Karlsruher Institut für Technologie

Karlsruhe Reports in Informatics 2010,12
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

Energy-Efficient Processing of Spatio-Temporal Queries in Wireless Sensor Networks

Extended version

Markus Bestehorn, Klemens Böhm,
Erik Buchmann, Stephan Kessler

2010

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

Fakultät für **Informatik**

Energy-Efficient Processing of Spatio-Temporal Queries in Wireless Sensor Networks

Extended Version

Markus Bestehorn Klemens Böhm Erik Buchmann Stephan Kessler

Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany

Abstract

Research on Moving Object Databases (MOD) has resulted in sophisticated query mechanisms for moving objects and regions. Wireless Sensor Networks (WSN) support a wide range of applications that track or monitor moving objects. However, applying the concepts of MOD to WSN is difficult: While MOD tend to require precise object positions, the information acquired in WSN may be incomplete or inaccurate. This may be because of limited detection ranges, node failures or detection mechanisms that only determine if an object is in the vicinity of a node, but not its exact position. In this paper, we study the processing of spatio-temporal queries in WSN. First, we adapt the models used in MOD to WSN while keeping their semantical depth. Second, we propose two approaches for processing such queries in WSN in-network instead of collecting all data at the base station. Based on a model that estimates communication costs of these strategies, nodes can determine the most energy-efficient strategy. Compared to collecting all data at the base station, our approaches reduce communication by up to 89%.

1 Introduction

Wireless Sensor Networks (WSN) have a broad range of applications. Many of them track moving objects and have spatio-temporal semantics. For example, environmental protection agencies track animals or hazardous materials in nature-protection areas [1, 25]. Authorities observe if unauthorized persons enter sensitive regions [21, 2].

Research on WSN has demonstrated that the declarativeness of queries (e.g., [37, 24]) is advantageous, but has focused on relational queries so far. However, [16, 35] have shown that expressing spatio-temporal semantics using relational query languages results in complex queries that are "hopelessly inefficient to process". This is because modeling the movement of objects and regions requires data types and operators not offered by purely relational database systems. To solve this problem, researchers on *Moving Object Databases (MOD)* have proposed languages to query moving objects and regions. To our knowledge, the question how to process such queries efficiently in WSN is untouched so far.

Well-known limitations of WSN render the problem difficult: MOD tend to assume precise and complete information on objects and regions queried. A region is modelled as a set of points that meet a user-defined criterion, e.g., all positions inside a forest. Applying this model to WSN would require precise knowledge which points fulfill such a criterion. This frequently is impractical, in particular if the area where the WSN has been deployed is large. Instead, for many WSN applications, one can say for each node if a certain user-defined criterion is met, e.g., if the node has been deployed inside or outside the forest. Thus, spatio-temporal queries in such settings typically target the topological relationship of objects detected and a set of nodes. The semantics of such spatio-temporal queries have not been defined yet. Additionally, object detection mechanisms in WSN have limited accuracy. Thus, even if precise information on the location of a region was available, nodes might be unable to determine if objects are inside, on the border or outside of a region. Finally, sensor nodes have limited energy resources. Communication in particular dominates energy consumption by orders of magnitude compared to other operations [24], e.g., computation. Thus, it is important to minimize communication when processing spatio-temporal queries in WSN.

In this paper, we show how to compute meaningful results for spatio-temporal queries in WSN. We first provide semantics for such queries related to object movement in relation to a set of nodes. These semantics also take the limited accuracy of object detection into account, while keeping the semantical depth of MOD. We then show how these semantics are integrated into the theoretical foundations for MOD, the 9-intersection model [9] in particular. This is important because it allows the re-use of existing concepts, e.g., spatio-temporal query languages [10].

To compute detection scenarios and query results, nodes must collect information on objects. A simple way to do so is sending all information about objects detected to the base station. This is prohibitive regarding energy consumption. To avoid this, we propose two strategies which compute re-

This is the extended version of a research paper submitted on June 24, 2010 to the 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2010).

sults in-network. They differ in the way they collect information from nodes close to each other. By combining the spatial correlation of object detections and spatio-temporal semantics, both strategies reduce the number of messages exchanged significantly. It depends on the query in question which strategy requires less communication. Our approach also addresses node failures. We provide mechanisms for the detection of such failures.

Since communication dominates energy consumption [24, 37], determining the strategy that is most energy efficient requires predicting the communication costs of each strategy. This prediction must meet two conflicting requirements: (1) It must be sufficiently simple to work with information that is available locally, i.e., on every node. (2) Its accuracy must be sufficient to determine the best strategy. – Given these difficulties, we propose a respective cost model.

Our study includes an evaluation using simulations as well as a Sun SPOT deployment. It shows that the predictions of the cost model are sufficiently accurate to choose the most energy-efficient strategy. Furthermore, it demonstrates that our in-network strategies reduce communication by 45-89%, compared to collecting all information at the base station.

Summing up, we focus on the following questions:

Q1 How can point-set based semantics of MOD be translated to node sets in WSN?

Q2 How can WSN derive predicate results efficiently using the semantics just mentioned?

We answer **Q1** by adapting the point-set model for WSN in Section 4. Section 5 answers **Q2** with execution strategies for spatio-temporal predicates in WSN.

2 Application Example

This section introduces our running example. Several biology projects track the movement of individuals at large spatial and temporal scale [19]. An example of a species studied in this way are caribous [26, 28]. The following query is an example of a spatio-temporal query scientists studying caribous could issue: "Which caribous have moved into the tree-covered swamp area on the south-western side of the river?" See Figure 1. The area that is covered by trees and on the south-western side of the river is a set of points. For most WSN deployments, recording the exact location of trees, swamp and river is impractical and unnecessarily complex. Instead, the majority of WSN use a *controlled deployment* [15] to reduce the time and effort spent for such recordings: Before the nodes [36, 31] start sensing, node positions and properties of their surroundings are recorded. Examples of such properties are if a node has been deployed inside the forest or in a treeless area, close to food resources, in the swamp or in a calving area. This information allows users to derive a set of nodes that are, say, in a tree-covered swamp area on the south-western side of the river (black-colored circles in Figure 1). It is sufficiently accurate for the purpose of such an installation if the WSN observes caribou movement in relation to this set of nodes. Our paper studies this case, i.e., users are interested in object movements in relation to a set of nodes. We refer to this set of nodes as *zone* to distinguish it from the term 'region' used in MOD, which is a set of points. Thus, users of WSN typically express their interest described above as follows: "Which caribous c have entered the zone Z?" See Figure 1. We define the exact meaning of such a query in Section 4.

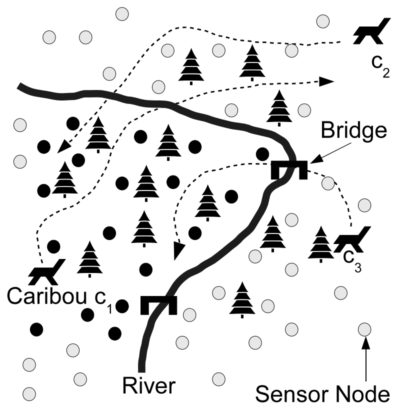

Figure 1. Illustration of the application scenario

This current work relies on several assumptions: Nodes are stationary, and they can distinguish between query-relevant objects and irrelevant ones. There exist several approaches to detect animals, e.g., acoustic recognition [22] or radio receivers which detect caribous wearing radio collars [26]. Typically these mechanisms only determine if an animal is in the vicinity of a node, but not its exact position. While our approach can be applied to more accurate mechanisms, we do not assume that object detection acquires more accurate information on the object location. Further, nodes are able to identify objects. For instance, if Sensor `Si` detects a certain object, and `Sj` detects the object later on, the WSN knows that it is the same object. Such an identification is usually available, e.g., through identification numbers on the radio collars or noise patterns that are characteristic. Finally, we limit this paper to queries regarding the relationship between moving objects and one zone.

3 Preliminaries

3.1 Foundations of MOD

We only review the foundations of MOD as far as relevant for this paper; see [16, 11] for further details. In MOD there exist three spatial entities: *objects*, *lines* and *regions*. An entity is a set of points of the d-dimensional Euclidean space $\mathbf{E^d}$ [14]. To ease presentation, we focus on $d = 2$, and leave aside lines in the following.

An entity divides the space into three pair-wise disjoint partitions: The *interior*, the *border* and the *exterior*. For a region $\mathbf{R}$, the border $\mathbf{R}^B$ contains all points of the line encompassing the interior $\mathbf{R}^I$. All points that are neither in $\mathbf{R}^B$ nor in $\mathbf{R}^I$ are part of the exterior $\mathbf{R}^O$. Typically, an object $\mathbf{x}$ is represented by its position $p \in \mathbf{E^2}$ and partitions the space as follows: The interior $\mathbf{x}^I$ contains only p, the border $\mathbf{x}^B$ is empty, and all points except p are the exterior $\mathbf{x}^O$. See [9] for formal definitions.

The 9-intersection model [9] describes the topological relationship of two entities A and B: As illustrated in Figure 2,

$$\begin{pmatrix} A^B \cap B^B \neq \varnothing & A^B \cap B^I \neq \varnothing & A^B \cap B^O \neq \varnothing \\ A^I \cap B^B \neq \varnothing & A^I \cap B^I \neq \varnothing & A^I \cap B^O \neq \varnothing \\ A^O \cap B^B \neq \varnothing & A^O \cap B^I \neq \varnothing & A^O \cap B^O \neq \varnothing \end{pmatrix}$$

Figure 2. 9-Intersection Model for two spatial entities A and B

$$\begin{pmatrix} F & F & F \\ F & T & F \\ T & T & T \end{pmatrix} \quad \begin{pmatrix} F & F & F \\ T & F & F \\ T & T & T \end{pmatrix} \quad \begin{pmatrix} F & F & F \\ F & F & T \\ T & T & T \end{pmatrix}$$

$$Inside\,(p_3,\mathbf{R}) \quad Meet\,(p_2,\mathbf{R}) \quad Disjoint\,(p_1,\mathbf{R})$$

Figure 3. 9-Intersection representation of spatial predicates ($A = p_i$ and $B = \mathbf{R}$)

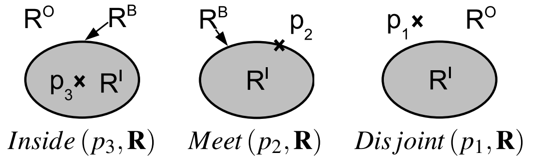

Figure 4. Spatial Predicates for Point/Region relations

there are nine possible intersections of the exterior, the border and the interior of A with the exterior, the border and the interior of B, respectively. Each of these intersections is either empty or not. Hence, a matrix of nine boolean values identifies the relationship of A and B.

There exist three predicates to describe the relationship of an object $\mathbf{x}$ and a region $\mathbf{R}$: $Inside\,(\mathbf{x},\mathbf{R})$, $Meet\,(\mathbf{x},\mathbf{R})$ and $Disjoint\,(\mathbf{x},\mathbf{R})$. Figure 4 illustrates each predicate.

Example 1: The left-most matrix in Figure 3 describes $Inside\,(p_3,\mathbf{R})$. Since the border ${p_3}^B$ is empty, it does not intersect with any partition of $\mathbf{R}$, as reflected by the first row of the matrix. ${p_3}^I$ contains one point, and the second row implies that $p_3^I \cap \mathbf{R}^I \neq \varnothing$, i.e., p_3 is inside $\mathbf{R}$. The last row shows that ${p_3}^O$ intersects with all partitions of $\mathbf{R}$. The matrices for $Meet\,(p_2,\mathbf{R})$ and $Disjoint\,(p_1,\mathbf{R})$ only differ from the matrix for $Inside\,(p_3,\mathbf{R})$ in the second row: The topological relation of p_2 and $\mathbf{R}$ conforms to $Meet\,(p_2,\mathbf{R})$ if $p_2^I \cap \mathbf{R}^B \neq \varnothing$. Similarly, $p_1^I \cap \mathbf{R}^O \neq \varnothing$ implies that p_1 is outside of $\mathbf{x}$, i.e., $Disjoint\,(p_1,\mathbf{x})$. □

Objects can move, and the topological relation of an object and a region can change over time. To deal with such changes, [11] defines the concatenation operator:

Definition 1 (Concatenation): The *concatenation of two predicates P and Q*, referred to as $P \triangleright Q$, is true if P is true for some time interval $[t_0;t_1[$, and Q is true at t_1. □

Concatenation allows users to formulate *sequences of predicates* $P_1 \triangleright P_2 \triangleright \ldots \triangleright P_p$. These sequences, also called *spatio-temporal developments*, express the interest in the spatio-temporal relationship of an object and a region over time.

