual-cuts is sufficient to traverse any connected wireless network. Nevertheless, the main issue faced here is *determining the actual-cuts*. As a matter of fact, actual-cuts identification is not a straightforward operation, especially while

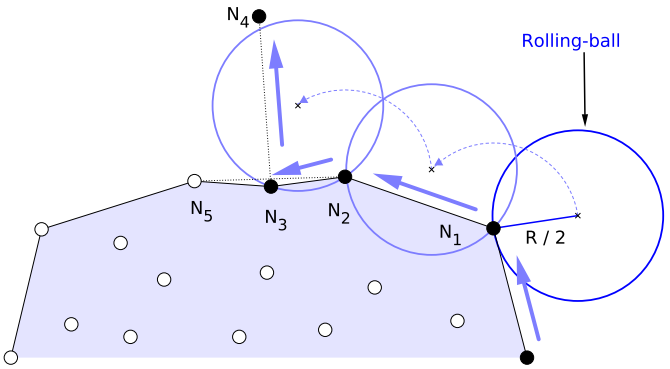


Figure 5.4: Disconnectivity example.

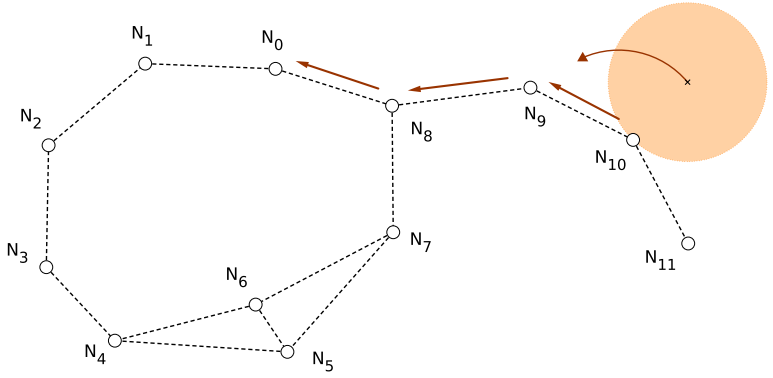


Figure 5.5: Example of actual-cuts. Considering node N_{10} as QL, nodes N_7, N_8, N_9, and N_{10} are actual-cuts, while the others are not (they can leave the traversal process).

trying to preserve the localizability and save both energy and time. In other words, based only on its one-hop neighbors' information, a node N_i cannot apply Definition 16 and say whether it is an actual-cut or not (Fig. 5.6). An intuitive way out of this dilemma is nodes collaboration. This technique, however, requires extra packets, time and energy. The solution we propose in this chapter does not require any overhead and actually depends only on the rolling ball. In addition to its initial traversal role, the latter contributes to the process of actual-cuts identification as a probing packet. Before getting into the details, let us first explain the notion of *potential-cuts* which is the building block of the proposed solution.

Definition 17: Potential-cut node (PCN)

A potential-cut N_i in a connected network is a node whose removal (marking) disconnects the set of its alive active one-hop neighbors $\mathcal{W}_i$ into two or more sub-sets.

A potential-cut is simply the unique possible connection between at least two of its active direct neighbors. For example, node N_{10} of Fig. 5.5 is a potential-cut, as well as node N_3 in Fig.5.4, 5.5 and 5.6. Note that each node N_i can locally[3] decide whether or not is a potential-cut. As regards the relationship PCN-ACN, an ACN is perforce a PCN, but the opposite is not true (i.e., a PCN is not necessarily an ACN).

[3]Based on its $\mathcal{W}_i \subset \mathcal{V}_i$.

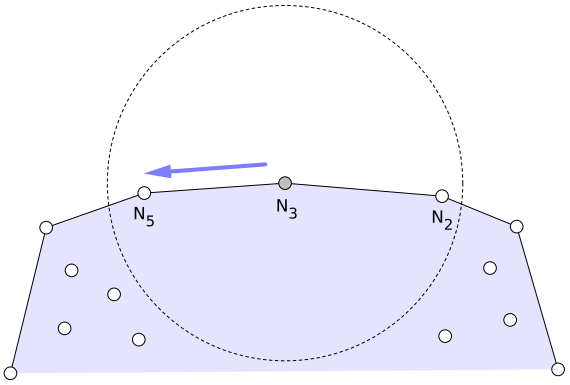


Figure 5.6: Example of potential-cuts. Exactly, as depicted in this figure, N_3 is a potential-cut that has no idea about the overall network topology and hence it does not know if it is an actual-cut.

We remind that our aim is an SSF approach that excludes control packets and counts exclusively on the rolling-ball and the $\mathcal{V}_i$ of each traversed node. To this end, as previously mentioned, the solution we propose to determine the actual-cuts in Ω_k assigns a twofold role to the rolling ball. We describe it briefly: upon receiving the rolling-ball, if node N_i is a potential-cut (Definition 17), it changes its status as so and forwards the rolling-ball as usual. Being given the boundary traversal of the rolling-ball, it will surely return to node N_i. Once it got the rolling-ball back, N_i does not contribute to Q_j but chooses to whether: (1) mark itself and leave the traversal, (2) declare itself as an actual-cut, or (3) stay as a potential-cut. For instance, in Fig. 5.6, based on which side the rolling-ball has returned to N_3, this node can properly determine its new status. Concretely speaking, if the ball returns from the same side (i.e., N_5), N_3 is an actual-cut; there is no other path connecting its neighbors. Otherwise, N_3 is not an actual-cut.

5.3.3/ GSS OVERVIEW

Algorithm 11 summarizes the operation of our proposal. In fact, GSS can be split into two major steps: (1) rolling-ball owned for the first time, or (2) rolling-ball has already been owned. This last case, in turn, can be divided into two sub-cases: (a) the re-visited current owner of the rolling-ball is no longer a potential-cut, or (b) is still a potential-cut (Algorithm 11). These three scenarios are detailed in the following:

- First of all, when a node N_i receives a rolling-ball for the first time, it starts by contributing to its corresponding query. Then, this node changes its status from *unmarked* to one of the following statuses: *marked*, *ACN*, or *PCN*. At last, node N_i rolls the ball and delivers it to the chosen new owner.
- Second, in the case where the rolling-ball has already been received before by a node that is no longer PCN, the latter simply changes its status to *marked* and forwards the rolling-ball. Note that the fact that node N_i is no longer a PCN signifies also that this node cannot be either an ACN (relationship PCN-ACN). In Algorithm 11 (statement 16), in order to check whether the currently traversed node is PCN or not, we do not check its local status (`currentOwner.status`), but instead we re-apply Definition 17 (`isPCN()` method).

Algorithm 11 Boundary-First Search

```
1: The starting point of Q_j, which must be external, performs the following steps:
2:    contribute to Q_j;
3:    change its status to MARKED or PCN;
4:    forwardRollingBall(newOwner);
5:
6: Upon receiving the rolling-ball, currentOwner executes which follows:
7: // case 1: rolling-ball owned for the first time.
8: if (currentOwner.status == UNMARKED) then
9:    contribute to Q_j;
10:   change its status to MARKED, PCN or ACN;
11:   forwardRollingBall(newOwner);
12:   return;
13: end if
14:
15: // case 2: rolling-ball has already been owned.
16: if (not currentOwner.isPCN()) then
17:   // currentOwner is no longer PCN
18:   currentOwner.status = MARKED;
19:   forwardRollingBall(newOwner);
20: else
21:   if (currentOwner.status == ACN) then
22:      call Algorithm 13;
23:   else
24:      // currentOwner.status == PCN
25:      call Algorithm 12;
26:   end if
27: end if
```

- Finally, in the case where the rolling-ball has already been received before by a node that is still PCN, the latter needs to check its local status to determine whether it is an ACN or PCN. Because having already received the ball and being PCN according to `isPCN()` method means that the current node can be either PCN or ACN. Once the status has been determined, the last step consists of executing the proper piece of code. Algorithm 12 and Algorithm 13 summarize respectively the instructions that need to be followed by the PCNs and ACNs.

When the rolling-ball returns to a PCN, it brought with it valuable information, allowing hence this node to properly determine whether it can become ACN, remain PCN or leave the process by marking itself. As a matter of fact, a PCN decides about its new status not only according to the set from which the rolling-ball has returned but also according to its one-hop neighbors status. The behavior of a PCN N_i when it re-owns the rolling-ball is summarized in the following paragraph (Algorithm 12).

