Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2014, Article ID 245924, 20 pages
http://dx.doi.org/10.1155/2014/245924

Research Article

In-Network Filtering Schemes for Type-Threshold Function Computation in Wireless Sensor Networks

Guillermo G. Riva[1,2] and Jorge M. Finochietto[2,3]

[1] *Universidad Tecnológica Nacional, Facultad Regional Córdoba, Maestro Lopez y Cruz Roja Argentina, X5016ZAA Córdoba, Argentina*
[2] *CONICET, Haya de la Torre S/N, Ciudad Universitaria, 5016 Córdoba, Argentina*
[3] *Universidad Nacional de Córdoba, Velez Sarsfield 1611, Ciudad Universitaria, X5016GCA Córdoba, Argentina*

Correspondence should be addressed to Guillermo G. Riva; griva@scdt.frc.utn.edu.ar

Received 2 February 2014; Revised 2 July 2014; Accepted 2 July 2014; Published 14 August 2014

Academic Editor: Jaime Lloret

Data collection in wireless sensor networks (WSNs) can become extremely expensive in terms of power consumption if all measurements have to be fetched. However, since multiple applications do not require data from all nodes but to compute a function over a smaller data set, much of the available data on the network can be considered irrelevant and not worthy of spending energy. In this context, in-network filtering schemes can be used to forward only relevant data towards a sink node for processing purposes. In this work, we propose and evaluate two schemes that can drive this filtering process. Both of them are based on the integration of metaheuristics and learning algorithms inspired by nature. In particular, we consider the computation of the maximum function as case study for these schemes. We investigate the trade-off between communications costs, which are directly associated with power consumption, and error costs due to fetching not all relevant data. We show by simulation that communication costs can be significantly reduced with respect to traditional schemes while keeping the computation error bounded.

1. Introduction

A wireless sensor network (WSN) consists of a set of small and low cost sensor nodes connected by wireless links, which can be deployed to obtain information about physical phenomena such as heat, light, noise, radiation, and chemical emissions. Typically, sensor readings are sent to a collector node (i.e., sink) using multihop forwarding, according to a data delivery model [1–3]. However, sensor nodes can have limited sensing, processing, and communication capabilities due to energy constraints resulting from battery operation. Hence, energy efficiency is a key issue in the design of systems based on this technology, where data communication can typically be considered the most energy demanding task.

Besides forwarding packets, WSN nodes are typically allowed to process in transit data available on the payload of these packets. Data can be aggregated or filtered (i.e., discarded) in order to decrease the amount of packets traveling through the network; thus, saving energy by reducing communication costs [4]. While data aggregation, which aims at fusing redundant data, has been thoroughly investigated [5], data filtering has received much less attention. In particular, the computation of *type-sensitive* and *type-threshold* functions [6, 7] can significantly benefit from in-network filtering schemes. The former requires to know a minimum fraction of arguments for the function value to be determined. Examples include average, median, histogram, and majority. The latter depends only on element-wise maximum (minimum) of the histogram and a threshold vector. Instances of these functions are maximum, minimum, range, bottom-n, and top-n. Intuitively, the value of a type-sensitive function can be accurately determined if a large fraction of the arguments are known, whereas the value of a type-threshold function can be determined by a smaller amount of arguments. Consequently, type-threshold function can potentially be computed with low communication costs.

A major issue in type-threshold function computation is *how to select a set of arguments* (i.e., *node readings) for the function value to be accurately determined*. The most naive solution is to consider all arguments, thus all network data without any kind of filtering. This guarantees the accuracy of the computed function value but involves querying all nodes and forwarding all readings to a sink for computation, which can demand a huge communication effort as the network size increases. Ideally, there exist a minimal set of nodes which can return the actual function value, but this set is unknown a priori. Since functions are typically computed over time, a learning scheme could be used to select a set of arguments that can provide with high probability the function value. If so, only a limited number of nodes can actually be queried for data, which can reduce the message count required to compute the given function. In this way, it is possible to implement low-latency *one-shot* computation schemes by issuing a single query at regular intervals that fetches relevant data only.

In this work, we consider the problem of filtering inside the network, those arguments not actually required to compute a type-threshold function. To this end, only the set of arguments which can be used to determine the function value at the sink node are selected. Indeed, this selection constitutes a precomputation of the function value for type-threshold functions, which can be performed inside the network in order to save energy. The network can provide the best argument candidates for a given function to a sink, which can then process these data to obtain the function value. For the maximum (minimum) function computation, just the argument with maximum (minimum) value could be provided, while, for the range case, all arguments belonging to the range. However, in general, we may provide (i) a larger set, which can result in a high communication cost but no computation error, or (ii) a smaller set, which results in a computation error. A trade-off between cost and error is thus present in solutions addressing this problem.