Example 2: A user interested in all objects $\mathbf{x}$ that move into a region $\mathbf{R}$ can express this using the following spatio-temporal development:

$$Disjoint\,(\mathbf{x},\mathbf{R}) \triangleright Meet\,(\mathbf{x},\mathbf{R}) \triangleright Inside\,(\mathbf{x},\mathbf{R}) \qquad (1)$$

This predicate sequence is paraphrased as $Enter\,(\mathbf{x},\mathbf{R})$. Other sequences like $Touch\,(\mathbf{x},\mathbf{R})$ are possible as well:

$$Disjoint\,(\mathbf{x},\mathbf{R}) \triangleright Meet\,(\mathbf{x},\mathbf{R}) \triangleright Disjoint\,(\mathbf{x},\mathbf{R}) \qquad (2)$$

3.2 Related Work

[5, 7] have shown that accessing WSN declaratively is desirable for several reasons: Users formulate queries instead of writing code on sensor-node level. Such query engines can be re-used for different applications, which reduces development costs. Furthermore, these systems prolong the lifetime of battery-powered nodes by reducing communication [37, 24]. So far, relational queries have been the focus of research on query processing in WSN [6, 8, 29, 23]. Relational approaches are sufficient for queries like "What is the average temperature measured by all nodes?". But even simple spatio-temporal queries become very complex [35] and inefficient to process [16] in particular because they result in a large number of joins. Research on join processing [38, 20] in WSN has shown that, except for special cases, joins should be processed at the base station. Examples of such cases are join queries with high selectivity or queries where source nodes of tuples that join are close to each other. For example, [38] concludes that the distance between source nodes must be less than a few hops. This is because determining which nodes store tuples that meet the join condition possibly requires communication among many nodes. For spatio-temporal queries one join condition would be a selection that distinguishes between objects detected that are relevant and those that are not. First, the number of nodes detecting an arbitrary number of query-relevant objects over time can be arbitrarily high, i.e., selectivity is low. Second, the distance between these nodes may be large. Hence, existing join-processing approaches for WSN are not applicable, or they send all tuples to the base station to compute the join. Therefore, our reference point is collecting all information on object movements at the base station. Additionally, we show that taking spatio-temporal semantics into account reduces communication significantly, see Section 7. This occurs even if nodes detecting an object are close to each other.

A straightforward application of the existing results on MOD [11, 13, 16, 17] instead of using the existing relational query processors for WSN is not possible as well: First, the semantics of spatio-temporal queries in MOD are defined for regions, i.e., point sets, but not for zones, i.e., node sets. Second, sensor nodes typically cannot determine an object position precisely. [34, 33] have shown how to process spatio-temporal queries in MOD if only some points of object trajectories are known. While this helps if object positions are determined periodically, it still does require precise object positions from time to time. Our approach tries to derive meaningful results based on inaccurate object detections by sensor nodes.

In [4], we have shown how to acquire meaningful results for spatio-temporal queries in WSN referring to regions. As mentioned in Section 2, acquiring precise information on the location of a region is sometimes impractical. Thus, we leave aside regions in the following and focus on zones. Additionally, detection mechanisms used in WSN often cannot determine if an object is inside, on the border or outside of a region. [4] addresses the inaccuracy of detection mechanisms used in WSN by categorizing detected objects as follows: One category for all objects that conform to the movement in question for sure, one for objects that do not conform for sure and one for objects that might conform. As the complexity of queries increases, the number of objects in the latter category increases. To leave aside these complications, we now focus on spatio-temporal queries regarding sets of nodes. Finally, [4] does not address the processing of queries at all, in contrast to this paper.

4 Predicate Semantics in WSN

This section provides the theoretical foundation of our approach. Based on a generic WSN model and a set of notations, we show how to acquire meaningful results for spatio-temporal predicates in WSN.

4.1 Network Model and Notations

Notation (WSN): A *wireless sensor network* is a set $\mathsf{N} = \{\mathtt{S1}, \dots, \mathtt{Sn}\}$ of sensor nodes. p_i is the position of $\mathtt{Si}$.

Processing spatio-temporal predicates requires detection of objects moving in the area where the WSN has been deployed. To detect an object, it must be "in range", i.e., in the area observed by the detection function.

Definition 2 (Detection Area): The *detection area of node* $\mathtt{Si}$ is the set of points $\mathbf{D}_i \subseteq \mathbf{E^2}$ where $\mathtt{Si}$ can detect an object. □

Definition 3 (Detection Function): The *detection function* $detect(\mathtt{Si}, \mathbf{x}, t)$ is defined as follows:

$$detect(\mathtt{Si}, \mathbf{x}, t) = \begin{cases} T & \text{iff } \mathbf{x} \in \mathbf{D}_i \text{ at } t \\ F & \text{otherwise} \end{cases} \tag{3}$$

We say that *object* $\mathbf{x}$ *is detected at time* t if $detect(\mathtt{Si}, \mathbf{x}, t) = T$ for at least one $i \in \{1, \dots, n\}$. The detection area can have any shape or size, may overlap with detection areas of other nodes and may change over time. This is illustrated by Figure 5 where a rock prevents a sensor node from detecting objects behind it. Overlap of detection areas results in *simultaneous detection*.

Notation (Detection Set): The *detection set* $\mathsf{D}_\mathsf{t}^\mathbf{x} \subseteq \mathsf{N}$ is the set of nodes that detect $\mathbf{x}$ at some time t.

$$\mathsf{D}_\mathsf{t}^\mathbf{x} = \{\mathtt{Si} \in \mathsf{N} \mid detect(\mathtt{Si}, \mathbf{x}, t)\} \tag{4}$$

Sensor nodes typically cannot determine their detection area. However, the maximum detection range of the detection mechanism is typically available prior to deployment.

Definition 4 (Maximum Detection Range): The *maximum detection range* $\mathcal{D}_{max}$ is the maximum distance of an object to a node to be detected. □

Example 3 illustrates the difference between the detection area of a node and the max. detection range.

Example 3: For PIR-based motion detectors, $\mathcal{D}_{max} \approx 30$ meters. The sensor node in Figure 5 has been deployed close to a rock and cannot detect any object behind it. Thus, the area observed is much smaller than $\mathcal{D}_{max}$. Next, if there is an object in front of the lens of such a sensor, the area observed may be only a few centimeters. Nodes cannot detect this. □

Definition 5 (Communication Area): The *communication area of node* $\mathtt{Si}$ is the set of points $\mathbf{C}_i \subseteq \mathbf{E^2}$ where a node $\mathtt{Sj}$ with $i \neq j$ can receive messages sent by $\mathtt{Si}$. □

Communication areas may change over time and can have any shape or size. We say that *a node* $\mathtt{Si}$ *can directly communicate with another node* $\mathtt{Sj}$ if $p_j \in \mathbf{C}_i$.

Definition 6 (Communication Neighbors): The *communication neighbors* $\mathsf{CN_i}$ *of a node* $\mathtt{Si}$ are the nodes $\mathtt{Si}$ can directly communicate with. □

Wireless communication is not bi-directional, i.e., $\mathtt{Sj} \in \mathsf{CN_i}$ does not imply that $\mathtt{Si} \in \mathsf{CN_j}$. – There exist several protocols [27, 12] that let nodes exchange information via multiple hops if they cannot communicate directly. These protocols typically require a *routing table* on each node that contains all communication neighbors.

4.2 Spatio-Temporal Predicates for Zones

In line with our application example in Section 2, users are interested in the spatio-temporal relationship between objects and a *zone*, i.e., a set of nodes.

Definition 7 (Zone): A *zone* Z is a non-empty set of nodes. We say *a node* $\mathtt{Si}$ *is inside of* Z if $\mathtt{Si} \in \mathsf{Z}$. $\mathtt{Si}$ *is outside of* Z otherwise. □

To process spatio-temporal predicates in WSN, one must consider which nodes inside and outside of the zone detect an object. I.e., we are interested in the intersections of $\mathsf{D}_\mathsf{t}^\mathbf{x}$ with Z and $\overline{\mathsf{Z}}$. We refer to the different cases as *detection scenarios*, and there are four of them:

$\mathtt{DS^O}$**:** Only nodes outside of Z currently detect $\mathbf{x}$:

$$\mathsf{D}_\mathsf{t}^\mathbf{x} \cap \mathsf{Z} = \varnothing \wedge \mathsf{D}_\mathsf{t}^\mathbf{x} \cap \overline{\mathsf{Z}} \neq \varnothing \tag{5}$$

$\mathtt{DS^I}$**:** Only nodes inside of Z detect $\mathbf{x}$:

$$\mathsf{D}_\mathsf{t}^\mathbf{x} \cap \mathsf{Z} \neq \varnothing \wedge \mathsf{D}_\mathsf{t}^\mathbf{x} \cap \overline{\mathsf{Z}} = \varnothing \tag{6}$$

$\mathtt{DS^B}$**:** $\mathsf{D}_\mathsf{t}^\mathbf{x}$ contains nodes from Z as well as $\overline{\mathsf{Z}}$:

$$\mathsf{D}_\mathsf{t}^\mathbf{x} \cap \mathsf{Z} \neq \varnothing \wedge \mathsf{D}_\mathsf{t}^\mathbf{x} \cap \overline{\mathsf{Z}} \neq \varnothing \tag{7}$$

$\mathtt{DS^N}$**:** $\mathsf{D}_\mathsf{t}^\mathbf{x}$ neither intersects with Z nor with $\overline{\mathsf{Z}}$, i.e., $\mathbf{x}$ is currently undetected:

$$\mathsf{D}_\mathsf{t}^\mathbf{x} \cap \mathsf{Z} = \varnothing \wedge \mathsf{D}_\mathsf{t}^\mathbf{x} \cap \overline{\mathsf{Z}} = \varnothing \tag{8}$$

LEMMA 1. *For any point of time, exactly one detection scenario holds.*

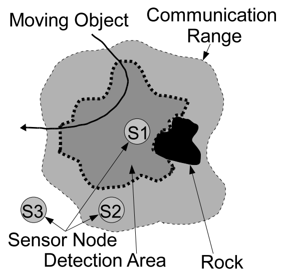

Figure 5. Illustration of the node model

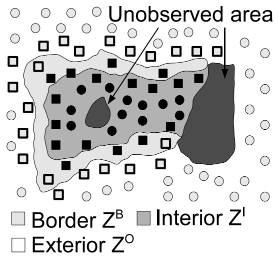

Figure 6. Point set model for zones

Detection Scenario	Predicate	Partition
$\mathtt{DS}^{\mathtt{N}}$	-	Z^N
$\mathtt{DS}^{\mathtt{O}}$	$Disjoint\left(\mathbf{x},\mathsf{Z}\right)$	Z^O
$\mathtt{DS}^{\mathtt{I}}$	$Inside\left(\mathbf{x},\mathsf{Z}\right)$	Z^I
$\mathtt{DS}^{\mathtt{B}}$	$Meet\left(\mathbf{x},\mathsf{Z}\right)$	Z^B

Table 1. Mapping detection of an object x to predicates/space partitions

Proof. The set of detection scenarios is exhaustive because it covers all points in $\mathbf{E^2}$. A point $p \in \mathbf{E^2}$ is either included in at least one detection area or unobserved. $\mathtt{DS}^{\mathtt{N}}$ covers all points $\mathbf{E^2} \setminus \bigcup_{1\le i\le n} \mathbf{D}_i$. The observed points $\bigcup_{1\le i\le n} \mathbf{D}_i$ are covered by one of the remaining scenarios: All points exclusively observed by nodes outside of $\mathbf{R}$ are covered by $\mathtt{DS}^{\mathtt{O}}$. Similarly $\mathtt{DS}^{\mathtt{I}}$ covers all points solely observed by nodes inside $\mathbf{R}$. All points observed by nodes inside and outside of $\mathbf{R}$ are covered by $\mathtt{DS}^{\mathtt{B}}$. Each of these point sets is pair-wise disjoint with the others, i.e., for all $t \in time$ exactly one detection scenario holds. □

Detection scenarios formalize the information acquired by object detection in WSN and abstract from the details of object detection, i.e., they are applicable to any detection hardware or mechanism. Based on detection scenarios, we define the semantics of spatio-temporal predicates as follows: When an object $\mathbf{x}$ is undetected, the WSN cannot make any statement on the topological relation of $\mathbf{x}$ and Z. Thus, when $\mathtt{DS}^{\mathtt{N}}$ occurs, no predicate is true. $\mathtt{DS}^{\mathtt{O}}$ occurs if $\mathbf{x}$ is exclusively detected by nodes outside of the zone. Likewise, $\mathtt{DS}^{\mathtt{I}}$ occurs if $\mathbf{x}$ is exclusively detected by nodes in the zone.

