Duty cycle aware spatial query processing in wireless sensor networks

Rone Ilidio da Silva, Daniel Fernandes Macedo, José Marcos S. Nogueira

1. Introduction

As wireless sensor networks (WSN) technology mature, its role as tool for periodic data gathering is expanding to data gathering in response to user’s queries. Several works model WSN as a distributed database and define energy-aware mechanisms for in-network query processing, such as TinyDB [18] and Cougar [8]. They process declarative SQL-like queries, which simplify the way users obtain data from the WSN. In-network query processing mechanisms can use location information to answer a special type of query, called spatial query. In these queries the users’ interests are expressed by geographical predicates. WSN can be modeled as distributed geodatabases in order to process queries containing spatial information, which request data collected at the region of interest (RoI) defined by the user.

In general, the works about spatial query processing in WSN consider that the sensor nodes are always on. However, nodes can enter in sleep mode (turn off their radio) in order to save energy. This work proposes an energy-efficient in-network spatial query processing mechanism that assumes nodes having no knowledge about their neighbors. The proposed mechanism is able to process spatial queries without the necessity of periodic beacon transmissions for neighbor table updates or for synchronization. Hence, it can work properly over different types of duty cycle algorithms.

This work proposes a new energy-efficient mechanism for in-network spatial query processing in WSN that works properly with different types of duty cycle algorithms. We assume that the user can start a query processing in any node of the WSN. In order to avoid periodic beacon transmissions to refresh the neighbor tables, we assume that the nodes know nothing about its vicinity. Hence, as part of this mechanism, we propose a location-based routing protocol called ABF (Ask Before Forwarding) to forward queries from the user to the RoI and to forward back query results. Moreover, we propose three algorithms to disseminate queries inside the RoI. All of the proposed algorithm consider RoIs having irregular shapes such as polygons, which can represent real objects plotted over maps or satellite photos. To our knowledge, only our previous work [25] treats the processing of queries that define RoIs with irregular contours, which require multiple points to be represented.

This article is organized into seven sections. Section 2 describes in-network query processing. Section 3 presents a review of the literature of duty cycle in WSN and spatial query processing. Sections 4 and 5 describe the proposed algorithms. Section 6 presents the simulations and analysis. Finally, Section 7 presents the conclusions and future works.

2. Spatial query processing in WSN

There are two basic types of spatial queries in WSN: window and KNN. In window queries, the user defines a RoI inside an area monitored by a WSN and asks the nodes inside this region to represent the highlighted regions need many points for their representation. Fig. 1 shows two satellite pictures of areas monitored by WSN. The highlighted regions need many points for their representation.
collect environmental data [25]. Fig. 2A exemplifies this type of query. Another type of query is called KNN (K Nearest Neighbor). The user defines a point inside the monitored area (called query point) and the value of K. The K nodes closest to the query point collect data [11]. Fig. 2B illustrates this type of query. In this work, we focus on window query processing.

Spatial query processing can be divided into six stages [25], as illustrated in Fig. 3. The Pre-Processing stage is performed in the user’s computer. It prepares queries to be processed by the WSN. The user, represented in Fig. 3 by the computer, defines the RoI. The first sensor node to receive the query in the network is named Originator. In the Forwarding stage, the query is forwarded from the Originator to a node inside the RoI. This last node in the Forwarding stage is called Coordinator. In the Dissemination stage, the query is disseminated from the Coordinator to all nodes inside the RoI. These nodes sense environmental data in the Sensing stage and send this information to be aggregated in the Aggregation stage. Finally, in the Return stage, the node with the query result forwards it to the Originator, which delivers it to the user’s computer.

3. State of the art

This section describes the duty cycle algorithms and spatial query processing mechanisms found in the literature.

3.1. Duty cycle in WSN

In the literature, there are many algorithms that control the node’s transition to sleep mode. Most of them are part of MAC protocols. These algorithms can be classified as synchronous or asynchronous. Synchronous algorithms put all nodes in sleep or active mode at the same time, such as those algorithms described in [9,29,33]. Asynchronous algorithms consider that nodes change to sleep mode periodically, but not all at the same time. Nodes can use preambles to wake up their vicinity [28,3,35], can send beacons to inform that they are active [30,12] or can maintain in their memory the sleep/wake schedules of one-hop neighbors [4]. Other algorithms use routing, location or density information to define which nodes will sleep and when they will wake up [32,38,2]. Moreover, some algorithms use random strategies to choose which nodes will sleep, such as in [24]. Recent works consider a new model of sensor nodes that maintains all nodes in sleep mode. They use RFID tags to turn on the radios in the vicinity before a transmission [14].

Most of the spatial query processing mechanisms do not consider nodes in sleep mode. The use of duty cycle algorithms requires periodic beacons in order to update the nodes’ neighbor tables. The spatial query processing mechanisms found in the literature use location-based routing protocols, such as Greedy [6] and GSPR [15]. These protocols obtain the next node to forward queries in the nodes’ neighbor tables. The duty cycle changes the data in these tables since it puts nodes in sleep mode. Modern geographic routing algorithms, such as CLDP, ECLDel and others [16,39,31], though, still present the same problem. They forward data using a combination of link quality estimation and location information, however there is a need to update the neighbor table, which may become invalid when a node fails or is turned down. Hence, WSN employing duty cycles and location based protocols need to spend additional energy updating data concerning the nodes’ vicinity.

3.2. Spatial queries

The existing spatial query processing mechanisms are classified into unstructured and structured. In the unstructured mechanisms, nodes possess information related to their neighbors only and no global structure is defined in the WSN. Structured mechanisms create indexes to help process the queries.