Type-threshold functions can be divided into two categories based on how the threshold level is defined. *Fixed-threshold functions* define a threshold which is known a priori and can be embedded on the query as a constant value. Nodes only read this threshold to determine whether their readings are relevant or not. The range and isocontour functions are good example of this category. On the other hand, *dynamic-threshold* defines a threshold which is embedded on the query as a variable value. Nodes can not only read but also update this threshold. Among the functions belonging to this category, the maximum function is the most interesting example as it can be used to implement other functions such as minimum, top-n, and bottom-n. Besides, the maximum function is typically required in several WSN applications. In this work, we focus on filtering schemes which can be used for computing the maximum function.

Dynamic-threshold functions require filtering schemes that can adapt and learn from the network. To this end, we propose to integrate computational intelligence techniques (CI) on nodes. CI is a set of nature-inspired computational methodologies to address complex problems of the real world to which traditional methodologies are ineffective or infeasible. In this work, these techniques are used to provide nodes with *two filtering rules*, as shown in Figure 1. The first one, namely, *self-filtering rule*, determines whether a reading constitutes a relevant argument to the function. The second rule, namely, *neighbor-filtering rule*, determines if neighbor nodes can have relevant arguments or not. In other words, if neighbor data are assumed irrelevant, the query message (containing the function) is only forwarded with some (low) probability. These rules are not static (i.e., preconfigured) but *built on the fly* by means of learning mechanisms.

Nodes implement these simple rules iteratively resulting in a complex network behavior, where the network evolves from a nonfiltering state to a filtering one by means of independent decisions made by its nodes. In particular, our work considers two different approaches to implement the neighbor-filtering rule. The first scheme is based on the simulated annealing (SA) concept and centralizes network learning at the sink node to update a global query forwarding policy which fetches data from relevant areas. Instead, the second strategy distributes learning on network nodes in order to locally update the query dissemination policy at each node. This is achieved by means of the ant colony optimization (ACO) algorithm which is a bioinspired scheme based on the behaviour of ants seeking a path between their colony and a source of food. In our context, relevant data seeks a path to the sink node. When implemented over time, these schemes can learn from the network and dynamically provide a set of arguments that offers good performance in terms of communication cost and computation error.

This paper significantly extends our previous analysis reported in [8, 9], providing a much deeper discussion of the proposed schemes as well as new results. Besides, both schemes are compared to identify benefits and limitations, which can determine the best application scenario for each proposal. The main contributions of this work are as follows.

(i) A detailed description of how filtering schemes can be used in the context of in-network computation is presented. General approaches as well as particular solutions are discussed.

(ii) Proposed schemes are described in terms of forwarding and learning processes, which helps to analyze its behavior and compare their performance.

(iii) Different simulation scenarios are evaluated, thus enhancing result analysis and strengthening conclusions.

The rest of the paper is organized as follows. Section 2 briefly discusses related work on in-network computation with special focus on query mechanisms. Section 3 formalizes the problem we consider in this work and introduces the concept of filtering as an efficient solution. Section 4 describes the general network model and the behavior of nodes. Forwarding and learning algorithms used to implement neighbor-filtering rules are explained in Section 5. Section 6 discusses main results obtained by simulation. Finally, Section 7 concludes the work.

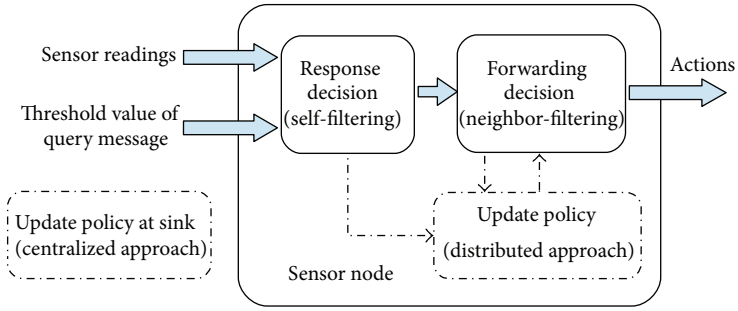

Figure 1: Decision rules at each sensor node.