Definition 8 ($Inside\left(\mathbf{x},\mathsf{Z}\right)$): Object $\mathbf{x}$ conforms to the predicate $Inside\left(\mathbf{x},\mathsf{Z}\right)$ if $\mathtt{DS}^{\mathtt{I}}$ occurs. □

Definition 9 ($Disjoint\left(\mathbf{x},\mathsf{Z}\right)$): Object $\mathbf{x}$ conforms to the predicate $Disjoint\left(\mathbf{x},\mathsf{Z}\right)$ if $\mathtt{DS}^{\mathtt{O}}$ occurs. □

If nodes from Z and $\overline{\mathsf{Z}}$ detect an object, i.e., $\mathtt{DS}^{\mathtt{B}}$ occurs, the WSN can derive that the object is between being inside and outside of the zone. Hence, we define:

Definition 10 ($Meet\left(\mathbf{x},\mathsf{Z}\right)$): The object $\mathbf{x}$ conforms to $Meet\left(\mathbf{x},\mathsf{Z}\right)$ if $\mathtt{DS}^{\mathtt{B}}$ occurs. □

Table 1 serves as a summary. $\triangleright$ and other concepts from MOD, e.g., lifting [16], are applicable to these predicates as well. Thus, one can construct developments that query the spatio-temporal relationship of objects and a zone. For instance, one could define:

$$Enter\left(\mathbf{x},\mathsf{Z}\right) = Disjoint\left(\mathbf{x},\mathsf{Z}\right) \triangleright Meet\left(\mathbf{x},\mathsf{Z}\right) \triangleright Inside\left(\mathbf{x},\mathsf{Z}\right) \tag{9}$$

4.3 Point-Set Topology for Zones

According to point-set topology, a region partitions the (Euclidean) space into three subsets of points. By definition, this partitioning is *regular* [32]: This means that a region $\mathbf{R}$ does not contain holes, is continuous, and the border $\mathbf{R}^B$ encompasses the interior $\mathbf{R}^I$ completely. The space partitioning of zones in turn is not regular, as we will explain. We show how this affects the semantics of spatio-temporal queries in WSN and how to deal with this non-regularity.

First we derive a space partitioning based on the predicate definitions and detection scenarios above using the following idea: Without loss of generality, let $\mathbf{x}$ be detected according to $\mathtt{DS}^{\mathtt{O}}$, i.e., $Disjoint\left(\mathbf{x},\mathsf{Z}\right)$. From this, we infer that the position of $\mathbf{x}$ is some $p \in \mathbf{E^2}$ exclusively observed by nodes outside of Z, i.e., *the exterior of the zone*. For each predicate we can derive such a subset of the space.

Definition 11 (Unobserved Area): All points not contained in a detection area form *the unobserved area* Z^N:

$$\mathsf{Z}^N = \left\{ p \in \mathbf{E^2} \mid \nexists \mathtt{Si} \in \mathsf{N} : p \in \mathbf{D}_i \right\} \tag{10}$$

Definition 12 (Exterior): All points exclusively observed by nodes in $\overline{\mathsf{Z}}$ constitute *the exterior of a zone* Z^O:

$$\mathsf{Z}^O = \left\{ p \in \mathbf{E^2} \mid p \notin \mathsf{Z}^N \wedge \nexists \mathtt{Si} \in \mathsf{Z} : p \in \mathbf{D}_i \right\} \tag{11}$$

Definition 13 (Interior): All points of space exclusively observed by nodes in Z constitute *the interior of a zone* Z^I:

$$\mathsf{Z}^I = \left\{ p \in \mathbf{E^2} \mid p \notin \mathsf{Z}^N \wedge \nexists \mathtt{Si} \in \overline{\mathsf{Z}} : p \in \mathbf{D}_i \right\} \tag{12}$$

Definition 14 (Border): All points of space observed by nodes in Z and $\overline{\mathsf{Z}}$ form the *border of a zone* Z^B:

$$\mathsf{Z}^B = \left\{ p \in \mathbf{E^2} \mid \exists \mathtt{Si} \in \mathsf{Z}, \exists \mathtt{Sj} \in \overline{\mathsf{Z}} : p \in \mathbf{D}_i, p \in \mathbf{D}_j \right\} \tag{13}$$

LEMMA 2. *The point sets Z^N, Z^O, Z^I and Z^B partition the space.*

Proof. Follows directly from Lemma 1. □

Figure 6 illustrates the space partitioning for zones. Black-colored circles or squares[1] represent nodes in the zone. Ex-

[1]The difference between squares and circles is irrelevant here and will be explained in Section 5.3.

ternal factors influence detection areas which in turn determine the partitions. Unobserved areas may exist permanently or temporarily. This results in holes in the interior Z^I of a zone or borders not encompassing Z^I completely. For example, node failures might cause an unobserved area as illustrated on the right-hand side of the zone in Figure 6. Hence, space partitioning based on zones is not regular.

Non-regularity has an impact on spatio-temporal developments, as we illustrate with $Enter(\mathbf{x}, \mathbf{R})$/$Enter(\mathbf{x}, \mathsf{Z})$: MOD can assume that point sets are regular, i.e., an object $\mathbf{x}$ has to cross the border $\mathbf{R}^B$. This is not true for WSN: $\mathbf{x}$ may be detected according to $\mathtt{DS}^0$ first, move through an unobserved area and then appear inside of Z. Users interested in objects moving in this way can not express this as follows:

$$\mathbb{P}(\mathbf{x}, \mathsf{Z}) = Disjoint(\mathbf{x}, \mathsf{Z}) \triangleright Inside(\mathbf{x}, \mathsf{Z}) \qquad (14)$$

This sequence in (14) never occurs, because $\triangleright$ requires *Inside* $(\mathbf{x}, \mathsf{Z})$ to follow $Disjoint(\mathbf{x}, \mathsf{Z})$ immediately. On the other hand, $Enter(\mathbf{x}, \mathsf{Z})$ excludes all objects that are unobserved while moving into the zone because $Meet(\mathbf{x}, \mathsf{Z})$ never occurs. In other words, users cannot express such a query given the three predicates and $\triangleright$.

Definition 15 (Relaxed Concatenation): The *relaxed concatenation of two predicates*, $P \tilde{\triangleright} Q$, is true if P is true for some interval $[t_0, t_1[$, and Q is true at $t_2 > t_1$. □

(15) expresses the query discussed above, and Example 4 shows that this new primitive increases the semantical depth:

$$WSNEnter(\mathbf{x}, \mathsf{Z}) = Disjoint(\mathbf{x}, \mathsf{Z}) \tilde{\triangleright} Inside(\mathbf{x}, \mathsf{Z}) \qquad (15)$$

Example 4: The area where the WSN in Figure 1 is deployed contains a river with several bridges. Suppose that nodes are deployed so that caribous moving over a bridge are detected, but caribous swimming are not, i.e., the river itself is unobserved. A user only interested in caribous c entering Z by crossing bridges can use $Enter(c, \mathsf{Z})$. A user interested in all caribous can express this as $WSNEnter(\mathbf{x}, \mathsf{Z})$. □

Next, we study some important properties of the $\tilde{\triangleright}$ operator like associativity and how it can be combined with the $\triangleright$ operator.

LEMMA 3. $P_1 \triangleright P_2 \Rightarrow P_1 \tilde{\triangleright} P_2$

Proof. According to Definition 15, the right-hand side is true if P_1 is true for some interval $[t_0, t_1[$ and P_2 is true at $t_2 \geq t_1$. The left-hand side of the implication states that P_1 is true for some interval $[t_0, t_1[$ and P_2 is true at $t_2 = t_1$. Hence, if the left-hand side is true, the right-hand side is also true. □

LEMMA 4. $P_1 \tilde{\triangleright} (P_2 \tilde{\triangleright} P_3) = (P_1 \tilde{\triangleright} P_2) \tilde{\triangleright} P_3$.

Proof. The left hand side means $\exists [t_0, t_1[: P_1$ and $\exists t_2 \geq t_1 : (P_2 \tilde{\triangleright} P_3)$. Further, $\exists [t_2, t_3[: P_2$ and $\exists t_4 \geq t_3 : P_3$. The right hand side expresses that $\exists [t'_0, t'_3[: (P_1 \tilde{\triangleright} P_2)$ and $\exists t'_4 \geq t'_3 : P_3$. Additionally, $\exists [t'_0, t'_1[, t'_1 \leq t'_3 : P_1$ and $\exists t'_2 \geq t'_1 \wedge t'_2 \leq t'_3 : P_2$. If the left hand side is true for $t'_0 = t_0, t'_1 = t_1, t'_2 = t_2, t'_3 = t_3$ the right hand side is fulfilled also (and vice versa). □

LEMMA 5. $P_1 \triangleright (P_2 \tilde{\triangleright} P_3) = (P_1 \triangleright P_2) \tilde{\triangleright} P_3$

Proof. $P_1 \triangleright (P_2 \tilde{\triangleright} P_3)$ implies $P_1 \tilde{\triangleright} (P_2 \tilde{\triangleright} P_3)$ based on Lemma 3. Similarly, $(P_1 \triangleright P_2) \tilde{\triangleright} P_3$ implies $(P_1 \tilde{\triangleright} P_2) \tilde{\triangleright} P_3$. Hence, we get $P_1 \tilde{\triangleright} (P_2 \tilde{\triangleright} P_3) = (P_1 \tilde{\triangleright} P_2) \tilde{\triangleright} P_3$ which is true according to Lemma 4. □

Users can express spatio-temporal queries using both concatenation operators using spatio-temporal developments.

Definition 16 (Spatio-Temporal Development): A *spatio-temporal development* $\mathbb{P}(\mathbf{x}, \mathsf{Z})$ is a sequence of predicates $P_1(\mathbf{x}, \mathsf{Z})\,\theta \ldots \theta P_q(\mathbf{x}, \mathsf{Z})$ with $\theta \in \{\triangleright, \tilde{\triangleright}\}$, and $P_i(\mathbf{x}, \mathsf{Z}) \in \{$*Inside* $(\mathbf{x}, \mathsf{Z})$, $Meet(\mathbf{x}, \mathsf{Z})$, $Disjoint(\mathbf{x}, \mathsf{Z})\}$.
The movement of an object $\mathbf{x}$ conforms to $\mathbb{P}(\mathbf{x}, \mathsf{Z})$ if each pair $P_{i-1}(\mathbf{x}, \mathsf{Z})\ \theta\ P_i(\mathbf{x}, \mathsf{Z})$ with $2 \leq i \leq q$ is true. □

This concludes our study regarding **Q1**. We have defined the semantics of spatio-temporal predicates for SN that express the topological relation of an object and a zone. Furthermore, we have shown how to take unobserved areas into account when expressing spatio-temporal queries for SN and provided a space partitioning for zones.

5 Computing Detection Scenarios

In this section, we provide an answer to **Q2**, i.e., process spatio-temporal queries efficiently: We propose two strategies that reduce the number of messages exchanged between nodes, compared to a straightforward approach that collects all information at the base station. Recall that this approach resembles relational query processors used for spatio-temporal queries, as discussed in Section 3.2. Appendix B shows that reducing communication can increase the battery-lifetime significantly for the used Sun SPOT sensor nodes. Other nodes show similar behaviour, e.g., [24] provides similar results for MicaZ [36]. Reducing the number of messages is crucial because communication dominates energy consumption by orders of magnitude [24]. We present our approach in the following steps; the numbers are in line with the ones of the respective sections:

5.1 Data Structures and Algorithms: We describe a relational schema to store information on objects detected and show how to derive detection scenarios from this. The remainder of the section deals with acquiring the necessary information.

5.2 Centralized Data Collection: A straightforward strategy to acquire information necessary to compute detection scenarios collects all data at the base station. This is our base-line strategy.

5.3 Distributed Data Collection: We propose two strategies that exploit spatial correlation of object detections to derive detection scenarios. Our evaluation shows that they reduce communication significantly.

5.4 Failures: Failures of nodes may impact query results. For every strategy, we investigate when a failure affects the query result and how to detect such a failure.

We assume that the following steps have been completed before the WSN starts to process a query:

1. Definition of a zone Z.
2. Specification of the movement of interest as a development $\mathbb{P}(\mathbf{x}, \mathsf{Z})$.
3. Dissemination of Z and $\mathbb{P}(\mathbf{x}, \mathsf{Z})$ to all nodes.

The query result returned to the user by the base station includes every object whose movement conforms to $\mathbb{P}(\mathbf{x}, \mathsf{Z})$. To accomplish this, the WSN must compute the detection scenario when an object is detected. Using Table 1, any node can determine which predicate the detected object conforms after the detection scenario has been computed.