3.2.1. Unstructured spatial query mechanisms

In our previous work [26] we proposed a spatial query processing mechanism devised to detect and manage node failures during spatial query processing. Moreover, in [25] we proposed an energy-efficient mechanism that processes spatial queries with RoIs having irregular shapes. This type of region is important since it can represent real objects plotted by a Geographic Information System in maps or satellite photos. However, these works do not consider duty cycle algorithms changing the network topology. Resa is a window query processing mechanism proposed by Liu et al. [17]. It requires each node to maintain data about the node location, the link quality and the residual energy of each one-hop neighbor. This data is used to reduce the energy consumption, since it reduces the amount of packet retransmissions during the query processing. However, each node has to periodically exchange its energy level and to calculate the link quality of its neighbors. This process consumes a substantial part of the network energy mainly when the network topology is dynamic due to duty cycle algorithms.
SWIP (Spatial Window Processing) was proposed by Coman et al. [5,6]. The authors consider WSN deployed in a forest to collect environmental data. This data is periodically saved in the cache of each sensor node. A spatial query processing may start in any node. Users establish a spatial window having a rectangular shape and a bounded period of time for data collection. The main contribution of these works is the study of forwarding and dissemination algorithms in spatial query processing.

IWQE (Itinerary-based Window Query Execution) was proposed by Xu et al.[37]. The main contribution of this mechanism is the definition of a structure-free window query processing technique based on itineraries. The authors argue that structured mechanisms consume too much energy with structure maintenance in mobile WSN. Hence, unstructured mechanisms perform better in this kind of network. A drawback of this technique is the shape of the RoI. They assume rectangular regions of interest that facilitate the itinerary definition. Moreover, it uses GPSR [15] to forward queries and to return query answers. This protocol depends on the neighbor table, which needs to be periodically refreshed, mainly when used in WSN employing duty cycle algorithms.

3.2.2. Structured spatial query mechanisms

Mohamed and Khokhar [20] describe an algorithm that creates indexes in WSN to save energy during query processing. This algorithm recursively divides the monitored area into hierarchical sub regions containing the same number of nodes (which is defined by the user). Each sub region becomes a cluster and cluster-heads know the area covered by the nodes in their cluster. All nodes send their sensed data to their cluster-heads and the cluster-heads send this data to the immediately superior cluster-heads. However, the algorithm does not consider that cluster-heads employ duty cycles. Hence, if one of these nodes goes to sleep, all data collected by their descendants will be lost.

SPIX (SPatial IndeX) was presented by Soheili et al.[27]. The main contribution of this work is the definition and optimization of a distributed spatial index based on MBR (Minimum Bounding Rectangle), the smallest rectangle that contains the node and all its descendants in the routing tree. After routing tree creation, each node performs a parent selection in order to optimize the index. Moreover, the mechanism has a maintenance phase, in which each node periodically verifies its parent/children links. A drawback of this mechanism is the energy consumption under node failure. If a node leaves its parent and the MBR changes, this change must be propagated node by node up to the sink.

Cluster and index was described by Park et al.[22]. The main contribution of this work is the definition of a semi-distributed spatial index. The WSN are divided into square sub-regions. Nodes in the same sub-region form a cluster. All cluster-heads know the location of the nodes within their clusters. They aggregate and transmit the data collected by all nodes in their clusters. The cluster-heads form an index based on MBR in order to transmit inter-cluster data. The authors do not consider duty cycle algorithms, hence a drawback of this mechanism is the energy consumption to maintain the cluster information in the cluster-head under duty cycle.

4. ABF – Ask Before Forwarding Protocol

We propose ABF, a location-based routing algorithm used to forward queries from the Originator to the Rol (Forwarding stage) and to forward back the query result from the Rol to the Originator (Return stage). The last node in the Forwarding stage and the first node in the Return stage depend on the dissemination algorithm. When we use Restricted Flooding in the Dissemination stage, the Coordinator is the first node to receive the query inside the Rol. The queries are forwarded towards the centroid of the Rol, here called reference point. When an itinerary-based algorithm is used, the reference point is the leftmost point (LMP) of the Rol. The Coordinator is a node having distance to LMP smaller than ID (Itinerary Distance: unit of distance define by the user, described with more details in Section 5.3.1). This strategy helps the itinerary definition, as shown in the next section. Fig. 4A illustrates this process for Restricted Flooding and Fig. 4B illustrates this process for Itinerary.

4.1. ABF strategy

Nodes using ABF first ask for a candidate and then forward the query or the query result to this candidate. Since we assume nodes having no knowledge about their vicinity (they can sleep due to the duty cycle), the sender needs to look for an active neighbor able to receive and forward packets. Hence, the sender transmits a request-packet message in order to find candidates. All neighbor of the sender receive the request-packet. Each neighbor schedules the transmission of an announce-packet message to the sender informing that it is active. This transmission is delayed for a period of time proportional to the node distance to the reference point. The sender sends the query to the neighbor that sent the first announce-packet received, which will repeat this process. When a neighbor receives a query-packet destined to another node, it cancels the announce-packet transmission before scheduled in order to avoid unnecessary packet transmissions. Fig. 5 illustrates the protocol operating during the Forwarding stage.

Fig. 6 presents a state diagram of a node that holds a query to be forwarded (Holding Query state). It sends a request-packet to its neighbors and waits for announce-packages (Waiting Announces state). If no announce-package is received, the node waits a period of time and resends the request-packet. After receiving the first announce-package, the node goes to the Sending Query state. During this state, this node sends all the query-packets (one query can be composed of more than one packet) to the node that sent the announce-package. Then, after it finishes the query transmission, the sender goes to the Idle state. Subsequent announce-packages are ignored.