A potential-cut is a node that may or may not partition the set of unmarked nodes. The main objective behind the use of this concept (i.e., potential-cuts) is the efficient determination of actual-cuts in Ω_k (time and energy-efficiency). So, first, when a PCN N_i receives back the rolling-ball, the first performed step by this node is checking whether or not it is an ACN (`isACN()` method in Algorithm 12). If so, N_i changes its status as such and forwards the rolling-ball to the new owner. As it has been previously cited, a node is an ACN when it is PCN only to guarantee the connectivity of other ACNs (i.e., without its ACN neighbors, this node is not a PCN and can be marked). In the opposite case, i.e., if

Algorithm 12 PCN code

```
1: if (currentOwner.isACN()) then
2:    currentOwner.status = ACN;
3:    forwardRollingBall(newOwner);
4:    return;
5: end if
6:
7: switch (oldOwner.status){
8:  case MARKED: {
9:    if (rolling-ball has returned from same set 𝒳 and 𝒳 ≠ ϕ){
10:      currentOwner.status = ACN;
11:   }
12:   break;
13:  }
14:
15:  case PCN: {
16:   if (rolling-ball has returned from same set 𝒳){
17:      currentOwner.status = ACN;
18:   }
19:   else {
20:      cut link with oldOwner;
21:      change status to MARKED or PCN;
22:      sendCycleRemovalPacket(oldOwner);
23:      return;
24:   }
25:  }
26: }
27:
28: forwardRollingBall(newOwner);
```

N_i is not an ACN according to the previous rule, the execution of Algorithm 12 continues according to the status of the old owner (of the rolling-ball) and other criteria, as follows:

- First, in the case where the old owner of the ball is a marked inactive node, there are only two possible scenarios; the currently traversed node N_i which is a PCN can become an ACN or will keep the same status (of being PCN). The first scenario takes place when (1) the rolling-ball returns to N_i from the same set $\mathcal{X} \subset \mathcal{V}_i$ to which it was sent, and (2) $\mathcal{X}$ still contains alive active neighbors ($\mathcal{X} \neq \phi$). In such a scenario, it is obvious that node N_i is essential for the connectivity of $\mathcal{X}$. Consequently, N_i changes its status to ACN and forwards the rolling-ball. The second opposite scenario happens (1) when the rolling-ball returns to N_i from the same set $\mathcal{X} \subset \mathcal{V}_i$ to which it was delivered, but $\mathcal{X}$ is empty and no longer contains any alive active neighbors ($\mathcal{X} = \phi$), or (2) when the rolling-ball returns from a different set $\mathcal{Y} \subset \mathcal{V}_i$. In both cases, node N_i keeps its old status (i.e., PCN) and forwards the rolling-ball.

- Second, in the case where the old owner of the ball is a PCN, two possible scenarios can be distinguished; the currently traversed node N_i which is also a PCN can become an ACN or there is a cycle that needs to be broken. The first scenario happens when the rolling-ball returns to N_i from the same set $\mathcal{X} \subset \mathcal{V}_i$ to which it was intended. Note that here there is no need to check whether or not $\mathcal{X}$ still contains alive active neighbors (for sure $\mathcal{X} \neq \phi$ because the old owner is an alive active node).

Algorithm 13 ACN Code

```
1: switch (oldOwner.status){
2:   case MARKED: {
3:     if (set for which currentOwner is ACN == φ){
4:       currentOwner.status = PCN;
5:     }
6:     break;
7:   }
8:
9:   case PCN: {
10:    if (rolling-ball has returned from a different set 𝒴){
11:      cut link with oldOwner;
12:      change status to MARKED or PCN;
13:      sendCycleRemovalPacket(oldOwner);
14:      return;
15:    }
16:  }
17:
18:  case ACN: {
19:    if (rolling-ball has returned from a different set 𝒴){
20:      currentOwner.status = PCN;
21:    }
22:    break;
23:  }
24: }
25:
26: forwardRollingBall(newOwner);
```

In this first scenario, the currently traversed node N_i, which is fundamental for the connectivity of set $\mathcal{X}$, changes its status from *PCN* to *ACN* and as usual forwards the rolling-ball. The second alternative scenario (i.e., that is when the rolling-ball returns to node N_i from a different set $\mathcal{Y} \subset \mathcal{V}_i$) creates a looping case that must be properly treated and removed (next Section; 5.3.4).

Note that the previous two points have talked only about the cases where the old owner N_j can be a marked or PCN node but did not mention the third case in which N_j can be an ACN. Actually, the reason behind this is the fact that even when the old owner is an ACN, this does not signify that the currently traversed node N_i is also an ACN. As demonstrated in Algorithm 12, when the old owner N_j is an ACN but the current owner N_i is not, the latter simply remains as a PCN and delivers the rolling-ball.

Being an ACN does not mean that node N_i will stay as so throughout the whole process. In fact, similarly to PCNs, when the rolling-ball returns to an ACN, the latter determines to whether (1) stay an ACN, (2) return to its initial state of being PCN, or finally (3) mark itself to indicate its non-involvement. The new status of an ACN node is determined according to its one-hop neighbors' statuses (including the old owner) and the set from which the rolling-ball has returned to it. The actions performed by an ACN, upon receiving the rolling-ball, are described in Algorithm 13.

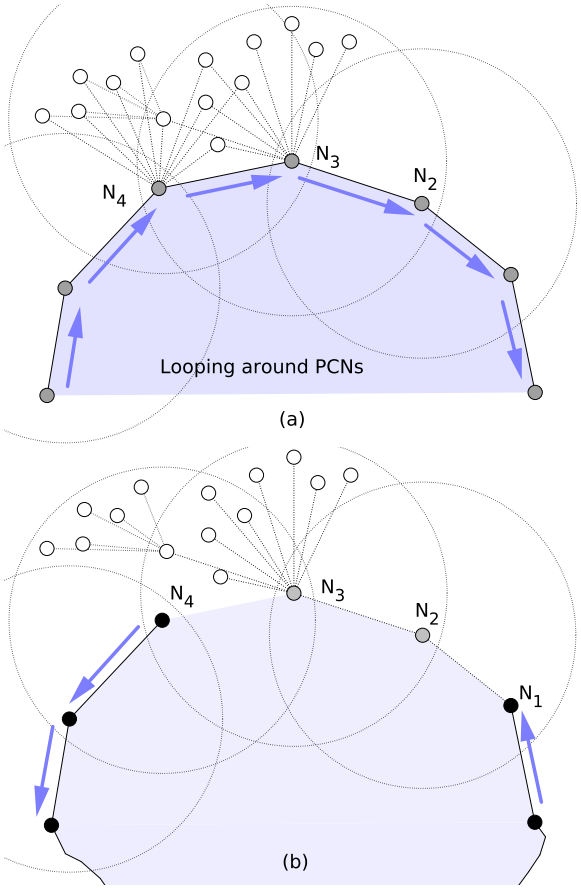


Figure 5.7: (a) Example of looping caused by the use of PCNs concept. (b) Appliance of cycle removal process.

5.3.4/ LOOPING AVOIDANCE MECHANISM

The concept of potential-cuts (Definition 17) keeps Ω connected and contributes to the localizability of GSS. Nonetheless, since this concept relies on $\mathcal{W}_i$ (which is a very limited knowledge), sometimes, it creates looping scenarios. In order to better explain the looping issue and the proposed solution, let us consider Fig. 5.7(a). In this example, according to Algorithm 11, node N_3 which is the query launcher changes its status to PCN and forwards the rolling-ball to $N_2 \in \mathcal{X}$ ($\mathcal{X} \subset \mathcal{W}_3$ represents the connected subset of one-hop active neighbors of N_3 to which node N_2 belongs; $\mathcal{X} = \{N_2\}$). The launched ball travels all around the initial boundary and gets back to N_3. Note that the latter which is still PCN has received the ball from another PCN ($N_4 \in \mathcal{Y} \neq \mathcal{X}$). This, actually, means that there are not only N_3 and N_4 that are PCNs but, in fact, a whole path of PCNs that connects these two nodes throughout the boundary. In such a situation, the rolling-ball will definitely get stuck, and will keep traversing the same boundary and re-marking each touched node as PCN (Fig. 5.7(a)).

In general, the looping issue occurs when (1) the rolling-ball is forwarded by a node N_i to a node N_k which belongs to a set $\mathcal{X} \subset \mathcal{V}_i$, but returns to N_i from N_j which belongs to a different set $\mathcal{Y} \subset \mathcal{V}_i$, and (2) N_i and N_j are both PCNs. To break this kind of cycles, as demonstrated by Algorithm 12, each node N_i which detects the existence of a cycle performs the following instructions (Fig. 5.7(b)). First of all, node N_i virtually

deletes its link with the old owner N_j. In other terms, N_i will no longer consider N_j as a neighbor. Second, after having done this, N_i changes its status (to *marked* or remains as a PCN) and broadcasts a cycle removal packet which orders the old owner N_j and the other nodes in range to virtually delete links responsible for this cycle creation. Once the looping issue has been properly removed, the old owner N_j updates its status and continues the process by delivering the ball to the next hop (Fig. 5.7(b)). Note that given the fact that another path exists connecting N_i and N_j, deleting links during the cycle removal process does not partition Ω. Note also that the main condition for a PCN to be an ACN is not to be contained in a cycle. For instance, in Fig. 5.7, node N_1, which is a PCN, can never be an ACN because it belongs to a cycle, and thus it is not required to maintain the connectivity of Ω.