An object $\mathbf{x}$ fulfills $\mathbb{P}(\mathbf{x}, \mathsf{Z})$ if the WSN derives from the detection scenarios of $\mathbf{x}$ that the predicates have occurred in the correct order. The distributed strategies notify the base station whenever a predicate $P(\mathbf{x}, \mathsf{Z})$ in $\mathbb{P}(\mathbf{x}, \mathsf{Z})$ is satisfied. Thus, the base station determines if $\mathbf{x}$ has fulfilled $\mathbb{P}(\mathbf{x}, \mathsf{Z})$. Note that a node `Si` may send several notifications regarding a predicate to the base station because it detects the same object more than once. This is intended, for two reasons: First, the query may be a predicate sequence that contains a predicate twice, e.g., *Touch* $(\mathbf{x}, \mathsf{Z})$ (cf. (2)). Second, coordinating nodes such that they only send notifications regarding predicates that have not occurred on any other node requires communication. A preliminary study of ours has shown that the communication effort for such coordination only pays off if the network is very small, the zone is small, and if the object moves through detection areas of many nodes repeatedly. Thus, we do not intend to prevent this. On the other hand, we show in Section 5.3 how to exploit spatio-temporal semantics to reduce the number of notifications, e.g., only few nodes send notifications for queries like *Enter* $(\mathbf{x}, \mathsf{Z})$.

5.1 Data Structures and Algorithms

A relation `Detections` contains data on objects detected. It depends on the strategy where it is stored: All tuples are sent to the base station (centralized), or they are distributed and/or replicated among the nodes in the WSN (distributed). `Detections` has the following attributes:

- `NodeID` Identifier of the node `Si` detecting the object identified by attribute `ObjectID`
- `ObjectID` Identifier of the object detected by `Si`.
- t_{entry}: $t \in time$ when `Si` starts to detect the object.
- t_{exit}: This value is either $\top$ or a $t > t_{entry}$. If it is $\top$, `Si` is still detecting the object. Otherwise, `Si` has detected the object during the time interval $[t_{entry}; t_{exit}[$.

We say that *a tuple* T *originates from node* `Si` if `T.NodeID` = `Si`.

The memory consumption for each tuple in `Detections` is computed as follows: Sun SPOT sensor nodes are uniquely identified by IEEE addresses which require 16 bytes of memory. Thus, the attribute `NodeID` requires 16 bytes. The timestamps t_{entry} and t_{exit} are `long` values which each require 16 bytes as well according to the specification of the Squawk VM running on Sun SPOT sensor nodes. For the identifiers of detected objects, i.e., `ObjectID`, our implementation uses `int` values, which require 4 bytes. Summing up, a tuple of in `Detections` requires $16 + 4 + 16 + 16 = 52$ bytes. Since Sun SPOTs have 512 kilobytes of main memory, a node can store thousands of tuples. Therefore, we assume in the following that nodes have sufficient memory to store tuples required to compute a detection scenario and leave aside memory optimization.

We refer to the moment an object $\mathbf{x}$ moves into the detection area of a node `Si` as *entry event*. When such an entry occurs at time t, a tuple $[\texttt{Si}, \mathbf{x}, t, \top]$ is added to `Detections`. Then the object may be inside the detection area for an arbitrary interval of time. We refer to the moment when $\mathbf{x}$ leaves the detection area as *exit event*. When $\mathbf{x}$ this happens at t', the existing tuple is updated, i.e., becomes $[\texttt{Si}, \mathbf{x}, t, t']$. Algorithm 5.1 illustrates the insertion of a new tuple and Algorithm 5.2 illustrates the corresponding update when an exit occurs.

Algorithm 5.1: Insertion of tuples into `Detections` when an object $\mathbf{x}$ enters the detection area $\mathbf{D}_i$ of `Si`

```
   Input: Object identifier $\mathbf{x}$, current time $t \in time$
1  INSERT INTO
2  |  Detections (NodeID, ObjectID, $t_{entry}$, $t_{exit}$)
3  VALUES (Si, $\mathbf{x}$, $t$, $\top$);
```

Algorithm 5.2: Updating tuples in `Detections` when an object $\mathbf{x}$ leaves the detection area $\mathbf{D}_i$ of `Si`

```
   Input: Object identifier $\mathbf{x}$, current time $t' \in time$
1  UPDATE Detections
2  SET $t_{exit} = t'$
3  WHERE $t_{exit} = \top$ AND ObjectID = $\mathbf{x}$
```

For non-continuous detection mechanisms nodes can derive the interval $[t_{entry}; t_{exit}[$ by temporal interpolation: Suppose `Si` checks periodically at $t_0, t_1, \ldots$ for objects. An entry event occurs at t_j if `Si` did not detect an object at t_{j-1} but detects it at t_j. An exit event occurs at t_j if `Si` detected an object at t_{j-1} and does not detect it at t_j. The frequency at which nodes must check for objects obviously depends on the properties of the used hardware and the object it is intended for. An example of such a property is the expected maximum speed of the objects observed.

To determine the detection scenario, the system must compute how the detection set $\mathsf{D}^{\mathbf{x}}_{\mathsf{t}}$ intersects with Z and $\overline{\mathsf{Z}}$ at t based on `Detections`. We refer to this computation as $\texttt{isDetecting}(\mathsf{S}^*, \mathbf{x}, \texttt{t})$, which is defined as follows:

$$\texttt{isDetecting}(\mathsf{S}^*, \mathbf{x}, \texttt{t}) = \begin{cases} T & \text{if } \exists \texttt{Si} \in \mathsf{S}^* : detect(\texttt{Si}, \mathbf{x}, t) \\ F & otherwise \end{cases} \tag{16}$$

The input parameter S^* is either Z or $\overline{\mathsf{Z}}$. Algorithm 5.3 implements this function and we define for each strategy where the algorithm is executed. $\texttt{isDetecting}(\mathsf{S}^*, \mathbf{x}, \texttt{t})$ consists of two nested loops: The outer loop runs through the nodes contained in S^*, and the inner loop determines for each of these nodes if it detects $\mathbf{x}$ at time t. To compute a detection scenario, we use this algorithm twice, to determine $\texttt{isDetecting}(\mathsf{Z}, \mathbf{x}, \texttt{t})$ as well as $\texttt{isDetecting}(\overline{\mathsf{Z}}, \mathbf{x}, \texttt{t})$. Based on the results, one can determine the detection scenario according to Table 2: Each cell corresponds to a pair $[\texttt{isDetecting}(\mathsf{Z}, \mathbf{x}, \texttt{t}), \texttt{isDetecting}(\overline{\mathsf{Z}}, \mathbf{x}, \texttt{t})]$ and contains the corresponding detection scenario. In the following, we deal with the collection of tuples in `Detections` to ensure a correct computation of detection scenarios.

Definition 17 (Correctness): The *computation of the detection scenario is correct* if the space partition corresponding to the detection scenario computed (cf. Table 1) contains the position $p \in \mathbf{E}^2$ of the object detected. □

Algorithm 5.3: `isDetecting`$(\mathsf{S}^*, \mathbf{x}, t)$ computes if a node from a set S^* detected object **x** at time t.

```
    Input: Relation Detections, an object identifier x, a
           set of nodes $S^*$ and a value $t \in time$
    Output: $T$ if a node in $S^*$ detected x at $t$, otherwise $F$.
1   for each node ID id in $S^*$ do
2       for each entry E in Detections do
3           if $E.ObjectID = \mathbf{x}$ AND $E.NodeID = id$
            AND $E.t_{entry} \le t \le E.t_{exit}$ then
4               return $T$; // Node in $S^*$ detects $x$ at
                $t$
5           end
6       end
7   end
8   return $F$
```

		`isDetecting`$(\overline{\mathsf{Z}}, \mathbf{x}, t)$	
		T	F
`isDetecting`$(\mathsf{Z}, \mathbf{x}, t)$	T	DS^{B}	DS^{I}
	F	DS^{O}	DS^{N}

Table 2. Deriving detection scenarios from `Detections`

Definition 18 (Completeness): `Detections` *is complete regarding an object* **x** *and a time* t if it contains all tuples $\{\mathtt{Si}, \mathbf{x}, t_1, t_2\}$ with $t_1 \le t$ and $t \le t_2$ or $t_2 = \top$. □

LEMMA 6. *If the relation* `Detections` *is complete, the detection scenario computed according to Table 2 is correct.*

Proof. Without loss of generality, assume the computed detection scenario is DS^{O} which means that **x** is in Z^O. Thus, at least one $\mathtt{Si} \in \overline{\mathsf{Z}}$ detects **x** at time t. The computed detection scenario would be incorrect, if there existed a node $\mathtt{Sj} \in \overline{\mathsf{Z}}$ that detects **x** at t as well. Such a node cannot exist since `Detections` is complete. For the other detection scenarios, the proof is similar. □

Summing up, the base station or an arbitrary node must store a complete set of tuples locally to compute a detection scenario for an object **x** and a time t. In the following we deal with acquiring these tuples.

5.2 Centralized Data Collection

A straightforward approach is that every node notifies the base station whenever an object enters or leaves a detection area. The base station then modifies `Detections` as shown and computes a detection scenario. See Algorithm 5.4.

Arbitrary nodes detecting an object execute the first part of Algorithm 5.4. Sending tuples from `Si` to the base station requires routing protocols [27, 12]. These protocols forward messages via multiple hops if `Si` is not a communication neighbor of the base station.

The base station executes the second part. Line 5 modifies `Detections` as described previously. The base station then has to wait a timeout t_{delay} before it computes the detection scenario according to Table 2. The timeout ensures that

Algorithm 5.4: Centralized Data Collection

```
1   When x enters/leaves $\mathbf{D}_i$ of Si at t do
2       Si sends corresponding notification to base station
3   end
4   When base station receives notification from Si do
5       Modify Detections at base station
6       Wait $t_{delay}$
7       Compute
        [isDetecting($S^{R}$, x, t), isDetecting($S^{\overline{R}}$, x, t)]
8   end
```

notifications of nodes which simultaneously detect an object have arrived before the detection scenario is computed. t_{delay} is the maximum time a notification may need to be forwarded to the base station. Its actual value depends on factors such as communication hardware, WSN size, routing protocol etc. For our reference implementation we use a delay of 30 seconds.

LEMMA 7. *If t_{delay} is the maximum time a notification needs to travel from a node* `Si` *to the base station,* `Detections` *stored at the base station is complete at* $t + t_{delay}$.

Proof. We produce the proof of the contrary. Let `Si` be a node that detected **x** at t and send a notification to the base station. If this notification has not arrived at the base station by $t + t_{delay}$, t_{delay} was not the maximum time a notification may need to reach the base station from an arbitrary node. □

Thus, the computation of the detection scenario based on centralized data collection is correct.

5.3 Distributed Data Collection

In the following, we propose two strategies which distribute the relation `Detections`. The distribution is done in such a way that any node `Si` detecting an object **x** can compute the detection scenario. As we show, this reduces communication for two reasons:

- Nodes only notify the base station on objects that possibly fulfill the query.
- The nodes data must be collected from are fewer, i.e., only some nodes communicate.

The latter point stems from the following idea: When a node `Si` detects an object **x**, only nodes in its vicinity can detect the object simultaneously. This is because **x** at position $p \in \mathbf{E}^2$ can be detected only by nodes whose detection area contains p. The problem is that detection mechanisms in WSN typically do not allow precise localization of the object detected, i.e., p is unknown. But in turn, `Si` can derive that only nodes whose detection area overlaps with $\mathbf{D}_i$ could possibly detect **x** simultaneously, i.e., contain p.

Definition 19 (Detection Neighbor): Node `Sj` is a *detection neighbor of* `Si` if the detection areas of both nodes overlap, i.e., $\mathbf{D}_i \cap \mathbf{D}_j \neq \varnothing$. DN_i is the set of detection neighbors of `Si`. □

Section 5.3.1 shows how to approximate the detection neighbors if detection areas are indeterminable. Since Z is disseminated to all nodes, every Si can derive for each detection neighbor $\texttt{Sj} \in \mathsf{DN_i}$ if it is in Z or not.

Notation (Detection-Neighbor Subsets): We refer to the subset of detection neighbors of Si in the zone Z as $\mathsf{DN_i}^{\mathsf{Z}}$. $\mathsf{DN_i}^{\overline{\mathsf{Z}}}$ contains all detection neighbors of node Si that are outside of Z.

LEMMA 8. Detections *stored at* Si *is complete regarding* **x** *and t if* Si *detects* **x** *at t and obtains all tuples on* **x** *originating from its detection neighbors* $\mathsf{DN_i}$.

Proof. We prove this by showing that there cannot exist a node $\texttt{Sj} \notin \mathsf{DN_i}$ that detects **x** at t. $\texttt{Sj} \notin \mathsf{DN_i}$ implies that the detection areas of Si and Sj do not overlap, i.e., $\mathbf{D}_i \cap \mathbf{D}_j = \varnothing$. Thus, there does not exist a $p \in \mathbf{E^2}$ where Si and Sj can detect **x** simultaneously. Hence, Sj cannot detect **x** at t. □

Lemma 8 limits the nodes from which Si must be acquire tuples to the detection neighbors $\mathsf{DN_i}$. By taking into account that Si is either in Z or $\overline{\mathsf{Z}}$ we actually can compute a correct detection scenario without Detections being complete.