Fig. 7 presents the state diagram of the receiver. If a node in the Idle state receives a request-packet, it goes to the Waiting $T_{wait}$ state ($T_{wait}$ is defined in the next subsection), stays in this state for $T_{wait}$ seconds, sends an announce-package and goes to the Waiting Query state. During the Waiting $T_{wait}$ state, if the node receives a query-packet and the query is not directed to it, this node cancels the announce-package transmission and goes to the Sleeping state in order to avoid overhearing. After sending an announce-package, if this
Fig. 4. Reference point in Forwarding: (A) for Restricted Flooding; (B) for Itinerary.

Fig. 5. Forwarding stage. (A) Sender transmits a request-packet. (B) The active neighbors of the sender delay the transmission of announce-packet. (C) The neighbor closest to the RoI sends the first announce-packet. (D) The sender forwards the query and their neighbors cancel the announce-packet transmission.

Fig. 6. State diagram of a forwarder node.

Fig. 7. State diagram of a node that received a request-packet.
node receives a query-packet directed to it, it will go to the Receiving Query state and stays in this state until it receives all the query-packets.

Using this strategy, ABF is able to cope failures caused by interference or node damage. These failures increase the number of lost packets. If a request-packet is lost by a neighbor of the sender, another neighbor will receive it and will send its announce-packet. Moreover, if an announce-packet is lost, the sender will receive announce-packets from other neighbors. However, this work focuses on WSN employing duty cycle algorithms. Our previous work [26] analyzed scenarios employing failures.

4.2. Delay to answer the request-packet

Nodes waits $T_{Wait}$ before answering a request-packet. $T_{Wait}$ is proportional to the node’s distance to the reference point. Let’s call $S$ the sender, $R$ the reference point, $D$ a neighbor of $S$ and $r$ the radio range. We define as ideal position (IP) a point that is $r$ meters distant of $S$ and is on the line segment $SR$. Fig. 8 illustrates these definitions. Since the request-packet has the location of $S$ and $R$, all nodes that receive it can calculate IP. The value of $T_{Wait}$ is based on the distance between the node and IP. We define the following equation to calculate $T_{Wait}$:

$$T_{Wait} = \frac{d}{r} + T_{Max}$$

where $d$ is the distance between $D$ and IP and $T_{Max}$ is the highest value of $T_{Wait}$. Using this equation, nodes close to IP send their announce-packets before nodes far from IP. Thus, these nodes have a higher probability of being the next node to forward the query.

4.3. Avoiding holes in the network

Holes in the network happen when a node needs to send a packet to a destination and it has no neighbor closer to the destination than itself [40]. Fig. 9 illustrates this situation. In order to avoid holes in the network, a node that receives a request-packet and is farther from IP than $S$ uses the following equation to calculate $T_{Wait}$:

$$T_{Wait} = T_{Max} + \frac{dp}{r}$$

where $r$ is the radio range and $dp$ is the distance of the projection of the node location on the line $SR$. Fig. 9 hows how $dp$ is calculated. Node $i$ defines a line $L_i$ that is perpendicular to $SR$ and passes on $i$’s location. $P_i$ is the point where $L_i$ crosses $SR$. $dp$, is the distance between $P_i$ and IP. Using this equation, nodes far from IP than $S$ have the biggest values for $T_{Wait}$. Hence, $S$ chooses these nodes only when it has no neighbor closer to IP than itself. Moreover, using this strategy the query can surround holes and the nodes do not need know data about the network topology.

4.4. Forwarding failure

A forwarding failure happens when a node cannot forward the query and drops it. Each node maintains two lists of nodes: nodes to whom it sent the query and nodes from whom it received the query. When a node receives a query more than once, it sends the request-packet normally, but it waits for an announce-packet sent by a node not on its lists. Moreover, nodes only send announce-packets to nodes out of its lists. Hence, a node can receive a query more than once from different sources, but it just sends or receives it once for each of its neighbors. This guarantees that queries can find loops during the Forwarding stage, but loops are followed just once. Fig. 10 illustrates a situation where a node identified a forward failure. This node has three neighbors, but it cannot send the query to them because it sent the query to one and received the query from the other two.

5. Dissemination and aggregation stages

In the Dissemination stage, the query is transmitted to all nodes inside the RoI. In the Aggregation stage, the readings collected by these nodes are transmitted to a node (called Aggregator) in order to aggregate these readings and to calculate the query result. We propose three algorithms for the Dissemination and Aggregation stages: Classic Restricted Flooding, Delayed Restricted Flooding and Itinerary. The first two are based on Restricted Flooding, but use different strategies to aggregate the nodes’ readings. The last algorithm creates itineraries to transmit queries.

5.1. Classic Restricted Flooding

This algorithm is called “Classic” because it is based on Flooding [1]: each node receives the query and rebroadcasts it. However, in Restricted Flooding only nodes inside the RoI broadcast the query. During query dissemination, each node saves as its parent the node that first sent the query to it. This operation creates a structure
called Routing Tree [40], which is used in the Aggregation stage. A node i puts the identifier of its parent p into the query it is disseminating in order to notify p that i is its son. Hence, nodes having no children are leaf nodes in the routing tree. A leaf node only collects environmental data and sends it to its parent. The other nodes collect environmental data, receive the data from their sons, apply the environmental data and send it to its parent. The other nodes collect environmental data and sends it to their parents. However, if they received a query from a son, they will send their aggregation-packet after receiving readings from all their descendents or after $T_{Sand}$ seconds.