5.4/ GSS PERFORMANCE ASSESSMENT

As it has been previously discussed in Chapters 3 and 4, in order to conduct the evaluations and assess the proposed solutions, we have opted for OMNeT++ [53]. Actually, among the different WSN frameworks proposed for this simulator, the selection has been made on Castalia [54], because besides its realistic radio/channel models (e.g., realistic modeling of RSSI and carrier sensing, interference, path loss, etc.), it also gives nodes a realistic behavior particularly with regard to medium access. We recall that we have used the well-known energy consumption model proposed by [55]. More specifically, in order to send a packet of k-bit size for a certain distance x, node N_i consumes $E_{TX}(k, x)$ Joules and consumes $E_{RX}(k)$ Joules to receive this same packet, such that:

$$E_{TX}(k, x) = E_{\mathrm{c}} * k + \epsilon_{\mathrm{amp}} * k * x^2$$

$$E_{RX}(k) = E_{\mathrm{c}} * k$$

where E_c and ϵ_{amp} are respectively the energy required to run the transmitter/receiver, and the energy needed to run the transmitter amplifier.

For comparison and evaluation purposes, in addition to the proposed SSF technique (i.e., GSS), we have implemented several other aggregation approaches among which, in the present chapter, we consider the Peeling Algorithm [18], Spreading Aggregation [39], Greedy Boundary Traversal [25], Depth-First Search [26], and Tree-based Aggregation [3]. Note that whatever the network topology, all the considered serial techniques in this section have been proven to ensure full network browsing. We, actually, did not implement the serial approach named *space-filling curve-based search* (described in Section 2.2) [24] nor the one called *PEGASIS* proposed in [22] because they do not ensure complete network traversal, which harm query completeness. On the other hand, apart from tree-based aggregation, no other parallel technique (such as the flooding or consensus-based aggregation) has been considered because, in addition to Chapter 3, many other works have already shown their ineffectiveness in terms of communications, time and energy, especially in large deployments [18, 25].

5.4.1/ EVALUATION METRICS

Besides the completeness metric, the four other criteria that were utilized to evaluate GSS and the other considered aggregation approaches are communications, time, energy, and

scalability.

- The communication criterion, which represents the number of packets issued to gather information from all sensor nodes, directly affects the three other ones (i.e., time, energy, and scalability). More precisely, the less an aggregation approach spends packets, the more efficient it is, and vice versa.
- The measured time criterion represents the delay that elapses between the moment when information gathering has been launched by the query initiator and the instant when the response is received by that node. The goal is always to have a very low latency.
- The energy criterion represents the sum of energy spent by all nodes in the network to gather and deliver the required information. We recall that the energy consumed by each node has been estimated using the model proposed in [55] (described earlier).

  Energy is the most precious resource in WSNs, preserving it is the main aim of every approach proposed for this kind of networks.
- In order to assess the scalability of GSS and the other implemented aggregation techniques, different network size and density scenarios have been considered in the conducted simulations. At first, only 100 wireless nodes were randomly placed in a 1000 x 1000 meters field. Then, gradually, at each step, 50 new wireless nodes were randomly inserted in the field, to a maximum of 500 nodes.

  We point out that GSS has not been evaluated only in the aforementioned random configurations, but also in different communication range and different network size and topology (e.g., linear, ring, different grids, regular networks with and without holes, irregular topologies, etc.). All the network configurations we generated whether manually or randomly have been successfully and efficiently browsed by GSS.

5.4.2/ SIMULATION PARAMETERS

Table 5.1 and Table 5.2 list respectively the different simulation parameters considered in the present chapter, and the average degree of nodes in the different deployments. The first line of Table 5.2 depicts the number of nodes in the network whereas the second one represents the corresponding average number of neighbors of each node.

5.4.3/ SIMULATION RESULTS

In the following two subsections, the obtained simulation results of the considered aggregation approaches are compared and evaluated. For the sake of reading ease and figures clarity, we begin by plotting and comparing the results of our proposal with only those of the implemented serial approaches (Section 5.4.3.1). Afterward (in Section 5.4.3.2), we plot the results of GSS and compare them with mainly the implemented tree-based approach. We point out that in both sections, communication, time and energy results have been plotted against the number of nodes specified in Table 5.1.

Table 5.1: GSS simulation configuration and parameters

Parameter	**Value(s)**
Field dimensions (m^2)	1000 x 1000
Nodes number	100, 150, 200, . . . , 500
Nodes deployment	Uniform
Nodes transmission range (m)	150
Nodes initial energy (J)	100
Query launcher location	Random
Query packet size ($byte$)	50
Radio type	CC2420
Radio data rate ($kbps$)	250
MAC protocol	CSMA/CA

Table 5.2: GSS average nodes' degree.

100	**150**	**200**	**250**	**300**	**350**	**400**	**450**	**500**
6	9	12	15	18	21	25	27	31

5.4.3.1/ GSS VERSUS SERIAL INFORMATION GATHERING TECHNIQUES

Fig. 5.8 depicts the transmissions that have been spent to fulfill information gathering by the different implemented serial approaches, namely GSS (proposed approach), PA (Peeling Algorithm) [18], SA (Spreading Aggregation) [39], GBT (Greedy-Boundary Traversal) [25], DFS (Depth-First Search) [26], and finally OPT (Optimal Algorithm). The latter represents an imaginary optimal serial algorithm[4] that spends $n-1$ transmissions to browse and query a network of n wireless connected nodes. In fact, both DFS and OPT are staples when it comes to serial approaches comparison. In exact words, the results of these two algorithms serve as an evaluation gauge for the efficiency of serial fusion approaches in terms of communications. An efficient serial algorithm must lay between OPT and DFS.

The calculated number of transmissions plotted in Fig. 5.8 includes path construction, query diffusion, data fusion, and result delivery to the query launcher. We underline that while the transmissions required by DFS and OPT can be expressed mathematically, the ones required by whether GSS, PA, SA, or GBT cannot because of the adaptive localized nature of these approaches. Recall that due to its inherent backtracking behavior, DFS consumes $2*(n-1)$ packets to query n wireless connected nodes and deliver the response to the query launcher (i.e., double the packets required by OPT).

When compared to DFS, GBT, PA, and SA from the standpoint of communications, the proposed approach is much more efficient. The obtained results depicted in Fig. 5.8 affirm the outperformance of GSS and its very low communication requirement. As a matter of fact, the denser the wireless network, the lower the consumption of GSS in terms of packets will be. This, however, does not mean that GSS loses its effectiveness in small-sized networks. In fact, the performance result of GSS is very close to that of OPT even in low-density sparse networks. For instance, GSS needs approximately 120

[4]Laying out a Hamiltonian path that does not exist in every network topology is an NP-complete problem [20].

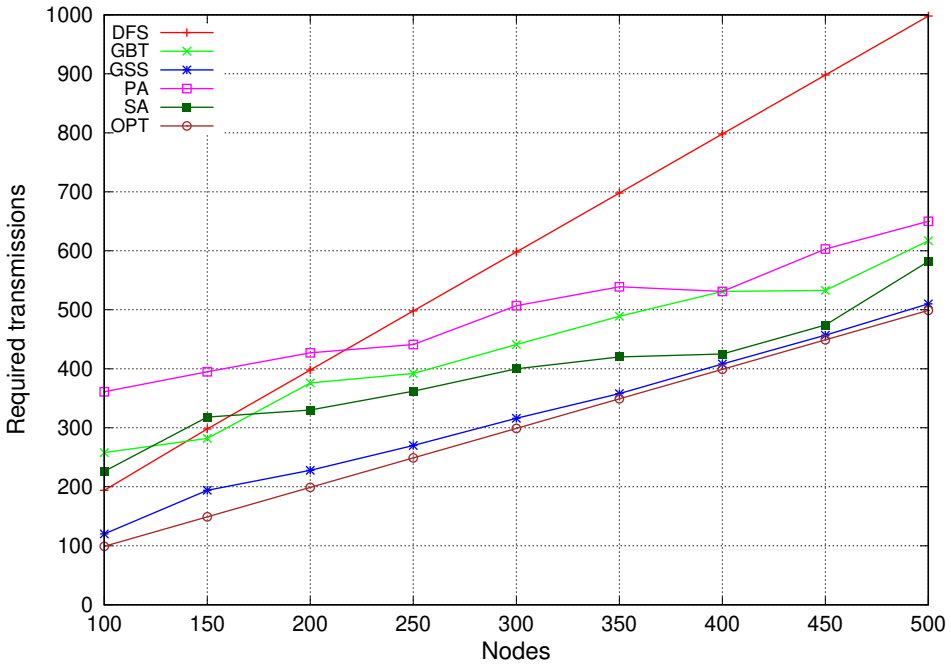


Figure 5.8: GSS versus serial approaches. Comparison in terms of required communications (sent packets).

hops to explore a network of 100 wireless nodes, and 510 hops to traverse 500 nodes.