Definition 20 (Semi-Completeness): Detections *regarding* **x** *and t stored at a node* $\texttt{Si} \in \mathsf{Z}$ *is semi-complete* if it contains all tuples $[\texttt{Sj}, \mathbf{x}, t_1, t_2]$ with $t_1 \le t \le t_2$ where $\texttt{Sj} \in \mathsf{DN_i}^{\overline{\mathsf{Z}}}$.

Detections *regarding* **x** *and t stored at a node* $\texttt{Si} \in \overline{\mathsf{Z}}$ *is semi-complete* if it contains all tuples $[\texttt{Sj}, \mathbf{x}, t_1, t_2]$ with $t_1 \le t \le t_2$ where $\texttt{Sj} \in \mathsf{DN_i}^{\mathsf{Z}}$. □

LEMMA 9. *Let* Si *detect* **x** *at t. Without loss of generality, let* $\texttt{Si} \in \mathsf{Z}$. *If* Detections *stored at* Si *is semi-complete regarding* **x** *and t, the computation of the detection scenario at* Si *according to Table 1 is correct.*

Proof. Since Si detects **x**, $\texttt{isDetecting}_{\mathsf{Z}}(\mathbf{x},\texttt{t}) = T$. Thus, only $\texttt{isDetecting}_{\overline{\mathsf{Z}}}(\mathbf{x},\texttt{t})$ remains to be computed by Si. This only requires tuples from $\overline{\mathsf{Z}}$. □

Lemma 9 implies that the detection scenario computation is still correct if Detections contains only tuples from a subset of certain detection neighbors. This reduces the communication, in particular when this subset is empty.

Definition 21 (Border Node): Si is a *border node* if

- $\texttt{Si} \in \mathsf{Z}$ and $\mathsf{DN_i}^{\overline{\mathsf{Z}}} \neq \varnothing$, or
- $\texttt{Si} \in \overline{\mathsf{Z}}$ and $\mathsf{DN_i}^{\mathsf{Z}} \neq \varnothing$. □

Figure 6 illustrates a WSN, a zone Z and the resulting space partitioning based on detection areas (cf. Section 4.3). Nodes are represented as squares or circles, and there are four kinds of nodes: Non-border nodes inside Z are represented by black-colored circles. Black-colored squares correspond to border nodes inside Z. Similarly, grey-colored squares and circles correspond to border and non-border nodes outside of Z, respectively. A significant share of the nodes in this scenario are non-border nodes. According to Lemma 10, non-border nodes can compute detection scenarios without obtaining tuples originating from any detection neighbor. This reduces communication and thus conserves energy.

LEMMA 10. *If a non-border node* Si *detects* **x** *at t and modifies* Detections *accordingly,* Detections *stored at* Si *is semi-complete.*

Proof. Without loss of generality let $\texttt{Si} \in \mathsf{Z}$ and $\mathsf{DN_i}^{\overline{\mathsf{Z}}} = \varnothing$, i.e., Si is not a border node. $\mathsf{DN_i}^{\overline{\mathsf{Z}}} = \varnothing$ implies that there does not exist a node $\texttt{Sj} \in \overline{\mathsf{Z}}$ whose detection area overlaps with the detection area of Si. Thus, simultaneous detection of an object by Si and some $\texttt{Sj} \in \overline{\mathsf{Z}}$ is not possible by definition. Hence, detection of an object **x** by Si implies $\texttt{isDetecting}_{\overline{\mathsf{Z}}}(\mathbf{x},\texttt{t}) = F$ and $\texttt{isDetecting}_{\mathsf{Z}}(\mathbf{x},\texttt{t}) = T$. □

LEMMA 11. *Let* $\mathbb{P}(\mathbf{x},\mathsf{Z}) = P_1(\mathbf{x},\mathsf{Z}) \triangleright P_2(\mathbf{x},\mathsf{Z})$. *Tuples originating from non-border nodes are not necessary to process* $\mathbb{P}(\mathbf{x},\mathsf{Z})$.

Proof. Without loss of generality, assume the non-border node Si detects **x** at time t and derives $P_1(\mathbf{x},\mathsf{Z})$. If **x** fulfills $\mathbb{P}(\mathbf{x},\mathsf{Z})$ at some time $t' > t$, there will be a border node Sj that detects **x** and derives $P_1(\mathbf{x},\mathsf{Z})$ directly followed by $P_2(\mathbf{x},\mathsf{Z})$. If no such border node exists, **x** does not fulfill $\mathbb{P}(\mathbf{x},\mathsf{Z})$ and therefore **x** is irrelevant regarding the users interest. Hence, the detection of a non-border node is irrelevant for developments like $\mathbb{P}(\mathbf{x},\mathsf{Z})$. □

Lemma 11 refers to developments constructed using $\triangleright$ exclusively. Sensor nodes typically have a deep-sleep modus [30] which reduces their energy consumption significantly. This is important since non-border nodes can use deep-sleep while developments like *Enter*$(\mathbf{x},\mathsf{Z})$ are processed.

5.3.1 *Approximation of Detection Neighbors*

As stated in Section 4.1, there exist detection mechanisms where the detection area is indeterminable. In this case, nodes cannot determine their detection neighbors. To solve this problem, we use a superset $\mathsf{DN_i}^{approx}$ which contains at least all detection neighbors $\mathsf{DN_i}$, i.e., $\mathsf{DN_i} \subseteq \mathsf{DN_i}^{approx}$. Using $\mathsf{DN_i}^{approx}$ instead of $\mathsf{DN_i}$ obviously still yields a correct result, because those nodes in $\mathsf{DN_i}^{approx}$ that are not detection neighbors of Si cannot detect an object simultaneously. Several approaches to derive such a superset are conceivable, and we outline two of them:

Communication Neighbors: If the communication range can be assumed to be much larger than the maximum detection range, a valid superset is $\mathsf{CN_i}$, i.e., $\mathsf{DN_i}^{approx} = \mathsf{CN_i}$. In this case, all detection neighbors are communication neighbors as well. This approach is applicable to most detection mechanisms used in WSN, and we use it for our evaluation.

Node Positions: Another approach is applicable if nodes know their position: The set $\mathsf{DN_i}^{approx}$ contains all nodes with a distance of at most $2 \cdot \mathcal{D}_{max}$ to Si.

Next, we propose two strategies which allow a node Si that detects **x** to obtain tuples originating from detection neighbors efficiently to compute the detection scenario.

5.3.2 *Reactive Strategy*

The core idea of the *reactive strategy* is as follows: At query-dissemination time, each node has received $\mathbb{P}(\mathbf{x},\mathsf{Z})$. Each predicate of the development is related to a detection scenario according to Table 1. For instance, for *WSNEnter*$(\mathbf{x},\mathsf{Z})$ each node knows that only $\mathtt{DS^O}$ and $\mathtt{DS^I}$ are rele-

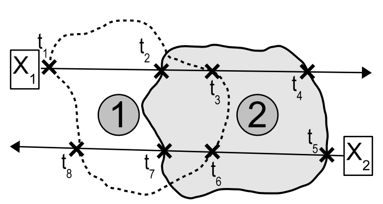

Figure 7. Detection Events ($\mathtt{S1} \in \overline{\mathsf{Z}}$, $\mathtt{S2} \in \mathsf{Z}$)

Reactive		$\mathtt{Si} \in \mathsf{Z}$	$\mathtt{Si} \in \overline{\mathsf{Z}}$
$\mathtt{DS}^{\mathtt{I}}$	Entry	$\mathsf{DN_i}^{\overline{\mathsf{Z}}}$	$\varnothing$
	Exit	$\varnothing$	$\mathsf{DN_i}$
$\mathtt{DS}^{\mathtt{B}}$	Entry	$\mathsf{DN_i}^{\overline{\mathsf{Z}}}$	$\mathsf{DN_i}^{\mathsf{Z}}$
	Exit	$\varnothing$	$\varnothing$
$\mathtt{DS}^{\mathtt{O}}$	Entry	$\varnothing$	$\mathsf{DN_i}^{\mathsf{Z}}$
	Exit	$\mathsf{DN_i}$	$\varnothing$

Table 3. Detection-neighbor partitions for the reactive strategy

Proactive		$\mathtt{Si} \in \mathsf{Z}$	$\mathtt{Si} \in \overline{\mathsf{Z}}$
$\mathtt{DS}^{\mathtt{I}}$	Entry	$\varnothing$	$\mathsf{DN_i}^{\mathsf{Z}}$
	Exit	$\varnothing$	$\mathsf{DN_i}^{\mathsf{Z}}$
$\mathtt{DS}^{\mathtt{B}}$	Entry	$\mathsf{DN_i}^{\overline{\mathsf{Z}}}$	$\mathsf{DN_i}^{\mathsf{Z}}$
	Exit	$\mathsf{DN_i}^{\overline{\mathsf{Z}}}$	$\mathsf{DN_i}^{\mathsf{Z}}$
$\mathtt{DS}^{\mathtt{O}}$	Entry	$\mathsf{DN_i}^{\overline{\mathsf{Z}}}$	$\varnothing$
	Exit	$\mathsf{DN_i}^{\overline{\mathsf{Z}}}$	$\varnothing$

Table 4. Detection-neighbor partitions for the proactive strategy

vant. When an object **x** enters or leaves the detection area of Si at time t, Si checks if this possibly results in a predicate $P(\mathbf{x}, \mathsf{Z})$ of the query being true. If so, Si requests tuples on **x** from some or all of its detection neighbors. Si stores each such tuple and computes the detection scenario after the tuples requested have arrived. If the detection scenario computed results in one predicate of $\mathbb{P}(\mathbf{x}, \mathsf{Z})$ being true, the base station is notified. A core question is: "When Si detects **x**, which detection neighbors could have tuples that are relevant to compute the detection scenario?" This depends on three points:

- The predicates $P(\mathbf{x}, \mathsf{Z})$ that form $\mathbb{P}(\mathbf{x}, \mathsf{Z})$.
- Whether $\mathtt{Si} \in \mathsf{Z}$ or $\mathtt{Si} \in \overline{\mathsf{Z}}$.
- Whether **x** has entered or left the detection area $\mathbf{D}_i$.

Table 3 summarizes from which detection neighbors a node has to request tuples to check if a given detection scenario has occurred when it detects **x**. In the following, we explain these cells, using Figure 7 as an illustration. Figure 7 shows two nodes and their detection areas as well as two objects that enter/leave these detection areas at different times t_i.

The first row of Table 3 is related to $\mathtt{DS}^{\mathtt{I}}$, i.e., *Inside* $(\mathbf{x}, \mathsf{Z}) \in \mathbb{P}(\mathbf{x}, \mathsf{Z})$. There are two cases that can lead to $\mathtt{DS}^{\mathtt{I}}$: (1) an object enters the detection area $\mathbf{D}_i$ of a node $\mathtt{Si} \in \mathsf{Z}$ or (2) an object leaves $\mathbf{D}_j$ of $\mathtt{Sj} \in \overline{\mathsf{Z}}$. For all other detection events, no communication is required, as reflected by the '$\varnothing$' entries. Case (1) occurs at t_2 and t_5 in Figure 7 since $\mathtt{S2} \in \mathsf{Z}$. Applying Lemma 9, S2 only requires tuples from $\mathsf{DN_i}^{\overline{\mathsf{Z}}}$ to compute the detection scenario. The corresponding $\mathsf{DN_i}^{\overline{\mathsf{Z}}}$ entry in Table 3 reflects this. For t_2, $\mathtt{S1} \in \mathsf{DN_i}^{\overline{\mathsf{Z}}}$ returns a tuple $[\mathtt{S1}, \mathbf{x_1}, t_1, \top]$. From this, S2 can derive that S1 and S2 detect $\mathbf{x_1}$ simultaneously, i.e., $\mathtt{DS}^{\mathtt{I}}$ did not occur. Contrary to this, S2 derives $\mathtt{DS}^{\mathtt{I}}$ for $\mathbf{x_2}$ for t_5. Case (2) is different, because objects leave $\mathbf{D}_j$ of $\mathtt{Sj} \in \overline{\mathsf{Z}}$, i.e., Sj does not detect the object any more and thus cannot apply Lemma 9. Hence, Detections stored at Sj has to be complete, i.e., Sj must request tuples from all detection neighbors. This is reflected by the $\mathsf{DN_i}$ entry in the first row of Table 3. This case occurs at t_3 and t_8 in Figure 7 since $\mathtt{S2} \in \overline{\mathsf{Z}}$. In both cases, S1 must verify that no other node outside of Z still detects the object, and that there is at least one node in the zone detecting it. Hence, $\mathtt{DS}^{\mathtt{I}}$ occurs at t_3 but not at t_8.