![Node has no neighbor to send the query](image)

**Fig. 10.** Forward failure.

Algorithm 1 is executed when a node receives a query to be disseminated. Line 2 verifies if the node is inside the RoI; if not, the query is dropped (line 13). A node inside the RoI verifies if this query is a new query (line 3). If so, the node that sent the query is saved as old query (line 5), it is disseminated (line 6) and the node starts a $T_{Log}$ timer (line 7). If the query is not new, the node verifies if the parent of the source node is itself and saves this node as its son (line 9).

![Algorithm 1: Classic Restricted Flooding Dissemination](image)

**Algorithm 1: Classic Restricted Flooding Dissemination**

1: procedure $RECEIVE_{- QUERY}(qry)$
2:     if in_polygon(qry.roi) then
3:         if qry.id $\neq$ old then
4:             parent_id -- qry.source
5:             old -- old $\cup$ qry.id
6:             resend_query(qry)
7:             $t_{Log}$ - start()
8:     else if qry.parent = my_id then
9:         son_id -- son_id $\cup$ qry.source
10:     else
11:         drop(qry)
12:     end if
13: end procedure

Nodes use tables of sons and two timers ($T_{Sons}$ and $T_{Sons}$) to perform the Aggregation stage. $T_{Log}$ is a timer used in order to identify if a node is a leaf node or not. $T_{Sons}$ defines the period of time a node waits for their sons' readings. Fig. 11 presents a state diagram that illustrates the Aggregation stage. After disseminating a query, nodes need to know how many sons they have. So, they wait for their sons' queries for $T_{Log}$ seconds. After it, if they have no sons, they are leaf nodes and will immediately send readings to their parents. However, if they received a query from a son, they will send their aggregation-packet after receiving readings from all their descendents or after $T_{Sand}$ seconds.

![State diagram of the Aggregation Stage for Classic Restricted Flooding](image)

**Fig. 11.** State diagram of the Aggregation Stage for Classic Restricted Flooding.

When a node receives an aggregation-packet after sending its reading, it sends to its parent another aggregation-packet containing only the received data. We call these packets as late aggregation packets - LAP. It avoids losing readings received after a node has sent the aggregation-packet.

5.2. Delayed Restricted Flooding

Delayed Restricted Flooding disseminates the query as in the previous algorithm, but the aggregation is performed in a different way. This algorithm was created in order to avoid packet losses during the Dissemination stage. Based on Hora et al. [13] and our experiments, we verified that algorithms based on Flooding are prone to packet collisions, since all neighbors of a node try to transmit the query at the same time. Thus, due to collisions, the number of perceived sons of a node is usually smaller than the number of actual sons. Consequently, nodes performing Classic Restricted Flooding may send aggregation-packets before receiving all readings from their sons.

Nodes performing the Delayed Restricted Flooding algorithm do not create lists of sons. The algorithm defines the coordinator timer $T_{Coord}$ on the Coordinator and the aggregation timer $T_{Agg}$ on the other nodes. After disseminating a query, each node starts its timer. When $T_{Agg}$ expires, nodes send to their parents the result of the aggregation operator in the aggregation-packet. When $T_{Coord}$ expires, the Coordinator calculates the query result and starts the Return Stage. The value of $T_{Coord}$ is calculated using the following equation:

$$T_{Coord} = T_{Pred_{dis}} + T_{Pred_{agg}}$$  \(3\)

where $T_{Pred_{dis}}$ is the predicted time to disseminate the query and $T_{Pred_{agg}}$ is the predicted time to aggregate. $T_{Pred_{dis}}$ is calculated by the following:

$$T_{Pred_{dis}} = predicted_{levels} * T_{send} * query_{packets}$$  \(4\)

$T_{send}$ is the approximate time to send a packet. We obtained $T_{send} = 0.1$ s empirically in our simulations. $query_{packets}$ is the number of packets needed to hold the query. $predicted_{levels}$ is the predicted number of levels in the routing tree created during the dissemination. This last value is obtained by dividing the length of the RoI by the radio range and multiplying by 2 (this is an adjust-
ment value obtained empirically). $T_{\text{Pred}_\text{Agg}}$ is calculated by the following equation:

$$T_{\text{Pred}_\text{Agg}} = \text{predicted levels} \times T_{\text{Decrease}}$$

where $T_{\text{Decrease}}$ is a default decrease value. $T_{\text{Agg}}$ is based on node's level in the routing tree. Since the Coordinator has the longest timer ($T_{\text{Coord}}$), the other nodes calculate $T_{\text{Agg}}$ using the following equation:

$$T_{\text{Agg},i} = T_{\text{Coord}} - \text{Level}_i \times T_{\text{Decrease}}$$

where $T_{\text{Agg},i}$ is the timer of node $i$ and $\text{Level}_i$ is the level (in hops) of this node in the routing tree. The best value for $T_{\text{Decrease}}$ was empirically obtained by simulation.

Usually, all descendents of a node try to send the aggregation-packet at the same time. Thus, these packets may collide. In order to avoid it, each node waits a small random period of time before sending the aggregation result. This period of time varies between 0% and 40% of $T_{\text{Decrease}}$, a value empirically obtained.

Fig. 12 presents the state diagram of Delayed Restricted Flooding. Assume that node $X$ is “Holding a Query”. It sends the query to its descendents and starts the timer $T_{\text{Agg}}$. While this node is “Waiting $T_{\text{Agg}}$”, it receives the readings collected by its descendents. So, after $T_{\text{Agg}}$ expires, it waits a random period of time, aggregates its collected data with the data sent by its descendents and sends the aggregation-packet to its parent.

5.3. Itinerary

This algorithm defines an itinerary within the RoI. It has a zigzag pattern, as illustrated in Fig. 13. Since we consider RoIs having irregular shapes, this pattern is the most flexible among the patterns found in the literature [37]. Moreover, since nodes are deployed randomly, the path shown in Fig. 13 is only the ideal itinerary used as a basis to generate the real itinerary.