2. Related Work

In-network computation of symmetric functions in multihop random WSNs has been discussed broadly in [6, 7]. In particular, the use of gossip-based algorithms has been identified as suitable solutions when both robustness and fault tolerance are required. These schemes have mainly considered type-sensitive functions (i.e., average) [10, 11] which are computed through data aggregation. However, a different approach can be considered for type-threshold functions, which actually require data searching rather than aggregation. In this sense, data aggregation techniques can be used to reduce the communication costs in WSN applications where a large amount of data needs to be sent from source nodes to a sink one (i.e, in many-to-one data flows) [2]. However, in case of type-threshold functions, where often only few nodes can possess arguments for computing a given function, these techniques can be inefficient. For this reason and because nodes with relevant readings are unknown a priori, our problem is more related to *data search/discovery problems* than to *data collection* ones.

Several query processing schemes for WSNs have been proposed in the literature, which can be classified from the point of view of both the required *infrastructure* and the *query dissemination* mechanism. The former can be infrastructure-based (such as tree, clustering, etc.) [12, 13] or infrastructure-free (i.e., unstructured) [13–15]. The latter can be walk-based [13–15], random walk-based [16], flooding-based [17, 18], and gradient-based [17]. In this sense, unstructured- and gradient-based schemes are the best option in case of type-threshold function computation due to the low overhead of unstructured mechanisms and the information gain concept of gradient-based strategies for query dissemination. The concept of using fingerprint gradients to direct query dissemination to nodes detecting events was proposed in [17]. This strategy is based on the fact that every physical event produces a fingerprint in the environment which results in a natural information gradient in the proximity of the phenomenon.

Most query processing schemes used in WSNs are based on *one-phase pull diffusion* [19] and rely on the network infrastructure for query propagation and data collection, which enables in-network processing for data reduction [14] (i.e., data aggregation). First, the sink node disseminates once an interest described as a function computation task. Besides, this query sets up backward paths from each sensor node to the sink. Afterwards, sensor nodes with relevant data for this function can either send data to the sink for computing the function value in a centralized way or distributively compute the function inside the network to deliver to the sink the final result. This last process can be repeated after a period of time to update the function value at the sink. However, since WSNs can be considered unreliable networks where nodes are prone to failure, these schemes can suffer from topology changes which can break up backward paths.

In this work, we focus on type-threshold function computation problems using *one-shot on-demand query-based schemes*. These types of schemes are much more robust to network changes since queries are disseminated each time the function is to be computed; thus backward paths are set up on-demand after each query. Note that this scheme, at first glance, might seem to demand high communication costs; however, if query dissemination can be combined with both *gradient-based routing strategies* for data searching and *in-network filtering schemes* query communication costs can be reduced. In this sense, each node, based on simple rules, participates in this process by filtering irrelevant readings as well as neighbor nodes not necessary for the function computation. As a consequence, the query area can be reduced iteratively, and the query dissemination can be directed only to those nodes which can give the best arguments for the function computation.

Our first proposed scheme drives the query dissemination process using a simulated annealing (SA) metaheuristic. Kirkpatrick et al. [20] proposed the SA algorithm to deal with traveling salesman and component placement problems. SA is a nature-based probabilistic metaheuristic for the global optimization problem of locating a good approximation to the global optimum of a given function in a large search space. Due to the distributed nature of WSNs, algorithms such as SA can be implemented in parallel fashion [17, 21], in order to compute type-threshold functions in the network. Our second proposed scheme is based on computational intelligence, in particular, bioinspired mechanisms. Kulkarni et al. [22] proposed reinforcement learning (RL) and swarm intelligence (SI) as the best options in WSNs from the point of view of computational and memory requirements, flexibility, and optimality. RL is biologically inspired and acquires its

knowledge by actively exploring its environment [23]. In the last years, communication and networking technologies are increasingly considered to integrate bioinspired strategies as robust and efficient solutions [24].

A current branch of swarm intelligence focuses on *ant colony optimization (ACO)* and pheromone-based mechanisms. In this sense, pheromone-based routing strategies mimic the behavior of ant colonies when searching for food. Upon finding food sources, ants return to their colony laying down pheromone trails. Other ants tend to follow these paths and to reinforce them releasing more pheromone. Over time, pheromone tends to evaporate to erase unused paths. Most pheromone-based strategies for WSNs address routing and aggregation problems [25, 26]. To the best of our knowledge, the problem of in-network filtering for computing type-threshold functions using bioinspired schemes in WSNs has not yet been treated.