In sparse low-density networks, GSS is slightly different than OPT because the looping scenarios created by potential-cuts have a high probability of occurrence (see Definition 17 and Fig.5.7). In such situations, due to the limited knowledge of nodes, the rolling-ball must travels all around the current boundary to inform the currently visited node about the existence or absence of a cycle. We recall that the optimality of GSS stems mainly from the originality of the idea we proposed to overcome the limited knowledge of nodes and to preserve the connectivity of the network. Concretely speaking, the trick was assigning a twofold role to the rolling ball. In addition to its main traversal mission, the latter was given a probing role, which considerably contributes to saving communications, time and energy. As Fig. 5.8 demonstrates, despite the certain presence of cycles in low-density networks, GSS is not drastically affected as is the case for the other serial approaches. As a matter of fact, in sparse networks, DFS which is a non-sophisticated serial approach outperforms PA, SA, and GBT. This is due, as previously mentioned, to the fact that in low-density deployments, it is more likely that the network contains more cycles, and hence requires more communications to get traversed. For instance, the bad performance of SA in sparse networks comes from the excessive use of DBS packets aimed to detect the existence of potential cycles. The presence of cycles results in greater packets expenditure by SA, PA, and GBT. A denser deployment, in contrast, yields fewer holes and hence fewer cycles, which improves the performance of these approaches. Actually, when the number of nodes increases, SA, PA, and GBT approximates the optimal number of transmissions, and the difference between them and DFS becomes more noticeable. For example, in the 500-nodes deployment, SA, PA, and GBT require almost third the communications consumed by DFS, while GSS requires almost half the transmissions of DFS (the difference between GSS and DFS is almost 50% in all the considered network sizes).

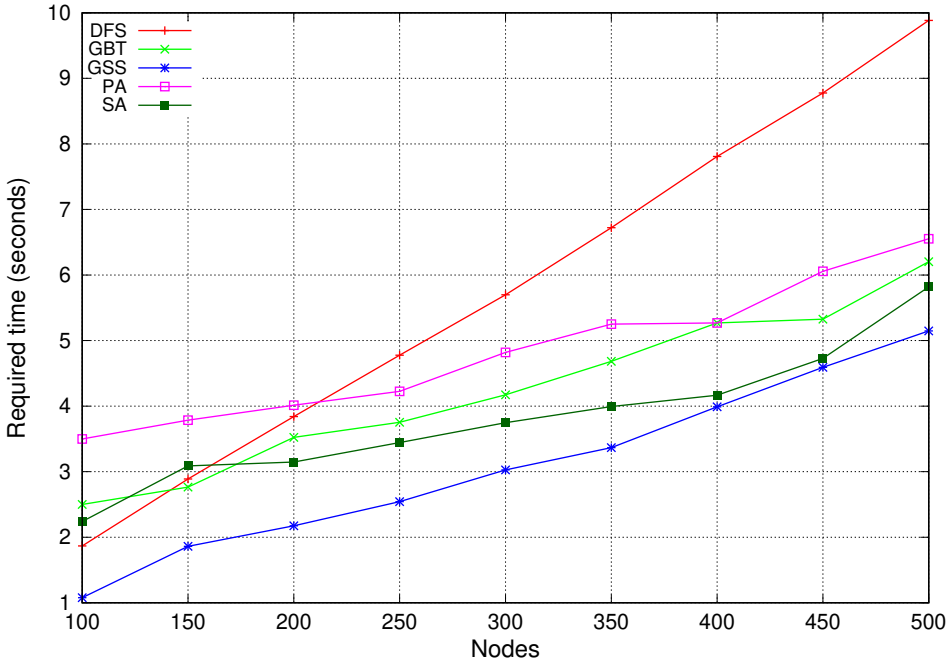


Figure 5.9: GSS versus serial approaches. Comparison in terms of information gathering time.

Given its sequential nature, the more a serial aggregation approach requires transmissions, the more time it will need to finish, and vice versa. Since GSS necessitates the smallest number of packets, it outperforms the other serial techniques in terms of data fusion rapidity (Fig. 5.9). In terms of energy, GSS also outperforms the other serial approaches (Fig. 5.10). As a matter of fact, in a WSN, energy is spent by the different actions performed by nodes such as communication, sensing, and computation. In the case where both computation and sensing are straightforward non-complicated tasks, the main source of energy consumption will be transmissions/receptions [1]. In other words, in such a scenario, the more a serial approach spends packets, the larger its energy consumption will be. As GSS requires the lowest amount of packets among the other serial techniques, it is superior from the energy efficiency standpoint.

5.4.3.2/ GSS VERSUS TREE-BASED INFORMATION GATHERING

In addition to DFS and OPT which have been pointed out to in the previous section, Fig. 5.11 also shows the number of transmissions that have been issued by both GSS and the considered tree-based approach; Tree-A [3]. The time and energy that GSS and Tree-A have necessitated to harvest the required information and deliver the response to the query-launcher are depicted respectively in Fig. 5.12 and Fig. 5.13. Note that with regard to transmissions, the following three types of packets have been taken into consideration. First, packets spent to traverse the network or respectively construct the tree. Second, packets issued to query the whole wireless network. Finally, packets utilized to report the gathered fused data to the query initiator.

We recall that in Tree-A or any other tree-based aggregation approach, structure establishment, query diffusion, and data fusion are three separate independent tasks. At first,

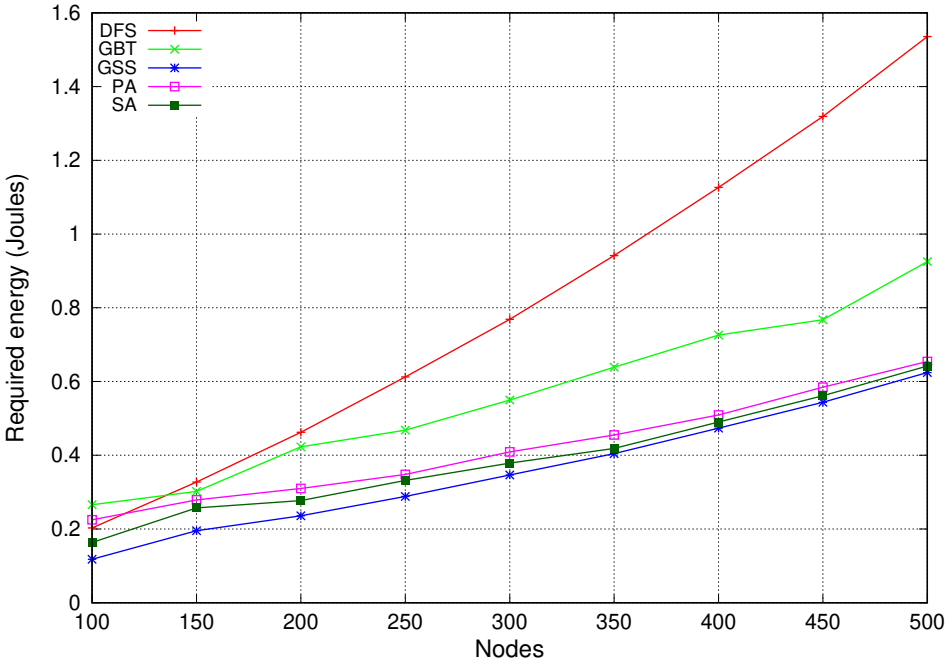


Figure 5.10: GSS versus serial approaches. Comparison in terms of information gathering energy.