The itinerary definition follows the ABF strategy, that is, it looks for active neighbors and chooses one of them to continue the query processing. However, a node on the itinerary does not use announce-packets. It sends the query to its neighbors and receives the packets with their readings (reading-packets). Fig. 14 presents a state diagram that illustrates the process in a node holding a query on the itinerary. First, it receives an aggregation-packet and goes to the Holding Query state. Thus, it broadcasts all the query-packets to its neighbors during the Sending Query state. Hence, it stays in the Waiting Packet-readings state for $T_{\text{Wait}}$ seconds, chooses the best neighbor and sends the aggregation-packet to the next node on the itinerary. $T_{\text{Wait}}$ is the same timer used by ABF.

5.3.1. Itinerary terminology

Before describing the algorithm, it is necessary to define some of the terms employed here. Fig. 13 illustrates them. The Itinerary Distance (ID) is an adjustable parameter in the algorithm that defines the distance between parallel lines of the itinerary ($2 \times \text{ID}$) and between the itinerary and the boundaries of the RoI. The itinerary starts from left to right (other directions are possible with simple rotations in the $X$ and $Y$ axis). We devise parallel guide lines, called reference lines, over which the itinerary will pass. The point where these lines cross the RoI is called Destination In the Itinerary (DII). The Coordinator starts the itinerary. This node is chosen among the nodes that are inside the RoI and its distance from the Leftmost Point (LMP – the point in the RoI with the smallest $X$ value) is less than ID. The Last Node on the Itinerary (LNI) finishes the itinerary. It is defined by the halting criteria defined in the following subsection.

A node chooses as the next node on the itinerary the neighbor closest to the current DII contained in the query. Moreover, when a node is closer to DII than the distance ID, it changes the direction of the itinerary and calculates another DII. In order to do so, the following data needs to be disseminated: points that define the RoI, the location of the Originator, Direction and the current DII. The Location of the Originator is the point where the query starts processing, and is disseminated because the LNI will forward the query to this point. Direction defines the current direction of the itinerary: UP, DOWN or RIGHT (the direction is represented as arrows in Fig. 13).

5.3.2. Itinerary algorithm

Algorithm 2 is executed when $T_{\text{Wait}}$ expires and a node nodes on the itinerary must select the next node among its neighbors. First,
it verifies if its distance to the RMP is smaller than the value of $ID$. If so, this node is the LNI and will start the Return stage (lines 2 and 3). If not, the node verifies its distance to the current DII (line 5). If this distance is smaller than $ID$, the node changes the current direction and DII (lines 6 up to 15), chooses the next node (line 16) and sends the aggregation-packet (line 17).

The functions $next\_down()$, $next\_up()$ and $next\_right()$ define new DIIIs. They are responsible for the zigzag pattern of the itinerary. The new DII is a point where the reference line crosses the boundaries of the RoI. The reference line is a perpendicular line that crosses the RoI twice, in an upper and in a down point. The function $next\_down()$ maintains the reference line and defines the new DII as the down point. The function $next\_right()$ also maintains the reference line and defines the new DII as the upper point. The functions $next\_right()$ defines a new reference line in a distance of $2 \times ID$ from the current DII. The new DII is the closest point from the current DII, between the two points on the reference line that crosses the RoI.

The algorithm has two halting criteria: ideal and failure. The ideal criterion happens when the query reaches a node which its distance to RMP is smaller than $ID$. It means that the query covered the entire RoI. The failure criterion is defined in order to avoid loops in the itinerary. It finishes the itinerary before the query reaches the whole RoI, compromising the coverage. This criterion happens when the aggregation-packet reaches a node more than once for the same DII and all of its neighbors were already chosen as the next node on the itinerary for this DII.

![Fig. 14. State diagram: dissemination using itinerary.](image)

### Algorithm 2: Next node itinerary definition

```plaintext
1: procedure next_node_itinerary
2: if my_distance_to(RMP) < ID then
3:     start_return_stage()
4:     return
5: else if my_distance_to(qry.dii) < ID then
6:     if qry.direction = UP ∨ qry.direction = DOWN then
7:         qry.dii ← next_right(qry.dii)
8:     else if qry.direction = RIGHT then
9:         if closest_top(qry.dii) then
10:            qry.dii ← next_down(qry.dii)
11:            qry.direction ← DOWN
12:        else
13:            qry.dii ← next_up(qry.dii)
14:            qry.direction ← UP
15:        next_neighbor ← choose_neighbor()
16:        send_aggregation_packet(next_neighbor)
17:     return
```

### 6. Simulated experiments

The proposal was evaluated by simulations using the network simulator NS 2.34 [21]. All packet transmissions are in broadcast. Hence, all the neighbors of the sender receive the packet. The simulation parameters simulate Iris sensor node [7] running the BMAC [23] medium access control protocol. Its parameters are presented in Table 1.

We described three Dissemination/Aggregation algorithms in Section 5. Thus, we implemented three variations of the proposed mechanism. All of them use in their Forwarding and Return stages the ABF location-based protocol, but each of them uses one of the Dissemination/Aggregation algorithms proposed. These variations are called as follows:

- Classic: uses Classic Restricted Flooding;
- DRF: uses Delayed Restricted Flooding, and;
- Itinerary: creates an itinerary on the RoI.

We consider the uniform random deployment of the nodes on a square area having one square kilometer. We assume that each node is able to obtain its location [36]. We use coordinates in the standard decimal degrees notation with four decimal places, giving a worst case accuracy of around 11 m.