The second row of Table 3 is related to $\mathtt{DS}^{\mathtt{B}}$, i.e., *Meet* $(\mathbf{x}, \mathsf{Z})$ is part of $\mathbb{P}(\mathbf{x}, \mathsf{Z})$. $\mathtt{DS}^{\mathtt{B}}$ requires simultaneous detection of **x** by nodes inside and outside of the zone. Thus, when an object leaves a detection area, $\mathtt{DS}^{\mathtt{B}}$ either already has occurred or does not occur at all, i.e., no communication is required. Contrary to that, objects entering a detection area can result in $\mathtt{DS}^{\mathtt{B}}$. This allows applying Lemma 9. Thus, if $\mathtt{Si} \in \mathsf{Z}$, only tuples from $\mathsf{DN_i}^{\overline{\mathsf{Z}}}$ are required and vice versa.

The entries for $\mathtt{DS}^{\mathtt{O}}$, i.e., *Disjoint* $(\mathbf{x}, \mathsf{Z}) \in \mathbb{P}(\mathbf{x}, \mathsf{Z})$, are derived analogously to those for $\mathtt{DS}^{\mathtt{I}}$.

Algorithm 5.5: Reactive Strategy

```
1 When x enters or leaves the detection area of Si do
2     Modify Detections as described in Section 5.1;
3     DN* ← Set of detection neighbors that must be
      queried according to Table 3;
4     Request tuples on x from every node in DN*;
5     Wait for response from every node in DN*;
6     Determine detection scenario d according to
      Table 2;
7     Compute predicate result from d based on Table 1;
8     Notify base station if x fulfills a predicate of the
      query;
9 end
```

5.3.2.1 *Reactive Strategy – Summary*

Algorithm 5.5 summarizes the reactive strategy. When **x** enters or leaves the detection area of Si at t, Si modifies Detections accordingly. Afterwards, Si requests tuples on **x** from a set DN^* of detection neighbors. Each response from a detection neighbor is inserted into Detections stored at Si. After all detection neighbors have responded, Si determines the detection scenario.

5.3.3 *Proactive Strategy*

As illustrated in Algorithm 5.5, the reactive strategy requires communication for requesting tuples and for responding to these requests. The proactive strategy tries to avoid responses. Algorithm 5.6 outlines the strategy, and the core idea is as follows: When **x** enters or leaves at t the detection area of Si, Detections stored at Si is modified. This modification is either an insertion of a tuple $[\mathtt{Si}, \mathbf{x}, t, \top]$ or an update of a tuple $[\mathtt{Si}, \mathbf{x}, t' < t, \top]$ to $[\mathtt{Si}, \mathbf{x}, t', t]$ (cf. Section 5.1). Afterwards, Si immediately sends the modified tuple to a subset DN^* of its detection neighbors. Each detection neighbor $\mathtt{Sj} \in \mathsf{DN}^*$ stores the modified tuple. This ensures that Detections stored at Si and each Sj is semi-complete. According to Lemma 9, Si and any $\mathtt{Sj} \in \mathsf{DN}^*$ that currently detects **x** can compute the detection scenario. Again, the important step is determining the set DN^* in Line 3 since it determines the number of messages. Analogously to the reactive strategy, Table 4 lists which detection neighbors must receive an update to ensure semi-completeness, for each detection scenario. We explain each cell in the following using Figure 7.

Algorithm 5.6: Proactive Strategy

```
1  When x enters/leaves detection area of Si do
2      Modify Detections as described in Section 5.1;
3      $\mathsf{DN}^* \leftarrow$ Set of detection neighbors whose
       information must be updated according to Table 4;
4      Send updated tuple(s) to every node in $\mathsf{DN}^*$;
5      Goto Line 9;
6  end
7  When Si receives updated tuples about x do
8      Insert updated tuples into Detections;
9      Determine detection scenario $d$ according to
       Table 2;
10     Compute predicate result from $d$ based on Table 1;
11     Notify base station if x fulfills a predicate of the
       query;
12 end
```

Recall that $\mathtt{DS}^{\mathtt{I}}$ can either occur (1) when an object enters the detection area of a node inside the region or (2) when the detection area of a node outside of the region is left. An object detection conforming to $\mathtt{DS}^{\mathtt{I}}$ requires at least one node $\mathtt{Si} \in \mathsf{Z}$ to detect the object. If such a detection occurs, `Si` must determine if there exists a simultaneous detection by another node $\mathtt{Sj} \in \overline{\mathsf{Z}}$. Using Figure 7 again, Case (1) occurs at t_2 and t_5. To compute the detection scenario correctly at t_2, `S2` must know that $\mathtt{S1} \in \overline{\mathsf{Z}}$ currently detects $\mathbf{x_1}$. Case (2) occurs when $\mathbf{x_1}$ leaves the detection area of `S1` at t_3. In this case, the information at `S2` is updated, and `S1` then correctly determines $\mathtt{DS}^{\mathtt{I}}$ for $\mathbf{x_1}$. Regarding $\mathbf{x_2}$, `S2` computes $\mathtt{DS}^{\mathtt{I}}$ at t_5, because there do not exist any relevant detections by any $\mathtt{Si} \in \overline{\mathsf{Z}}$. Thus, if the query requires $\mathtt{DS}^{\mathtt{I}}$, nodes outside of the zone must send updates to their detection neighbors inside the zone when objects enter/leave their detection areas.

$\mathtt{DS}^{\mathtt{B}}$ requires simultaneous detection by nodes in Z as well as $\overline{\mathsf{Z}}$. Thus, every $\mathtt{Si} \in \mathsf{Z}$ must be informed about detections of detection neighbors in $\overline{\mathsf{Z}}$ and vice versa.

5.4 Node Failures

When a node fails, there are two possible consequences: (1) An object **x** that would have been detected is not detected. (2) Nodes detect **x**, but the detection-scenario computation is possibly incorrect because it is based on an incomplete relation `Detections`. We have shown how users can express queries if they are interested in objects that are temporarily unobserved in Section 4.3. Therefore we focus on (2), i.e., we notify the user if query results returned could be incorrect due to node failures. We discuss the detection of failures first and continue with failure handling.

5.4.1 Failure Detection

It depends on the strategy used for data collection how failures are detected. A node `Si` using the reactive strategy requests tuples from its detection neighbors DN^* and expects a response from each of them. If no such response has been received after a timeout, `Si` derives that the detection neighbors whose responses are missing have failed.

The drawback of the proactive strategy is that nodes cannot detect failures of detection neighbors using missing responses. Without further measures, a failed node might not send updates to detection neighbors and thus affect query results. A practical approach to solve this is sending beacon messages periodically to detection neighbors. If beacon messages are missing from a detection neighbor, nodes will assume a failure and send a notification if this failure could have an impact on the detection scenario computed. Our evaluation includes the additional messages induced by this. Note that this problem also occurs with the centralized strategy, i.e., additional messages are required to detect node failures.

5.4.2 Failure Handling

The user must be notified of a node failure if it could have an impact on the query result, i.e., if the computation of the detection scenario is incorrect. In the following, we refer to the node whose failure has been detected as `Sf`. When `Si` detects the failure of $\mathtt{Sf} \in \mathsf{DN_i}$ and computes a detection scenario later, the result is possibly incorrect. We denote the detection scenario computed based on an incomplete relation `Detections` with DS_{fail}.

LEMMA 12. *If $DS_{fail} = \mathtt{DS}^{\mathtt{B}}$, the failure of* `Sf` *did not affect the computation of the detection scenario.*

Proof. According to Table 1, $\mathtt{DS}^{\mathtt{B}}$ occurs if there exists at least one node in Z and one node outside of Z that detect the object. This is independent from the potential detection of `Sf` and thus the detection scenario computation is not affected by the failure of `Sf`. □

LEMMA 13. *If $DS_{fail} = \mathtt{DS}^{\mathtt{I}}$ and $\mathtt{Sf} \in \mathsf{Z}$ or $DS_{fail} = \mathtt{DS}^{\mathtt{O}}$ and $\mathtt{Sf} \in \overline{\mathsf{Z}}$, the failure of* `Sf` *did not affect the computation of the detection scenario.*

Proof. We prove this only for the case of $DS_{fail} = \mathtt{DS}^{\mathtt{I}}$. The reasoning for $DS_{fail} = \mathtt{DS}^{\mathtt{O}}$ is analogous. $DS_{fail} = \mathtt{DS}^{\mathtt{I}}$ implies that there exists a node $\mathtt{Sj} \in \mathsf{Z}$ that currently detects **x**. Since $\mathtt{Sf} \in \mathsf{Z}$, an additional tuple originating from `Sf` would not change the result of the detection scenario computation. Hence, the failure of `Sf` cannot affect the result. □

Summing up, the base station must be notified of node failures in the following two cases:

- $DS_{fail} = \mathtt{DS}^{\mathtt{I}}$, and $\mathtt{Sf} \in \overline{\mathsf{Z}}$
- $DS_{fail} = \mathtt{DS}^{\mathtt{O}}$, and $\mathtt{Sf} \in \mathsf{Z}$

This notification is a message that contains DS_{fail} and an identifier of `Sf`.

6 Estimating Energy Consumption

In this section we derive a cost model to estimate the number of messages sent and received during the execution of a strategy for a given node. Since communication dominates energy consumption [24], this allows us to pick the most energy-efficient strategy.

For each strategy, the total number of messages is estimated by two addends: $E_{collect}$ estimates the energy consumption for collecting all tuples required to compute the detection scenario. $E_{forward}$ estimates the energy consumption for sending the detection scenario to the base station.

$$E_{total} = E_{collect} + E_{forward} \tag{17}$$

This energy consumption occurs for every object that conforms to the query. Since every correct strategy determines the objects, it is sufficient to look at the communication regarding one object in the following.

For the model, we assume the energy consumption for sending and receiving to be constant and thus introduce two constant cost factors: $\mathtt{e_{snd}}$ represents energy consumption for sending, $\mathtt{e_{rcv}}$ for receiving a message. The function $forward(h)$ estimates the energy consumption for sending a message from one node to another one via $\mathtt{h}$ hops as follows:

$$forward(\mathtt{h}) = \mathtt{h} \cdot (\mathtt{e_{snd}} + \mathtt{e_{rcv}}) \tag{18}$$

The value $\mathtt{h}$ from $\mathtt{Si}$ to the base station is typically available in data structures of routing protocols [27, 12].

LEMMA 14. *Suppose* $\mathtt{Si}$ *is a non-border node and an object* $\mathbf{x}$ *enters and exits the detection area of* $\mathtt{Si}$. $E_{total} = 2 \cdot forward(\mathtt{h})$ *estimates the energy consumption for the centralized strategy and the upper bound for the energy consumption for the distributed strategies is* $E_{total} = forward(\mathtt{h})$.

Proof. Nodes using the centralized strategy notify the base station whenever an object enters or leaves the detection area, i.e., two notifications in this case. Afterwards, computing the detection scenario at the base station does not require any further communication, hence $E_{total} = 2 \cdot forward(\mathtt{h})$. According to algorithms 5.5 and 5.6, a node $\mathtt{Si}$ detecting $\mathbf{x}$ has to exchange messages with the appropriate partition of the measurement neighbors $\mathsf{DN_i}$. If $\mathtt{Si}$ is a non-border node, this partition is empty by definition, i.e., no messages are sent or received. Thus $\mathtt{Si}$ computes the detection scenario from `Detections` according to Table 2. The result of this computation is sent from $\mathtt{Si}$ if the object detected fulfills at least one predicate of the query. The energy consumption for this is estimated by $forward(\mathtt{h})$. These costs only occur once: Either when the object enters or when it leaves the detection area as shown in tables 3 and 4. $E_{total} = forward(\mathtt{h})$ is an upper bound, because $\mathtt{Si}$ does not notify the base station at all if the object does not fulfill at least one predicate of the query. $\square$

Lemma 14 implies that for non-border nodes the energy consumption of the centralized strategies is at least twice as high as with the distributed strategies. Furthermore, for non-border nodes the distributed strategies perform equally well. Thus, for the remainder of this section we focus on estimating energy consumption of border nodes.