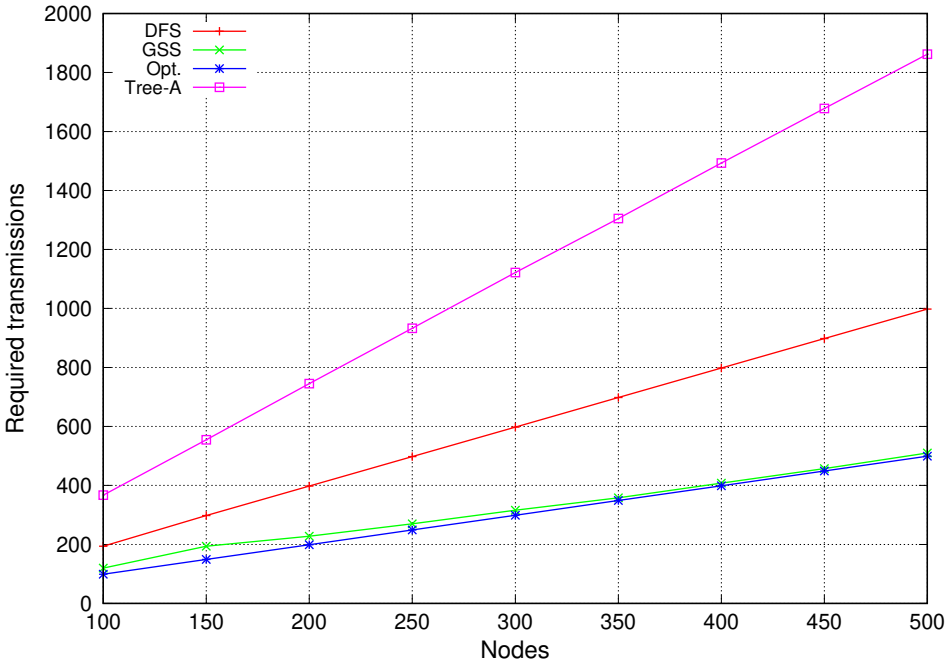


Figure 5.11: GSS versus Tree-based aggregation. Comparison in terms of required communications (sent packets).

the network must be thoroughly covered with a spanning tree[5]. After being built, the tree can be interrogated by its root. At last, query response starts from leaves and gradually goes up towards the root. In GSS, like in any other SSF approach, these three separate

[5]Query accuracy requires the covering of each and every node in the network.

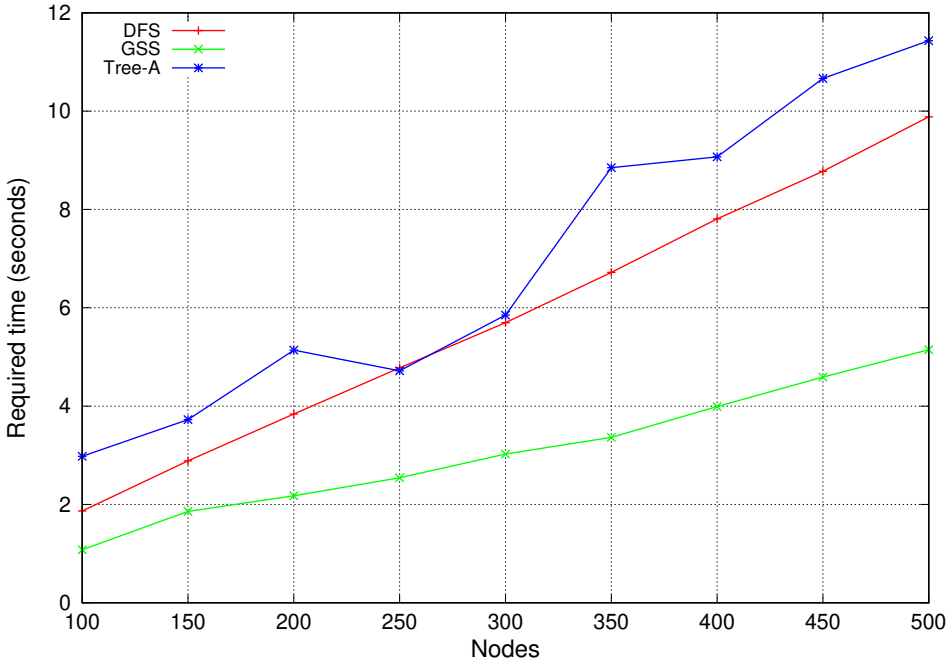


Figure 5.12: GSS versus Tree-based aggregation. Comparison in terms of information gathering time.

operations have been all melted down into a single phase. Simultaneously, as the network is being progressively browsed, nodes are interrogated and their data is collected. This compactness feature of SSF approaches allows them to significantly reduce the required transmissions, time, and energy.

The big difference between GSS and Tree-A in regards to energy dissipation can be explained as follows. Actually, in addition to the separate indispensable (1) building/fixing of the spanning tree, (2) query dissemination, and (3) response delivery, given its concurrent collision-prone behavior, Tree-A also consumes a lot of energy to retransmit the lost collided packets, especially in dense deployments. As regards information gathering time, intuitively, even with a larger number of transmitted packets, Tree-A is supposed to beat all serial approaches thanks to its parallel concurrent performance. In reality, however, this is not the case. Due to what has been already mentioned, GSS shows a better response than Tree-A (Fig.5.12).

In a generic tree-based aggregation technique, the number of packets required to accomplish data fusion can be calculated as follows. Actually, without considering the retransmissions of collided packets and without considering tree construction or any probable maintenance operation of this structure, $2 * n$ packets need to be sent to fulfill data gathering from a network of n wireless nodes. More specifically, in query diffusion phase, n packets will be broadcast because all nodes are involved in this stage. During data fusion and query processing phase, n other packets will be issued because each node must forward a packet to the upper level except the root. As previously mentioned in Section 2.2, constructing or even maintaining a distributed logical structure over a wireless error/interference-prone environment is a complicated task that requires a considerable time and energy. On the other hand, in addition to the fact that SSF approaches are maintenance-free, do not generate any collisions, and do not require any initial separate

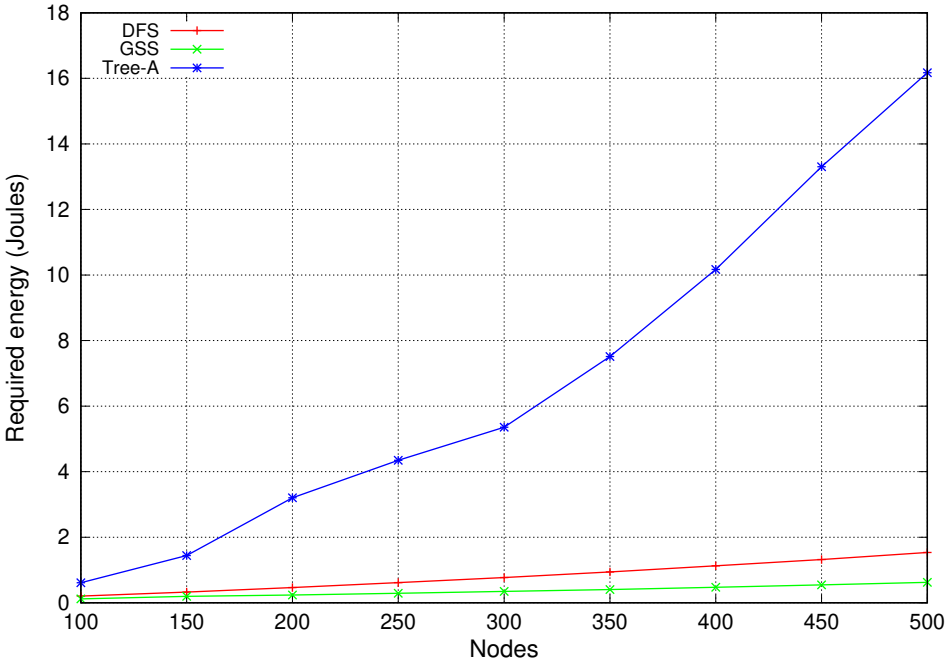


Figure 5.13: GSS versus Tree-based aggregation. Comparison in terms of information gathering energy.

structure building phase, the existence of an optimal itinerary (that crosses each node once) means that only $n-1$ packets are needed to accomplish the desired information collection.

Apart from their balance and robustness issues, the main drain in tree-based approaches stems from collisions and retransmissions. Actually, packet collisions considerably affect these approaches in terms of time and energy, especially in dense large-scale wireless networks (Fig.5.12 and 5.13). Besides the fact that retransmissions significantly waste time and energy, they are themselves subject to error and collisions, which deeply affect these approaches. The problem of unbalanced energy consumption is also considered as one of the main issues in tree-based aggregation approaches [12]. This phenomenon occurs primarily because wireless nodes located nearby the root are heavily solicited to forward the upward traffic. The unbalanced energy consumption issue can also take place when the initially created tree is not balanced. Finally, in tree-based approaches, structure-maintenance is indispensable because a spanning tree is nothing but a bunch of pre-set paths that are very fragile to failures and topology changes. The maintenance solution can handle those problems at the expense of being very costly in terms of time and energy, particularly in very large deployments.