We also created an application in Java that enables the user to generate the RoI on satellite photos and performs the Pre-Processing stage. This application implements the Douglas-Peucker [34] algorithm in order to reduce the number of points in the RoI representation, thus decreasing the number of packets necessary to transmit the query. Due to the infinite possibilities for regions of interest, we chose a real contour, that of the lake in Fig. 1(A) represented by 140 points, requiring 10 packets of 28 bytes to be transmitted. The experiments ran the Pre-Processing algorithm in order to reduce the query size from 10 up to 1 packet. Hence, the first graphics presented ahead have the X-axis varying between 1 and 10. This algorithm was described in our previous work [25]. Every point plotted on the graphics represents the mean of 60 simulations, with confidence interval of 95%.

### Table 1: Simulation parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radio range</td>
<td>80 m</td>
</tr>
<tr>
<td>Sensing range</td>
<td>40 m</td>
</tr>
<tr>
<td>TX power</td>
<td>0.0484 W</td>
</tr>
<tr>
<td>RX power</td>
<td>0.051 W</td>
</tr>
<tr>
<td>Battery voltage</td>
<td>3 V</td>
</tr>
<tr>
<td>Payload length</td>
<td>28 bytes</td>
</tr>
<tr>
<td>Number of nodes</td>
<td>2000</td>
</tr>
</tbody>
</table>
In this work, we employed a simple duty cycle control algorithm based on asynchronous MAC protocols, such as [23,3,10,30]. All nodes have the same sleep \((T_{\text{sleep}})\) and active times \((T_{\text{on}})\), but they operate in different phases. Further, the length of a cycle \(T_{\text{cycle}} = T_{\text{off}} + T_{\text{on}}\). In our simulations, during the network startup all nodes choose randomly when they will start their cycles. The experiments were performed in two phases. In the first phase, we evaluate the best parameter values for each proposed algorithm. In the second phase, we vary the duty cycle and compare the algorithms using their best configuration against one spatial query processing mechanism found in the literature.

### 6.1. Metrics

The experiments analyze four metrics: coverage area, end-to-end delay, energy consumption and query processing success. The coverage area is the portion of the RoI that is covered by one or more sensor nodes. We consider that the sensing range of the nodes is half of the radio range [19]. The end-to-end delay is the time elapsed from the Originator receiving the query from the user and the reception of the query result. It measures the transmission time and delays of the mechanism. For simplicity, we do not consider the time used for processing the query inside each node. Energy consumption is the energy consumed by the WSN as a whole in order to process one query. We do not consider the energy consumed when nodes are in sleep or idle states. In our conception, MAC protocols should manage the energy consumption during these states. The query processing success is the percentage of query processing that obtained a result among all the queries processed by the WSN. Hence, this metric represents the quality of the answer.

### 6.2. Evaluating Restricted Flooding

These experiments compare the Classic Restricted Flooding (Classic) against Delayed Restricted Flooding (DRF). They also present the performance of Delayed Restricted Flooding using different values of \(T_{\text{Decrease}}\) (Section 5.2) in order to define the best configuration for this parameter. We use sleep time \(T_{\text{off}} = 0.5\) s and active time \(T_{\text{on}} = 0.5\) s for the duty cycle.

#### 6.2.1. Classic versus Delayed Restricted Flooding

In this scenario, we compare Classic against DRF using \(T_{\text{Decrease}} = 0.5, 1.0\) and \(2.0\) s. It is important to mention that Classic uses \(T_{\text{sleep}} = 5\) and \(T_{\text{on}} = 5\) s. These values were obtained empirically. Fig. 15 presents the end-to-end delay. The experiments show that Classic processes queries faster than DRF in practically all scenarios. It is because the timers used by Classic are smaller than the timer defined by the Coordinator \((T_{\text{Co}})\), the longest timer used by DRF. Moreover, nodes using Classic can send aggregation-packets immediately after receiving all the aggregation-packets from their descendents. Using DRF, nodes wait \(T_{\text{Agg}}\) seconds before sending aggregation-packets. In the following graphics we verify the trade-off between end-to-end delay and coverage area and between end-to-end delay and energy consumption.

Fig. 16 presents the coverage area. DRF 2.0 presented the best coverage area for all the analyzed query sizes. DRF 1.0, DRF 0.5 and Classic had their performance decreased due to packet collisions caused by the increased number of late packets (LAPs, Section 5.1). Classic is propitious to collision since all nodes in a neighborhood try to retransmit the query at the same time. It decreases the number of descendents in the nodes’ tables of sons, increasing the number of LAPs. When DRF uses small values of \(T_{\text{Decrease}}\), several nodes do not define large enough aggregation timers. Hence, these nodes do not wait for all their descendents’ packets before sending the aggregated readings. Thus, the number of LAPs in DRF 0.5 and DRF 1.0 increases for queries higher than 2 and 5 packets, respectively. Moreover, DRF using small values of \(T_{\text{Decrease}}\) (0.5 and 1.0 s) also increases the number of collisions during the Aggregation stage. It happens due to the random delay used to avoid collisions before sending the aggregation-packet (Section 5.2). This delay is proportional to \(T_{\text{Decrease}}\), so the collisions tend to increase if \(T_{\text{Decrease}}\) is not large enough.

LAPs also increase the energy consumption of the query processing. If \(T_{\text{Decrease}}\) is large enough, all nodes will wait for all their descendents readings and will transmit aggregation-packets only once. However, if \(T_{\text{Decrease}}\) is small, nodes can transmit several LAPs, increasing the energy consumption. Fig. 17 presents the energy consumption. DRF 0.5 and 1.0 present low energy consumption when the query is small. However, when the query size increases, the value of \(T_{\text{Decrease}}\) is not sufficient for receiving the readings of all the descendents. Hence, the number of LAPs increases. The increase of the energy consumption accentuates for queries using more than two and five packets in DRF 0.5 and 1.0, respectively. Since the number of LAPs is small for DRF 2.0, the energy consumption is also small.