6.1 Estimating $E_{forward}$

The centralized strategy does not require any messages to send the result to the base station:

$$E_{forward}^{centralized} = 0 \tag{19}$$

For the other two strategies, after an arbitrary $\mathtt{Si}$ has computed the detection scenario, it sends a message to the base station. Thus, we estimate $E_{forward}$ as follows:

$$E_{forward}^{reactive} = E_{forward}^{proactive} = forward(\mathtt{h}) = \mathtt{h} \cdot (\mathtt{e_{snd}} + \mathtt{e_{rcv}}) \tag{20}$$

6.2 Broadcast vs. Unicast

With the distributed strategies, a border node $\mathtt{Si}$ must send messages to a set of nodes $\mathsf{S}^* \in \{\mathsf{DN_i}^{\mathsf{Z}}, \mathsf{DN_i}^{\overline{\mathsf{Z}}}, \mathsf{DN_i}\}$ depending on its own position. $\mathtt{Si}$ can achieve this using either *unicasts* or *broadcast*. For unicast, this results in $|\mathsf{S}^*|$ separate messages.

$$ucast(\mathsf{S}^*) = |\mathsf{S}^*| \cdot (\mathtt{e_{snd}} + \mathtt{e_{rcv}}) \tag{21}$$

Broadcasting the same message consumes $\mathtt{e_{snd}}$ once, but reaches all communication neighbors $\mathsf{CN_i}$:

$$bcast(\mathsf{CN_i}) = \mathtt{e_{snd}} + |\mathsf{CN_i}| \cdot \mathtt{e_{rcv}} \tag{22}$$

Thus, it depends on $|\mathsf{S}^*|$ and $|\mathsf{CN_i}|$ if $\mathtt{Si}$ should use broadcast or unicast. The function $neighbors(\mathsf{S}^*, \mathsf{CN_i})$ estimates the respective energy consumption:

$$neighbors(\mathsf{S}^*, \mathsf{CN_i}) = min(ucast(\mathsf{S}^*), bcast(\mathsf{CN_i})) \tag{23}$$

6.3 Estimating $E_{collect}$

Every border node $\mathtt{Si}$ using the reactive strategy requests tuples from a set of nodes S^*. Each recipient sends a response, which $\mathtt{Si}$ receives. Equation (24) reflects this.

$$reactive(\mathsf{S}^*, \mathsf{CN_i}) = neighbors(\mathsf{S}^*, \mathsf{CN_i}) + |\mathsf{S}^*| \cdot (\mathtt{e_{snd}} + \mathtt{e_{rcv}}) \tag{24}$$

If $\mathtt{Si}$ uses the proactive strategy, messages are only sent when updating information on detection neighbors S^*.

$$proactive(\mathsf{S}^*, \mathsf{CN_i}) = neighbors(\mathsf{S}^*, \mathsf{CN_i}) \tag{25}$$

Nodes using the centralized strategy send all tuples to the base station.

$$centralized(\mathtt{h}) = forward(\mathtt{h}) \tag{26}$$

To compare the communication required to compute a given detection scenario, the number of times each strategy is triggered must be taken into account. While the exact number depends on the object movement, we can derive ratios from Tables 3 and 4: The centralized strategy notifies the base station whenever an object enters or leaves a detection area. Thus, any detection event consumes $centralized(\mathtt{h})$. Contrary to this, for $\mathtt{DS^I}$ and $\mathtt{DS^O}$, both distributed strategies do not cause any communication in 50% of the cases.

$$E_{collect}^{centralized}(\mathtt{DS^I}) = centralized(\mathtt{h}) \tag{27}$$

$$E_{collect}^{proactive}(\mathtt{DS^I}) = 0.5 \cdot proactive(\mathsf{S}^*, \mathsf{CN_i}) \tag{28}$$

$$E_{collect}^{reactive}(\mathtt{DS^I}) = 0.5 \cdot reactive(\mathsf{S}^*, \mathsf{CN_i}) \tag{29}$$

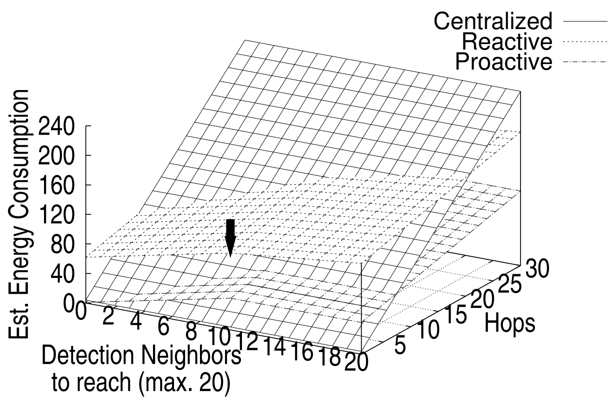

Figure 8. Exemplary Estimation of Energy Consumption

Formulas for $\mathtt{DS}^{\mathtt{B}}$ and $\mathtt{DS}^{\mathtt{O}}$ are derived similarly but omitted here for lack of space.

Example 5: Figure 8 shows the estimated energy consumption for $\mathtt{DS}^{\mathtt{I}}$, a varying size of S^* and a varying distance to the base station. It shows two facts: First, the centralized approach is best for nodes close to the base station, which is expected. This changes with increasing distance to the base station, because notifications of object detections require more hops to reach the base station. Second, for nodes beyond a distance of 5-10 hops, the proactive strategy is better than the centralized or the reactive strategy. This is expected as well, since objects leaving the detection area of a node $\mathtt{Si} \in \overline{\mathsf{Z}}$ must request tuples from all detection neighbors $\mathsf{DN_i}$ for the reactive strategy. Contrary to that, the proactive strategy requires only sending updates to $\mathsf{DN_i}^{\mathsf{Z}}$. The edge marked with an arrow indicates the size S^* must have so that switching from unicast to broadcast is more energy-efficient. □

7 Evaluation

We have evaluated our approach thoroughly using simulations and a Sun SPOT deployment to investigate the following hypotheses:

H1 Both distributed strategies scale better with the number of nodes than the centralized strategy.

H2 The proactive strategy is the most energy-efficient for $\mathit{Inside}(\mathbf{x}, \mathsf{Z})$ and $\mathit{Disjoint}(\mathbf{x}, \mathsf{Z})$.

H3 The reactive strategy is the most energy-efficient for $\mathit{Meet}(\mathbf{x}, \mathsf{Z})$.

H4 The centralized strategy is energy-efficient for small networks and nodes around the base station.

H5 Distributed strategies reduce communication required for processing spatio-temporal developments like $\mathit{Enter}(\mathbf{x}, \mathsf{Z})$ or $\mathit{WSNEnter}(\mathbf{x}, \mathsf{Z})$.

7.1 Simulation Setup

To run exactly the same software for simulations and case study, we used the Sun SPOT simulator of the KSN project [3]. Each simulation run consists of the following steps:

1. Generate a WSN of 100-300 nodes that are randomly deployed over an area. The size of the area is constant to account for different node densities, i.e., varying numbers of detection and communication neighbors.
2. Define a zone of varying size. Zones contain between 2 and 30 nodes.
3. Generate 50 different object paths using a random walk model with starting points randomly chosen.
4. For each object path evaluate each detection scenario using each strategy.
5. Count the number of messages sent and received.

Overall, the results presented here are based on more than 100.000 simulation runs.

Since detection areas tend to be indeterminable, we have approximated the set of detection neighbors with the set of communication neighbors: To do so, a node sends a beacon message periodically. Each node receiving it adds the sender to the list of detection neighbors. We graph the communication required for these beacons for distributed strategies separately. For the proactive approach, these periodic beacon messages would allow the detection of failures and notification of the base station as well.

7.2 Simulation Results

Figure 9 shows the average number of messages per simulation run for WSN of 100-300 nodes to compute $\mathtt{DS}^{\mathtt{I}}$. Graphs for other detection scenarios are similar and omitted here. As expected, the number of messages required by the centralized strategy increases linearly with network size. Contrary to this, network size only affects both distributed strategies marginally. The reason for this is the increasing node density, i.e., more detection neighbors per node. Even the added overhead for the approximation of detection neighbors does not change this. The large share of communication related to detection-neighbor approximation suggests that more sophisticated mechanisms for this could reduce energy-consumption even further. Thus, we conclude that **H1** is true. Detection-neighbor approximation should be investigated in future work.

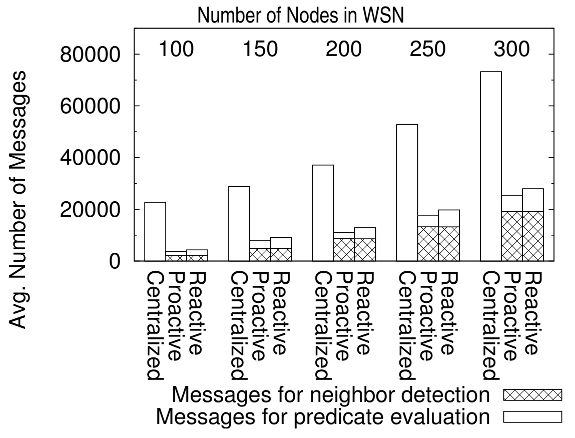

Figure 9. Scalability of data-collection strategies

Figure 10 shows the average number of messages per detection-scenario computation. The result is that distributed strategies require between $45\% - 85\%$ less messages than the centralized strategy. Comparing both distributed strategies shows that the proactive strategy is advantageous for $\mathtt{DS}^\mathtt{I}$ and $\mathtt{DS}^\mathtt{O}$. This is expected, because S^* is smaller for the proactive strategy when objects leave the detection area of a node (cf. Tables 3 and 4). These roles are reversed for $\mathtt{DS}^\mathtt{B}$, because the proactive strategy is triggered more often than the reactive one. Summing up, these results confirm **H2** and **H3**.

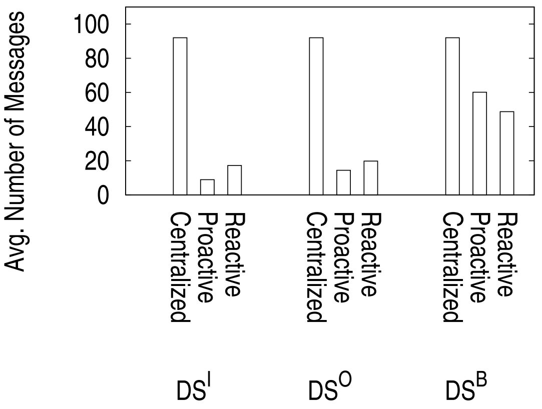

Figure 10. Communication per detection-scenario computation

The distributed strategies reduce communication to process spatio-temporal developments as well. Table 5 shows the average number of messages to determine that $\mathbf{x}$ conforms to $Enter(\mathbf{x}, \mathsf{Z})$ (see (9)) or $WSNEnter(\mathbf{x}, \mathsf{Z})$ (see (15)), respectively. As expected, the centralized strategy requires at least twice as much communication since every detection event must be forwarded to the base station. For $WSNEnter(\mathbf{x}, \mathsf{Z})$, the proactive strategy is most efficient. This is because this development does not contain $Meet(\mathbf{x}, \mathsf{Z})$. The difference between $Enter(\mathbf{x}, \mathsf{Z})$ and $WSNEnter(\mathbf{x}, \mathsf{Z})$ must be attributed to Lemma 11 because all non-border nodes are basically inactive for $Enter(\mathbf{x}, \mathsf{Z})$. Compared to the centralized strategy the savings of distributed strategies are between 51% and 89%. This confirms **H5**.

Strategy	**Number of Messages per Object for**	
	$Enter(\mathbf{x}, \mathsf{Z})$	$WSNEnter(\mathbf{x}, \mathsf{Z})$
centralized	334	334
proactive	44,3	123,8
reactive	39,1	163,1

Table 5. Avg. number of messages for $Enter(\mathbf{x}, \mathsf{Z})$ **and** $WSNEnter(\mathbf{x}, \mathsf{Z})$

7.3 Sun SPOT Case Study

Since simulations always abstract from certain real-world phenomena and these may impact performance, e.g., interferences or collisions, we conducted a case study using real sensor nodes. For our case study, we have deployed 26 Sun SPOT sensor nodes and a base station on our office floors. The query was $Inside(\mathbf{x}, \mathsf{Z})$. In analogy to the simulations, we assumed that nodes cannot determine their detection areas by themselves. Thus, a node periodically sent beacons to approximate the set of its detection neighbors, i.e., $\mathsf{DN_i} = \mathsf{CN_i}$.

Strategy	**Number of Messages**		
	Collect	Result Forward.	Total
centralized	137	0	137
proactive	115	42	157
reactive	145	33	178

Table 6. Case study results

Table 6 shows the result of the case study: The rightmost column contains the total number of messages sent, i.e., the sum of the two columns in the middle which reflect messages for data collection (left) and result forwarding (right). Since the centralized strategy computes all results at the base station, the number of messages sent to forward the result is 0. A simulation that replicated the node setup and object movement of the case study had the same results. This confirms that the findings based on simulations are not significantly changed by real-world phenomena from which the simulations abstracted.

The centralized strategy required 137 messages, the distributed approaches a few more. Considering the cost model, this is expected due to the relatively small network: Messages required 5 hops at most to reach the base station. Considering the simulation results and the result of the case study, we conclude that **H4** is true.