5.5/ CONCLUSION

Serial structure-free (SSF) approaches have proven their efficiency in several aspects but have also shown their deficiency in front of path construction complexity. This chapter has thoroughly addressed this issue by presenting *Geometric Serial Search* (GSS); a novel scalable SSF approach specifically fashioned to effectively gather information from large

and medium-scale WSNs. The main contribution of this approach is its reliance on a new efficient localized path construction algorithm that requires fewer communications to derive short near-optimal visiting itineraries. Actually, the only input of this algorithm is the neighbors' table of each traversed node, and the only tool used to traverse networks is a packet which has figuratively the shape of a rolling ball. To ensure a full network search, this packet, which can be issued by any node, sees wireless networks as a set of boundaries (layers). While progressively finding its non-predefined track across boundaries, the rolling ball simultaneously queries and collects the encountered nodes answers. In addition to its inherent collision-free nature, high scalability, robustness, and energy/time efficiency, the extensive conducted OMNeT++ tests have confirmed that GSS constantly approaches the optimal targeted number of transmissions. The performed OMNeT++ simulations have also confirmed that GSS avoids indefinite looping and ensures the traversal of all nodes regardless of the network topology.

As future directions, we are interested in exploiting GSS in other areas of WSNs such processing topological and spatial window queries. We also plan to address some other issues such as the effect of imprecise nodes' locations on GSS and to what extent is this approach robust in face of failures in links/nodes and other topological changes.

6

GENERAL CONCLUSION AND FUTURE WORKS

The issue of querying large wireless sensor networks through data aggregation is crucial because of both the limited energy supply of nodes and the query time latency. Strictly speaking, in WSNs, tasks such as data processing and query result delivery to the sink can be performed in two different ways: concurrently or sequentially. Likewise, and accordingly, the numerous research works proposed in the literature as solutions for the querying problem can be categorized into four classes [3, 10, 20, 24, 25, 46, 47]: (1) parallel structure-based approaches with a pre-constructed logical infrastructure (a spanning tree for instance), (2) parallel structure-free or iterative approaches in which the query result is iteratively carried out through one-hop communications, (3) serial structure-based approaches in which the query answer is sequentially obtained via a pre-constructed itinerary that passes through every node in the network, and (4) serial structure-free approaches which gradually build paths, and at the same time, interrogate nodes and gather their answers (i.e., each time a new query is launched, a new path is drawn for it).

Targeting large-scale or even medium network deployments, recent research has shown that compared with the other three approaches cited above, serial structure-free aggregation is scalable and very efficient in terms of collisions avoidance, communication reduction, energy conservation, and, more importantly, in terms of responsiveness. For instance, parallel structure-based approaches operate in three separate phases: (i) structure construction (e.g., trees, clusters, ...), (ii) query dissemination, and (iii) data aggregation. First, a spanning tree is created over the network. Once the whole network is covered, a query can then be diffused throughout the tree to all nodes in the network. Finally, data aggregation starts from leaf nodes and goes up towards the root/sink. Performing these three phases separately increases energy consumption and delays the response time, particularly in very large-scale networks. Conversely, serial structure-free approaches save energy and time because they combine the three previous phases into one phase. While the path is gradually constructed at each hop with the help of local information (i.e., a localized nature), the query is being disseminated and data aggregation is taking place at the same time. Furthermore, in opposition to parallel or serial structure-

based approaches in which only the sink/root is able to request information from sensor nodes, in serial structure-free approaches, queries can be launched from any node in the network. This is, in fact, a fundamental need for emergent multi-owner-multi-user WSNs where several nodes can be responsible for updating configuration parameters and/or distributing management commands [50]. In addition to all this, as serial approaches involve only one communication at each moment in time, they can therefore avoid collisions, improving hence their performance in terms of energy conservation and delay time reduction. This is not the case for parallel approaches; they generate a lot of collisions during structure construction, query diffusion, and data aggregation, particularly in dense networks [25], even when sophisticated MAC protocols are used [49].

As confirmed by the evaluation studies we have conducted, despite their interesting features/advantages and outperformance in comparison with other approaches, state-of-the-art serial structure-free processing techniques still suffer from several deficiencies among which we mainly cite path construction. More exactly, recent serial approaches require an additional cost to construct their non-optimal and relatively long traversal paths. In fact, the major challenges encountered when designing an efficient serial aggregation approach are (1) ensuring the exploration of all nodes in the network whatever the considered topology, and (2) reducing the needed time and energy. To achieve the latter objective, two points must be taken into consideration. First, finding the shortest possible visiting path (a short path enhances the overall performance and vice versa), and second, avoiding extra communications by counting only on the local knowledge of each traversed node to explore the network.

In this work, in order to respond to these interesting and challenging questions, and in order to provide an efficient information gathering mechanism for wireless resource-constrained networks, our primary objective was developing scalable serial algorithms that shorten the traversal path and reduce communications to the maximum extent possible. The objective was attaining the optimal number of hops, i.e., $n-1$ packets to traverse a network of n nodes; no extra overhead or control packets must be required. In addition to that primary goal, our second aim was the algorithms ability to explore any connected network and to visit all of its nodes regardless of the topology in hand (regular/irregular, large/sparse, with/without holes, etc.). Concretely speaking, in this manuscript, we proposed three novel serial processing approaches, called Peeling Algorithm (PA), Spreading Aggregation (SA), and Geometric Serial Search (GSS), which have been specifically designed to be scalable and support very large numbers of nodes. The obtained simulation results confirm that the proposed algorithms outperform state-of-the-art approaches in terms of deriving shorter visiting paths, and in terms of time and energy consumption reduction. The simulation results we obtained also confirm the very good scalability of our three proposals. In addition to these interesting experimental results, we provided in this manuscript formal proofs that demonstrate the correctness of the proposed approaches; i.e., they terminate and are able of visiting all connected nodes without falling into looping.

First, Peeling Algorithm (PA) [18], which as its name hints, peels the network layer-by-layer starting from the external boundary. In other words, the exploration of nodes starts by progressively removing (marking as visited) the external boundary (layer) of the network and hence disclosing a new internal unexplored layer which will be removed (peeled) in its turn, and so forth. Initially, in order to find the peeling starting point (i.e., any node located on the network external boundary), the query launcher (which can also be any node in the network) collaborates with other nodes. Once the network external boundary has been determined and the starting point has been found, the network peeling can start

from this node using the graph-free boundary traversal algorithm called curved-stick [38]. Being costly in terms of energy and time, if executed with each new query, the process of network external boundary determination can considerably harm the overall peeling performance. However, this is not the case, this process is executed only once during network setup (unless, of course, we have unstable topologies with continuous changes).

Our second approach; Spreading Aggregation (SA) [39, 40] starts information gathering by simply placing a rolling ball [41] (launching a single packet) and letting it sequentially hop over nodes and aggregate their data. The next hop of the querying/aggregation packet is determined locally by each traversed node using only its one-hop neighborhood information. No network topology information needs to be known by nodes nor collisions are generated as only one node is communicating at any given time. In addition to this, unlike the peeling technique [18], in SA, there is no necessity for network external boundary determination. As a matter of fact, in SA, non-boundary nodes can be query launchers by simply utilizing a shrunken rolling-ball. The latter visits nodes and gets progressively enlarged if possible at each hop until eventually it gains its optimal shape and continues the ordinary traversal process. As confirmed by the numerous OMNeT++ simulation results we obtained, Spreading Aggregation is highly scalable and very effective in terms of communication, energy, and time consumption reduction. The obtained results also confirm the superiority of SA over state-of-the-art serial and parallel approaches, particularly in very large-scale network deployments.

Finally, in spite of the fact that PA and SA excel in large and medium-scale networks, their path-construction mechanisms are not optimal and can be improved by reducing the involved communications and further shortening the visiting path. Geometric Serial Search (GSS) [42, 43]; our third approach, came to respond to this need by totally excluding control packets and relying only on one packet that jumps from node to node and explores the entire network. In fact, while progressively traversing the network, this packet simultaneously interrogates nodes and harvests their query responses. Paths created by GSS are not constructed in advance, can commence from any node in the network, and necessitates only the one-hop neighbors' table of each explored node to be traced. As a matter of fact, in addition to its completeness[1], collision-freeness, scalability, energy/time efficiency, and robustness[2], GSS considerably reduces communications and lays out very short traversal paths by always approaching the optimal number of hops (i.e., $n-1$ packets to explore n nodes). The obtained OMNeT++ simulation results presented in this manuscript demonstrated the efficiency of GSS as an information gathering technique for large wireless resource-constrained networks and confirmed all the points cited above.

To summarize, serial localized processing algorithms explore nodes one-by-one and are able of performing a multitude of tasks including scheduling nodes, querying or gathering data from nodes, supplying nodes with data, etc. As future research directions, we are interested in exploiting localized serial processing in other areas of WSNs such as *topological aggregation* and *spatial window querying*. In the latter example, instead of addressing queries to the entire network, the end-user indicates the region to be interrogated by specifying a rectangular shape (or any other geometric form) called Window Query Region (WQR). For instance, through the below query, the end-user will obtain the average temperature, maximum humidity, minimum light, and finally the number of sensor nodes in the region bounded by a rectangle defined by two points.