This scenarios shows that DRF using \(T_{\text{Decrease}} = 2.0\) presents the best coverage area and energy consumption in practically all scenarios. Classic presents the smallest end-to-end delay, however the amount of collisions increases its energy consumption and decreases its coverage area. Hence, we verify that DRF 2.0 has the best performance, however it spends more time to process queries. In experiments not presented here, we increased the value of \(T_{\text{Decrease}}\) to 3, 4 and 5 s. The end-to-end delay increased during these experiments, however the energy consumption and coverage area were practically the same.

### 6.3. Evaluating the Itinerary

These experiments were performed in order to evaluate the best configuration of the Itinerary algorithm defined in Section 5.3.
Here, we also use \( T_{off} = 0.5 \text{ s} \) and \( T_{on} = 0.5 \text{ s} \) for the duty cycle. The parameters that most affect the performance of the query processing are the itinerary distance (ID – Section 5.3.1) and the maximum delay to send an announce-packet (\( T_{Max} \) – Section 4.2). ID defines the distance between two parallel lines in the itinerary as well as the distance between the itinerary and the bounds of the RoI. The value of \( T_{Max} \) influences the end-to-end delay and the energy consumption, since larger values of \( T_{Max} \) reduce the amount of announce-packets.

### 6.3.1. The value of \( T_{Max} \)

Fig. 18 presents the end-to-end delay of the itinerary using different query sizes. When \( T_{Max} \) and the query size increase, the end-to-end delay also increases, as expected. \( T_{Max} \) also influences the processing success. The figure shows that \( T_{Max} \) using 0.2 and 0.4 s are not able to process queries bigger than 4 and 8 packets, respectively. If \( T_{Max} \) is small, no child node can send readings in time. So, a node can consider that it has no neighbor to forward the query, hence the query processing finishes without a result.

The coverage area of the query processing is analyzed in Fig. 19. Queries processed using different values of \( T_{Max} \) presented practically the same coverage area. The performance of the itinerary algorithm decreases only for \( T_{Max} = 0.2 \) and \( T_{Max} = 0.4 \) when the query size is bigger than 4 and 8, respectively. It occurs because the value of \( T_{Max} \) is not enough for all nodes to receive announce-packets during the Forwarding and Return stages, halting the query processing.

Fig. 20 presents the energy consumption. \( T_{Max} = 1.0 \text{ s} \) has the smallest consumption for small queries, because this value minimizes the number of announce packets. Nodes holding a query send it to the first node that sent an announce-packet. When a node has an announce-packet scheduled to be sent and it hears a query not directed to it, this node cancels the packet transmission (Section 4). Hence, large values of \( T_{Max} \) increase the amount of announce-packet cancellations, consequently reducing the energy consumption.

We use \( T_{Max} = 1.0 \text{ s} \) for the remainder of the experiments. This value increases the end-to-end delay, but presents acceptable energy consumption and coverage area. We also performed experiments using bigger values of \( T_{Max} \). The energy consumption and coverage area do not change when compared to \( T_{Max} = 1.0 \text{ s} \). These experiments are not presented to improve the readability of the article.

### 6.3.2. The value of Itinerary Distance – ID

Here we evaluate the impact of the parameter ID (Section 5.3.1) in the performance of Itinerary. The end-to-end delay of the mechanism is presented by Fig. 21. Increasing the value of ID, the end-to-end delay decreases because this parameter defines the distance between parallel lines in the itinerary. Hence, if ID increases, the
path followed by the query during the dissemination will decrease, thus decreasing the end-to-end delay. Another characteristic shown by Fig. 21 is that the query size does not influence significantly the end-to-end delay.

The coverage area is presented by Fig. 22. Small values of ID provide better coverage since the parallel lines in the itinerary are closer to each other. Hence ID = 60 m presents the best performance. When ID = 120 m, the mechanism presents the smallest coverage area. It occurs because queries cannot reach some sections of the RoI due to the large distance between parallel lines of the itinerary. ID = 100 m and ID = 80 m presented practically the same performance.

The energy consumption is directly affected by ID (see Fig. 23). Using higher values of this parameter, the path followed by the query decreases, reducing the energy consumption. Analyzing these graphics we verify that ID = 100 m is the best value of this parameter since it presents an acceptable end-to-end delay, energy consumption and coverage area.

6.4. Comparing the mechanisms

These experiments were performed in order to compare the proposed mechanism and its variations against the state of the art. Moreover, we evaluate their performance in different duty cycles. Three algorithms are presented in the graphics: DRF, Itinerary and SWIP-2. SWIP-2 is a spatial query processing mechanism based on SWIP [6]. We chose SWIP because it is the unique window query processing that we found in the literature that starts the process in any node. However, it does not consider RoIs with irregular shapes, duty cycle algorithms or packet collisions. SWIP uses a greedy strategy to forward the query to the RoI. However, this algorithm does not cope with holes in the network (definition in Section 4.3). In the Dissemination/Aggregation stages, it uses Classic and in the Return stage it uses the reverse route created during the Forwarding stage.

SWIP-2, on the other hand, uses GPSR [15] in the Forwarding stage, because this protocol can surround holes in the network. In the Dissemination/Aggregation stage it uses Delayed Restricted Flooding, since Classic does not present a good performance. In the Return stage, it creates a new route using GPSR. The reverse route created in the Forwarding stage cannot be used because nodes can sleep. Moreover, all nodes using SWIP-2 transmit a beacon every 30 s in order to update the neighbor table used by GPSR. The original paper of GPRS [15] established 3 s as beacon interval, however we increased the beacon interval since our network is static.