Further analysis shows that the approximation $\mathsf{DN_i}^{approx} = \mathsf{CN_i}$ has resulted in an over assessment: Prior to the experiment, we determined the number of detection neighbors $|\mathsf{DN_i}|$ for every node by calibration. Approximately 50% of the communication neighbors were not detection neighbors. While this does not affect the result, it increases the number of messages sent for data collection. Thus, while the simple approximation $\mathsf{DN_i}^{approx} = \mathsf{CN_i}$ yields correct results, the potential for further reduction of energy consumption by improving detection-neighbor approximation is large.

Summing up, our evaluation confirms all of our hypotheses.

8 Conclusions

For many applications, WSN are used to track moving objects. Research has shown that accessing WSN declaratively is important. But research so far has focused on relational queries which are insufficient to express spatio-temporal semantics inherently required by these applications. This paper is the first that addresses the processing of declarative, spatio-temporal queries based on object detections of WSN. First, we defined the fundamental concepts of spatio-temporal queries in WSN and the semantics of spatio-temporal predicates for zones. Second, we have provided a space partitioning for zones which allows the application of the 9-intersection model and other existing research to WSN. Processing of spatio-temporal queries requires nodes

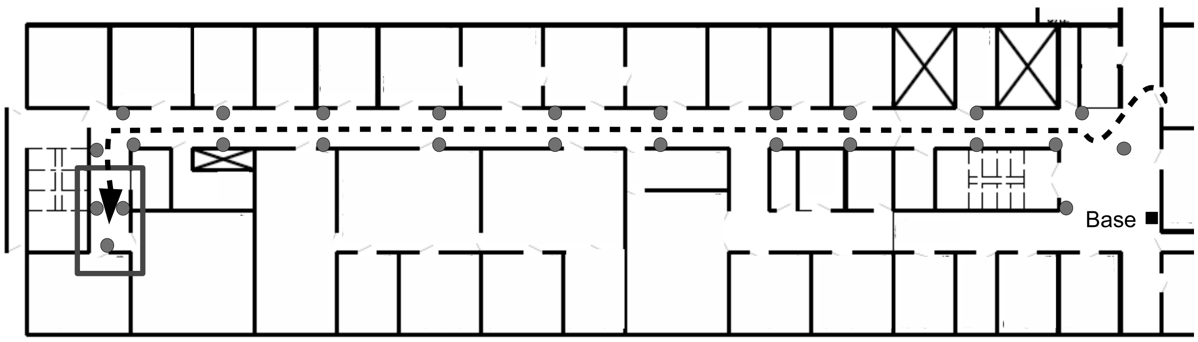

Figure 11. Case Study: Sun SPOT positions and object path

to exchange information on objects detected. We have shown how to reduce the communication for this exchange significantly and proposed two execution strategies for processing predicates in-network. Based on the different execution strategies, we derive a cost model, which allows each node to determine an energy-efficient strategy for the execution of spatio-temporal queries based on easy-to-acquire information about its surroundings. Our strategies can deal with node failures that could affect the query result. This is important for the applicability of our approach for outdoor deployments where nodes may be subject to unfavorable influences that lead to sporadic or permanent failures. We have evaluated our approach using both simulations and a Sun SPOT deployment. Our evaluation shows that distributed strategies perform well for WSN consisting of many nodes in particular.

References

[1] A. Ailamaki et al. An Environmental Sensor Network to Determine Drinking Water Quality and Security. *SIGMOD Rec.*, 2003.

[2] A. Arora et al. A line in the sand: A wireless sensor network for target detection, classification, and tracking. *Computer Networks*, 2004.

[3] M. Bestehorn. Karlsruhe Sensor Networking Project, 2010. http://www.ipd.kit.edu/KSN/.

[4] M. Bestehorn et al. Deriving Spatio-Temporal Query Results in Sensor Networks. In *SSDBM*, 2010.

[5] P. Bonnet et al. Querying the Physical World. *Personal Communications, IEEE*, 2000.

[6] P. Bonnet et al. Towards sensor database systems. In *MDM '01*, 2001.

[7] D. Chu et al. The Design and Implementation of a Declarative Sensor Network System. In *SenSys*, 2007.

[8] J. Considine et al. Approximate aggregation techniques for sensor databases. In *ICDE '04*, 2004.

[9] M. J. Egenhofer and R. D. Franzosa. Point set topological relations. *IJGIS*, 1991.

[10] M. Erwig and M. Schneider. Developments in spatio-temporal query languages. In *DEXA*, 1999.

[11] M. Erwig and M. Schneider. Spatio-temporal predicates. *IEEE TKDE*, 2002.

[12] R. Fonseca et al. The collection tree protocol (ctp), 2007.

[13] L. Forlizzi et al. A data model and data structures for moving objects databases. In *SIGMOD*, 2000.

[14] S. Gaal. Point set topology. *Academic Press*, 1964.

[15] C. Gamage et al. Security for the mythical air-dropped sensor network. In *ISCC*, 2006.

[16] R. H. Güting et al. A Foundation for Representing and Querying Moving Objects. *ACM TODS*, 2000.

[17] R. H. Güting et al. Modeling and querying moving objects in networks. *VLDB J.*, 2006.

[18] A. Hergenröder, J. Wilke, and D. Meier. Distributed Energy Measurements in WSN Testbeds with a Sensor Node Management Device (SNMD). Feb. 2010.

[19] W. Koenig et al. Detectability, Philopatry, and the Distribution of Dispersal Distances in Vertebrates. *Trends in Ecology & Evolution*, 1996.

[20] Y. Kotidis. Processing proximity queries in sensor networks. In *DMSN '06*, 2006.

[21] P. Langendorfer et al. A Wireless Sensor Network Reliable Architecture for Intrusion Detection. In *NGI*, 2008.

[22] N.-H. Liu et al. Long-term animal observation by wireless sensor networks with sound recognition. In *WASA '09*, 2009.

[23] S. Madden et al. Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS OSDI*, 2002.

[24] S. Madden et al. TinyDB: An Acquisitional Query Processing System for Sensor Networks. *ACM TODS*, 2005.

[25] M. U. Mahfuz and K. M. Ahmed. A Review of Micro-Nano-Scale Wireless Sensor Networks for Environmental Protection: Prospects and Challenges. *STAM*, 2005.

[26] J. M. Metsaranta. Assessing Factors Influencing the Space Use of a Woodland Caribou Rangifer Tarandus Caribou Population using an Individual-Based Model. *Wildlife Biology*, 2008.

[27] C. E. Perkins et al. Internet Connectivity for Ad Hoc Mobile Networks, 2002.

[28] J. W. Rettie and F. Messier. Hierarchical Habitat Selection by Woodland Caribou: Its Relationship to Limiting Factors. *Ecography*, 2000.

[29] M. A. Sharaf et al. Balancing energy efficiency and quality of aggregate data in sensor networks. *The VLDB Journal*, 2004.

[30] E. Shih et al. Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks. In *MOBICOM '01*, 2001.

[31] SUN Microsystems Inc. SPOT, 2009.

[32] R. B. Tilove. Set Membership Classification: A Unified Approach to Geometric Intersection Problems. *IEEE TC*, 1980.

[33] G. Trajcevski et al. The geometry of uncertainty in moving objects databases. In *EDBT*, 2002.

[34] G. Trajcevski et al. Managing uncertainty in moving objects databases. *ACM TODS*, 2004.

[35] O. Wolfson et al. Moving objects databases: Issues and solutions. *SSDBM*, 1998.

[36] XBow Technology Inc. Wireless sensor networks, 2009.

[37] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. *SIGMOD Rec.*, 2002.

[38] M. L. Yiu et al. Retrieval of spatial join pattern instances from sensor networks. *Geoinformatica*, 2009.

A Sun Smart Programmable Object Technology (SPOT)

This section provides a brief description of the sensor nodes used during our experiments. [31] provides more detailed information and technical specifications of the hardware as well as the software used.

Figure 12. Sun SPOT sensor node

Sun SPOTs consist of three parts as shown in Figure (from top to bottom): The *sensor board*, a *main board* and a battery. The main board contains a 32-bit ARM920T processor which executes at 180MHz max. clock speed. Furthermore, 512KB RAM and 4MB flash memory are used to store data. Even though access to flash memory consumes non-negligible amounts of energy, we disregard this, since all our mechanisms run exclusively on RAM. For communication, SPOTs use a CC2420 radio chip which is IEEE 802.15.4 compatible ("ZigBee"). Software for SPOTs is written in Java and executed using a Squawk virtual machine which is specifically designed for platforms such as sensor nodes.

The default sensor board contains sensors for light, temperature and acceleration. Furthermore, the sensor board has pins which can be used to attach almost any kind of sensing or detection hardware. For real deployments, these pins could be used to attach detection mechanisms customized to the application.

SPOTs are powered by a rechargeable 3.7V lithium-ion battery with a capacity of 750 mAh.

B Energy Consumption in WSN

This section provides results of energy measurements taken for Sun SPOT sensor nodes. It shows that reducing the number of messages sent and received is important to prolong the lifetime of battery-powered sensor nodes. More precisely, the following hypotheses are proven experimentally:

H1 Exchanging information via wireless communication reduces the time until batteries are depleted significantly.

H2 Energy consumption for sending a message is marginally higher than receiving a message.

H3 The number of bytes contained in a single message has a minor impact on energy consumption.

At first, we describe the setup and then present results which underline above hypotheses.

B.1 Experimental Setup

To measure the energy consumption of Sun SPOT sensor nodes we used the hardware introduced in [18]. The measurement unit was attached between the battery and the other components of the sensor nodes, e.g., CPU, memory, wireless communication chip and sensing board. Figure 13 illustrates a simplified circuit diagram of the setup. We measured the voltage

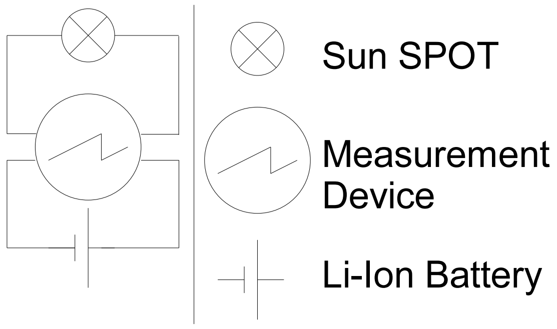

Figure 13. Circuit diagram for energy measurements

drawn by the node at a high temporal resolution of up to 20 kHz and computed the energy consumption based on this.

B.2 Results and Analysis

In this section we discuss the results of the energy measurement displayed in Figures 14 and 15 with regard to our hypotheses.

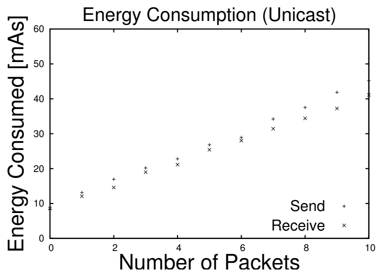

Figure 14. Energy Consumption for communication

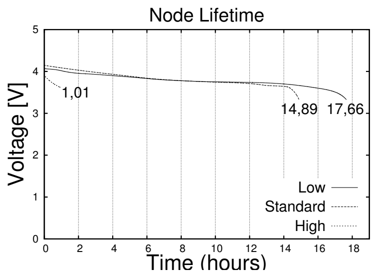

Figure 15. Node lifetime measurement result

To prove **H1**, we fully charged the batteries of three SPOTs according to the specification of the battery. Afterwards, each SPOT executed three different applications: The application "high" was permanently using the communication interface and CPU. The "medium" application sent data every 5 minutes and then put the SPOT into shallow sleep mode to conserve energy. The third application "low" only put a node into shallow sleep mode for the whole experiment. Figure 15 shows the measured voltage over time (in hours). The result proves **H1**, as the usage of the communication interface significantly reduces the lifetime of the sensor nodes: The application "high" depleted the battery an hour while each of the other applications ran 15 hours or more. Regarding the absolute values in Figure 15 it must be noted, that in shallow sleep certain parts of the hardware are still switched on and consume energy. To save more energy, SPOTs also support a deep sleep mode which allows lifetimes of up to 900 hours.

To show that **H2** and **H3** are valid, we conducted the following experiment whose results are displayed in Figure 14: One node was sending messages to a specified node. The size of the messages was increased from 1 packet to 10 packets and for each message, the energy consumed was measured on both nodes. The result in Figure 14 shows the following:

- The difference between sending and receiving is relatively small and constant over all messages sizes.
- The size of the message has a minor impact on energy consumption. For example sending a message consisting of a single packet consumed 13.14 mAs. Doubling the message size to two packets only increases the energy consumption to 16.94 mAs, i.e., approximately 28%.

We conclude that **H2** as well as **H3** are valid.