[1] Ability to explore all nodes is a fundamental requirement in many sensitive practical applications.
[2] Resilience to topology changes (failures in links/nodes, ...).

```
SELECT AVG(temperature), MAX(humidity), MIN(light), COUNT()
FROM sensors as s
WHERE s.location WITHIN [(75, 75), (320, 630)]
```

The idea behind serial window querying is to reach the WQR by means of any efficient geographic routing protocol. Once the desired WQR has been reached, the querying process starts by sequentially collecting the answers of all nodes located inside that region. Finally, the obtained query result has to be sent to the sink/query launcher using the same geographic routing protocol that has been utilized in the beginning. Even though this technique seems to be simple, it is in reality hard to design because it must be able to handle any network deployment, especially those with communication holes. In other terms, the proposed approach has to be able to deal with the so-called void problem and guarantee query accuracy by visiting all nodes located inside the WQR in question. In fact, we have started a research work on this matter and the preliminary results are encouraging and very promising.

As other perspectives, some other issues of localized serial processing algorithms can be also addressed such as (1) the effect of imprecise nodes' locations on serial aggregation in general and on our proposed approaches in particular. (2) To what extent are serial structure-free approaches robust in face of failures in links/nodes and other topological changes.

BIBLIOGRAPHY

[1] Azzedine Boukerche. *Algorithms and protocols for wireless sensor networks*, volume 62. John Wiley & Sons, 2008.

[2] Ian F. Akyildiz and Mehmet Can Vuran. *Wireless sensor networks*, volume 4. John Wiley & Sons, 2010.

[3] Ramesh Rajagopalan and Pramod K. Varshney. Data-aggregation techniques in sensor networks: A survey. *IEEE Communications Surveys and Tutorials*, 8(1-4):48–63, 2006.

[4] Soummya Kar, José M. F. Moura, and Kavita Ramanan. Distributed parameter estimation in sensor networks: Nonlinear observation models and imperfect communication. *IEEE Trans. Information Theory*, 58(6):3575–3605, 2012.

[5] Angelia Nedic, Alexander Olshevsky, Asuman E. Ozdaglar, and John N. Tsitsiklis. On distributed averaging algorithms and quantization effects. *IEEE Trans. Automat. Contr.*, 54(11):2506–2517, 2009.

[6] Amir Beck, Petre Stoica, and Jian Li. Exact and approximate solutions of source localization problems. *IEEE Trans. Signal Processing*, 56(5):1770–1778, 2008.

[7] Jianyong Lin, Wendong Xiao, Frank L. Lewis, and Lihua Xie. Energy-efficient distributed adaptive multisensor scheduling for target tracking in wireless sensor networks. *IEEE Trans. Instrumentation and Measurement*, 58(6):1886–1896, 2009.

[8] Engin Masazade, Ruixin Niu, Pramod K. Varshney, and Mehmet Keskinoz. Energy-aware iterative source localization for wireless sensor networks. *IEEE Trans. Signal Processing*, 58(9):4824–4835, 2010.

[9] Bahador Khaleghi, Alaa M. Khamis, Fakhreddine Karray, and Saiedeh N. Razavi. Multisensor data fusion: A review of the state-of-the-art. *Information Fusion*, 14(1):28–44, 2013.

[10] Xiang-Yang Li, Yajun Wang, and Yu Wang. Complexity of data collection, aggregation, and selection for wireless sensor networks. *IEEE Trans. Computers*, 60(3):386–399, 2011.

[11] Xiaobing Wu, Guihai Chen, and Sajal K. Das. Avoiding energy holes in wireless sensor networks with nonuniform node distribution. *IEEE Trans. Parallel Distrib. Syst.*, 19(5):710–720, 2008.

[12] Anfeng Liu, Peng-Hui Zhang, and Zhi-Gang Chen. Theoretical analysis of the lifetime and energy hole in cluster-based wireless sensor networks. *J. Parallel Distrib. Comput.*, 71(10):1327–1355, 2011.

[13] Soummya Kar and José M. F. Moura. Distributed consensus algorithms in sensor networks with imperfect communication: Link failures and channel noise. *IEEE Trans. Signal Processing*, 57(1):355–369, 2009.

[14] Boris N. Oreshkin, Mark Coates, and Michael G. Rabbat. Optimization and analysis of distributed averaging with short node memory. *IEEE Trans. Signal Processing*, 58(5):2850–2865, 2010.

[15] Stacy Patterson, Bassam Bamieh, and Amr El Abbadi. Convergence rates of distributed average consensus with stochastic link failures. *IEEE Trans. Automat. Contr.*, 55(4):880–892, 2010.

[16] Lin Xiao, Stephen P. Boyd, and Seung-Jean Kim. Distributed average consensus with least-mean-square deviation. *J. Parallel Distrib. Comput.*, 67(1):33–46, 2007.

[17] Satish M. Srinivasan and Azad H. Azadmanesh. Survivable data aggregation in multiagent network systems with hybrid faults. *IEEE Trans. Computers*, 62(10):2054–2068, 2013.

[18] Ahmed Mostefaoui, Azzedine Boukerche, Mohammed Amine Merzoug, and Mahmoud Melkemi. A scalable approach for serial data fusion in wireless sensor networks. *Computer Networks*, 79:103–119, 2015.

[19] Robert D. Nowak. Distributed EM algorithms for density estimation and clustering in sensor networks. *IEEE Trans. Signal Processing*, 51(8):2245–2253, 2003.

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

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

[22] Stephanie Lindsey, Cauligi S. Raghavendra, and Krishna M. Sivalingam. Data gathering algorithms in sensor networks using energy metrics. *IEEE Trans. Parallel Distrib. Syst.*, 13(9):924–935, 2002.

[23] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[24] Swapnil Patil, Samir R. Das, and Asis Nasipuri. Serial data fusion using space-filling curves in wireless sensor networks. In *Proceedings of the 1st Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, SECON'04, Santa Clara, CA, USA, October 4-7, 2004*, pages 182–190.

[25] Azzedine Boukerche, Ahmed Mostefaoui, and Mahmoud Melkemi. Efficient and robust serial query processing approach for large-scale wireless sensor networks. *Ad Hoc Networks*, 47:82–98, 2016.

[26] Shimon Even and Guy Even. *Graph Algorithms, Second Edition*. Cambridge University Press, 2012.

[27] Azzedine Boukerche and Xin Fei. A coverage-preserving scheme for wireless sensor network with irregular sensing range. *Ad Hoc Networks*, 5(8):1303–1316, 2007.

[28] Azzedine Boukerche, Xin Fei, and Regina Borges de Araujo. An optimal coverage-preserving scheme for wireless sensor networks based on local information exchange. *Computer Communications*, 30(14-15):2708–2720, 2007.

[29] Azzedine Boukerche, Horacio A. B. F. Oliveira, Eduardo F. Nakamura, and Antonio A. F. Loureiro. Localization systems for wireless sensor networks. *IEEE Wireless Commun.*, 14(6):6–12, 2007.

[30] Horacio A. B. F. Oliveira, Azzedine Boukerche, Eduardo F. Nakamura, and Antonio A. F. Loureiro. An efficient directed localization recursion protocol for wireless sensor networks. *IEEE Trans. Computers*, 58(5):677–691, 2009.

[31] Jing Wang, Ratan K. Ghosh, and Sajal K. Das. A survey on sensor localization. *Journal of Control Theory and Applications*, 8(1):2–11, 2010.

[32] Guang Tan, Hongbo Jiang, Shengkai Zhang, Zhimeng Yin, and Anne-Marie Kermarrec. Connectivity-based and anchor-free localization in large-scale 2d/3d sensor networks. *TOSN*, 10(1):6:1–6:21, 2013.

[33] Mort Naraghi-Pour and Gustavo Chacon Rojas. A novel algorithm for distributed localization in wireless sensor networks. *TOSN*, 11(1):1:1–1:25, 2014.

[34] Fu Xiao, Wei Liu, Zhetao Li, Lei Chen, and Ruchuan Wang. Noise-tolerant wireless sensor networks localization via multinorms regularized matrix completion. *IEEE Trans. Vehicular Technology*, 67(3):2409–2419, 2018.

[35] Jukka Suomela. Survey of local algorithms. *ACM Comput. Surv.*, 45(2):24:1–24:40, 2013.

[36] Eduardo F. Nakamura, Antonio A. F. Loureiro, Azzedine Boukerche, and Albert Y. Zomaya. Localized algorithms for information fusion in resource-constrained networks. *Information Fusion*, 15:2–4, 2014.

[37] Ahmed Mostefaoui, Mahmoud Melkemi, and Azzedine Boukerche. Routing through holes in wireless sensor networks. In *15th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM'12, Paphos, Cyprus, October 21-25, 2012*, pages 395–402.

[38] Ahmed Mostefaoui, Mahmoud Melkemi, and Azzedine Boukerche. Localized routing approach to bypass holes in wireless sensor networks. *IEEE Trans. Computers*, 63(12):3053–3065, 2014.