During these experiments, we fix $T_{\text{Cycle}} = 1$ s and vary $T_{\text{on}}$ from 0.1 to 1.0 s. Thus, we vary the duty cycle from 10% to 100%. The $X$-axis of all graphics represent this variation. In order to evaluate all stages of the spatial query processing, we fixed the Originator in the top left corner of the monitored area. It forces the mechanisms to execute the Forwarding and Return stages, otherwise the Originator could be inside the RoI and start processing queries in the Dissemination stage.

Fig. 24 presents the query processing success of the three algorithms. SWIP-2 with duty cycles smaller than 60% is not able to perform the Forward and Return stages since the query and the query result are often sent to sleeping nodes and, consequently, the query processing fails. DRF and itinerary presented more than 90% of query processing success in practically all scenarios, since they use only active nodes. These mechanisms are not able to process queries only when they reach a forward failure (Subsection 4.4). In the following graphics, the line that represents SWIP-2 starts in 60% since there is no query processed with smaller duty cycles.

Fig. 25 shows the end-to-end delay of the three algorithms. Itinerary presents the worst performance since the $T_{\text{Wait}} = 2$ s used in Forwarding, Dissemination and Return stages increases the time necessary to forward the query and the query result. DRF presents the smallest end-to-end delay because it uses ABF in the Forwarding and Return stages and does not need to transmit periodic beacons. SWIP-2 needs to transmit beacons during the query process in order to update the neighbor tables of GPSR. Hence, it suffers from congestion, which increases the amount of packet collisions during the Dissemination stage, increasing the end-to-end delay.

The coverage area of the three algorithms is shown by Fig. 26. SWIP-2 presents good performance with 50% and 60% of duty cycle. When the duty cycle increases, the amount of packet collisions increases too, since more nodes are active during the query processing. Hence, the coverage area decreases. DRF presents small coverage area with duty cycle in 10% because there are few nodes active to disseminate the query. However, with duty cycle in 20% more nodes stay active during the query dissemination. Thus, the
coverage area grows to around 90% and maintains this value for larger duty cycles. The coverage area of Itinerary increases if the duty cycle increases, since there are more nodes active to create the itinerary. However, this mechanism presents coverage larger than 90% only for duty cycles bigger than 30%. When small duty cycles are used the nodes have less active neighbors. Hence, occurs more often prematurely stopping due to the second halting criterion (Section 5.3.2).

The energy consumption is presented in Fig. 27. The beacon transmissions increase the consumption of SWIP-2 since nodes that do not participate in the query processing also transmit and receive packets. Using DRF and Itinerary, only nodes that participate in the query processing transmit packets. Itinerary presents acceptable coverage area and energy consumption, even in scenarios in which nodes stay only 20% of the time active. Itinerary presented acceptable performance in WSN in which nodes stay more than 30% of the time active. This algorithm presented the worst end-to-end delay, however it presented the best energy consumption. It shows a tradeoff between energy consumption and end-to-end delay.

In future works, we intend to consider different types of failures during the spatial query processing. Failures in WSN occur due to many reasons, such as interference from simultaneous transmissions, energy depletion or node failure. Damage can compromise the query processing since part of the WSN can be inaccessible. Moreover, we intend to consider imprecision in nodes' position because all the location methods for WSN found in the literature presents non negligible imprecision [36]. Finally, we intend to implement the proposed mechanism on real sensor nodes and compare the simulation results against the real implementation.

7. Conclusions and future works

This work presented an energy-efficient spatial query processing mechanism for WSN. It was created in order to work properly on networks using different types of duty cycle algorithms. Since the literature presents several duty cycle algorithms, the proposed mechanism assumes that nodes are unaware of their neighbor's sleep schedules. Hence, there is no need of periodic beacons for node synchronization or to update neighbor tables.

The spatial query process was divided in stages. For the Forwarding and Return stages, we proposed a new location-based routing protocol called ABF, which uses only active nodes to forward packets. In the Dissemination and Aggregation stages we proposed three algorithms. Thus, the mechanism has three variations: Classic, DRF and Itinerary.

The experiments were performed in two phases. In the first phase, we analyzed the best parameter values for all the proposed algorithms. In the second phase, we varied the duty cycle and compared these algorithms against SWIP-2, a improved version of SWIP [6]. The first phase of the experiments showed that Classic suffered from packet collision during the Dissemination and Aggregation stages, hence it did not present a good performance in most of the experiments. In the second phase, DRF and Itinerary consumed only 25% and 15% of the energy consumed by SWIP-2, respectively. We verified that SWIP-2 finishes less than 10% of the queries when the duty cycle is smaller than 50%. DRF presented acceptable coverage area and energy consumption, even in scenarios in which nodes stay only 20% of the time active. Itinerary presented acceptable performance in WSN in which nodes stay more than 30% of the time active. This algorithm presented the worst end-to-end delay, however it presented the best energy consumption. It shows a tradeoff between energy consumption and end-to-end delay.

In future works, we intend to consider different types of failures during the spatial query processing. Failures in WSN occur due to many reasons, such as interference from simultaneous transmissions, energy depletion or node failure. Damage can compromise the query processing since part of the WSN can be inaccessible. Moreover, we intend to consider imprecision in nodes' position because all the location methods for WSN found in the literature presents non negligible imprecision [36]. Finally, we intend to implement the proposed mechanism on real sensor nodes and compare the simulation results against the real implementation.

References