[39] Mohammed Amine Merzoug, Azzedine Boukerche, Ahmed Mostefaoui, and Samir Chouali. Spreading aggregation: A distributed collision-free approach for data aggregation in large-scale wireless sensor networks. *J. Parallel Distrib. Comput.*, 125:121–134, 2019.

[40] Mohammed Amine Merzoug, Ahmed Mostefaoui, and Samir Chouali. Distributed collision-free data aggregation approach for wireless sensor networks. In *13th IEEE International Conference on Distributed Computing in Sensor Systems, DCOSS'17, Ottawa, ON, Canada, June 5-7, 2017*, pages 175–182.

[41] Wen-Jiunn Liu and Kai-Ten Feng. Greedy routing with anti-void traversal for wireless sensor networks. *IEEE Trans. Mob. Comput.*, 8(7):910–922, 2009.

[42] Mohammed Amine Merzoug, Azzedine Boukerche, and Ahmed Mostefaoui. Efficient information gathering from large wireless sensor networks. *Computer Communications*, 132:84–95, 2018.

[43] Mohammed Amine Merzoug, Azzedine Boukerche, and Ahmed Mostefaoui. Serial in-network processing for large stationary wireless sensor networks. In *Proceedings of the 20th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM'17, Miami, FL, USA, November 21-25, 2017*, pages 153–160.

[44] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[45] Elena Fasolo, Michele Rossi, Jörg Widmer, and Michele Zorzi. In-network aggregation techniques for wireless sensor networks: a survey. *IEEE Wireless Commun.*, 14(2):70–87, 2007.

[46] Hongyu Gong, Luoyi Fu, Xinzhe Fu, Lutian Zhao, Kainan Wang, and Xinbing Wang. Distributed multicast tree construction in wireless sensor networks. *IEEE Trans. Information Theory*, 63(1):280–296, 2017.

[47] Long Cheng, Jianwei Niu, Chengwen Luo, Lei Shu, Linghe Kong, Zhiwei Zhao, and Yu Gu. Towards minimum-delay and energy-efficient flooding in low-duty-cycle wireless sensor networks. *Computer Networks*, 134:66–77, 2018.

[48] Panayiotis Andreou, Andreas Pamboris, Demetrios Zeinalipour-Yazti, Panos K. Chrysanthis, and George Samaras. ETC: energy-driven tree construction in wireless sensor networks. In *10th International Conference on Mobile Data Management, MDM'09, Taipei, Taiwan, May 18-20, 2009*, pages 513–518.

[49] Qiang Ma, Kebin Liu, Zhichao Cao, Tong Zhu, Xin Miao, and Yunhao Liu. Opportunistic concurrency: A MAC protocol for wireless sensor networks. *IEEE Trans. Parallel Distrib. Syst.*, 26(7):1999–2008, 2015.

[50] Daojing He, Sammy Chan, Mohsen Guizani, Haomiao Yang, and Boyang Zhou. Secure and distributed data discovery and dissemination in wireless sensor networks. *IEEE Trans. Parallel Distrib. Syst.*, 26(4):1129–1139, 2015.

[51] Brad Karp and Hsiang-Tsung Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking, MOBICOM'00, Boston, MA, USA, August 6-11, 2000*, pages 243–254.

[52] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Worst-case optimal and average-case efficient geometric ad-hoc routing. In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc'03, Annapolis, MD, USA, June 1-3, 2003*, pages 267–278.

[53] OMNeT++: Simulation Environment. https://www.omnetpp.org, October 2018.

[54] Castalia: Wireless Sensor Network Simulator. https://github.com/boulis/Castalia, October 2018.

[55] Wendi B. Heinzelman, Anantha P. Chandrakasan, and Hari Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Trans. Wireless Communications*, 1(4):660–670, 2002.

[56] Joseph Polastre, Jason L. Hill, and David E. Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, SenSys'04, Baltimore, MD, USA, November 3-5, 2004*, pages 95–107.

[57] Wei Ye, John S. Heidemann, and Deborah Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Trans. Netw.*, 12(3):493–506, 2004.

Mohammed A. Merzoug — University of Bejaia 2019.
Contact: amine.merzoug@univ-batna2.dz

Document generated with LaTeX and:
1. The LaTeX style for Ph.D. Thesis created by S. Galland — http://www.multiagent.fr/ThesisStyle
2. The `tex-upmethodology` package suite — http://www.arakhne.org/tex-upmethodology/

Localized adaptive routing in wireless sensor networks

Abstract a wireless sensor network is made up of a set of small units, called sensor nodes, which can communicate with one another via direct radio communications or by relays. Sensor nodes are deployed in order to monitor a particular area of interest by sampling and collecting data (e.g., temperature, humidity, pressure, etc.). The collected information is subsequently sent to a fusion station for analysis purposes. The application areas of these networks are numerous in different domains such as military applications, domotics, health, etc. In reality, the specificities of wireless sensor networks bring many new possibilities but also pose many challenges in terms of deployment (limited energy resources, self-organization, etc.), coverage (monitored space), communication (total absence of any network infrastructure), and finally data collection and processing. In the context of this thesis, we are particularly interested in the general problem of data routing and fusion in this kind of networks. In fact, due to the inherent constraints of wireless sensor networks (primarily the very limited energy resources), this task remains particularly arduous despite the numerous works in the literature.

Keywords wireless sensor networks; distributed algorithms; localized routing; data fusion; query processing.

توجيه تكيفي محلي في شبكات الاستشعار اللاسلكية

ملخص تتكون شبكة الاستشعار اللاسلكية من مجموعة من الوحدات الصغيرة، المسماة عُقد الإستشعار، القادرة على التواصل مع بعضها البعض بواسطة الاتصالات اللاسلكية المباشرة أو غير المباشرة. تنشر عقد الاستشعار لمراقبة منطقة معينة عن طريق أخذ العينات وجمع البيانات (درجة الحرارة، الرطوبة، الضغط، الخ). يتم بعد ذلك إرسال المعلومات المجمعة إلى محطة التجميع والإدماج لأغراض التحليل. مجالات التطبيق لهذه الشبكات عديدة في مجالات مختلفة مثل التطبيقات العسكرية، التقنيات المنزلية، الصحة، الخ. في الواقع، خصوصيات شبكات الاستشعار اللاسلكية تجلب العديد من الامكانات الجديدة، ولكنها في نفس الوقت تطرح العديد من التحديات خاصة فيما يتعلق بالنشر (موارد الطاقة المحدودة، التنظيم الذاتي ، الخ)، التغطية (المساحة المراقبة)، الاتصالات (الغياب التام لأي بنية تحتية للشبكة)، وأخيرًا جمع البيانات ومعالجتها. في سياق هذه الأطروحة، نحن مهتمون بشكل خاص بالمشكلة العامة لتوجيه ودمج البيانات في هذا النوع من الشبكات. في الحقيقة، بسبب القيود المفروضة والمتأصلة في شبكات الاستشعار اللاسلكية (في المقام الأول، موارد الطاقة المحدودة للغاية)، تبقى هذه العملية صعبة للغاية على الرغم من العديد من الأعمال البحثية المنجزة.

الكلمات الدالة شبكات الاستشعار اللاسلكية؛ خوارزميات موزعة؛ توجيه محلي؛ دمج البيانات؛ معالجة الاستعلامات.

Routage localisé et adaptatif dans les réseaux de capteurs sans fil

Résumé un réseau de capteurs sans fil est constitué d'un ensemble de petites unités, appelées nœuds capteurs, pouvant communiquer les unes avec les autres via des communications radio directes ou par relais. Les nœuds capteurs sont déployés afin de surveiller une zone d'intérêt particulière par le prélèvement et la collecte de données (e.g., température, humidité, pression, etc.). Les informations ainsi collectées sont envoyées par la suite à une station de fusion à des fins d'analyse. Les domaines applicatifs de ces réseaux sont nombreux dans différents domaines tels que les applications militaires, la domotique, la santé, etc. En réalité, les spécificités des réseaux de capteurs sans fil amènent de nombreuses possibilités nouvelles mais posent également de nombreux défis en termes de déploiement (ressources énergétiques limitées, auto-organisation, etc.), de couverture (espace surveillé), de communication (absence totale de toute infrastructure réseau), et enfin de collecte et d'exploitation de données. Dans le cadre de cette thèse, nous nous intéressons plus particulièrement à la problématique générale de routage et de fusion de données dans ce genre de réseaux. En effet, en raison des contraintes inhérentes aux réseaux de capteurs sans fil (principalement les provisions énergétiques très limitées), cette tâche demeure particulièrement ardue malgré les nombreux travaux dans la littérature.

Mots-clés réseaux de capteurs sans fil; algorithmes distribués; routage localisé; fusion de données; traitement de requêtes.