3. Problem Formulation

Most WSN applications are required to compute a function over sensed data (i.e., measurements). We can formalize this as computing a function $f(X)$ where X is the set of readings from x_1 to x_n, with n being the number of sensor nodes in the network. In general, all x readings could be made available to the sink node in order to compute the function value in a centralized manner. In this case, sink incurs no computation error at the expense of a high communication cost (i.e., energy consumption), especially in large scale WSNs. However, type-threshold functions can typically be computed over a subset $X' \subseteq X$ of readings such that $f(X') = f(X)$. Thus, the problem of finding the set X' becomes relevant as it has the potential to compute the function value at lower communication costs.

Let $\widehat{X}' = g(X)$ be the set of arguments provided by a given $g(\cdot)$ in-network filtering function. If $\widehat{X}' \supseteq X'$, $g(\cdot)$ does not introduce any computation error. However, if $\widehat{X}' \not\supset X'$, the filtering adds some computation error $e = |f(X') - f(\widehat{X}')|/f(X')$. The communication cost c is harder to estimate but it is expected to be inversely proportional to $|\widehat{X}'|$ (i.e., the size of the reported set). This is due to the fact that as $\widehat{X}' \to X'$, the more message exchange is required by the $g(\cdot)$ filtering scheme. Consider, for example, flooding the network with a query related to computing a given range of values. Each node whose reading belongs to the range can report its value to the sink node. Since all nodes receive the query, then $\widehat{X}' = X'$ and $|\widehat{X}'| = |X'|$. However, if the query scheme is based on random walk, not all relevant nodes may receive the query message. Thus, we expect $\widehat{X}' \not\supset X'$ with $|\widehat{X}'| < |X'|$, which has some computation error but a lower communication cost since not all the network was explored.

The computation of a function $f(\cdot)$ can then be performed in two parts: a first one, given by $\widehat{X}' = g(X)$, which is computed inside the network, and a second one, given by $f(\widehat{X}')$, which is done at the sink node. Both processes are described in Figure 2. Since $g(\cdot)$ is to be implemented in a distributed fashion, it can be governed by two filtering rules: a *self-filtering* one, which decides whether the node's reading is relevant or not for the computation of the function $f(\cdot)$, and a *neighbor-filtering rule*, which determines if it is worth forwarding the query to neighbor nodes. Note that if the query is not forwarded, these nodes can be potentially excluded from the in-network computation process as their data are actually not even considered for reporting to the sink node.

The self-filtering rule is straightforward as it only requires evaluating the local data against a threshold available on the query message. However, as discussed on Section 1, fixed and dynamic thresholds can be considered. Fixed thresholds [27, 28] are used for searching readings within a specific range of values, while dynamic ones, for readings without a predefined range. Fixed thresholds can be implemented locally by each node despite the actual readings available at other nodes, while dynamic ones depend on these readings to update the threshold values [29]. The canonical example for a dynamic threshold is the maximum function where, given an initial threshold, all nodes whose readings are above the threshold update it on the query message. In this way, the threshold tends to increase as the query message is disseminated through the network. In general, we will consider for our analysis the case of the maximum (minimum) function since it makes use of dynamic thresholds; however, the analysis can also be extended to fixed ones.

Among both rules, the second one (neighbor-filtering) is the most challenging as it requires learning towards which directions relevant data are present. Since data can change over time, this process needs to be robust enough to track changes that can lead to considering new nodes and/or discarding existing ones. Thus, solutions to the problem of computing $\widehat{X}'$ need to tackle these challenges.

4. Network and Data Models

We consider a WSN composed of sensor nodes uniformly distributed over a square area, where communication range r and node density ρ are constant. In this sense, the more the sensor nodes in the network, the greater the area covered by the network. We assume the broadcast protocol model for wireless communication, where a node transmits information to all nodes in its communication range. In this sense, a node i can successfully transmit a packet to another node j if $d(i, j) < r$, where $d(\cdot)$ is the distance between these nodes. The communication range r is defined in order to assure a network connectivity near 99%. In this sense, a network with n nodes uniformly distributed (random network) can be considered as asymptotically connected with probability approaching one if each node is connected to more than $5.1774 \log(n)$ nearest neighbors [30].

Information sources (i.e., events) are modeled as uniformly distributed functions following a diffusion law with the distance; that is, $f(d) \propto 1/d^{\alpha}$, where α is the diffusion parameter (e.g., for light $\alpha \approx 2$, and for heat $\alpha \approx 1$) [17]. In this sense, most of the physical phenomena (e.g., light, heat, noise, radiation, etc.) follow this law. As a consequence, the reading

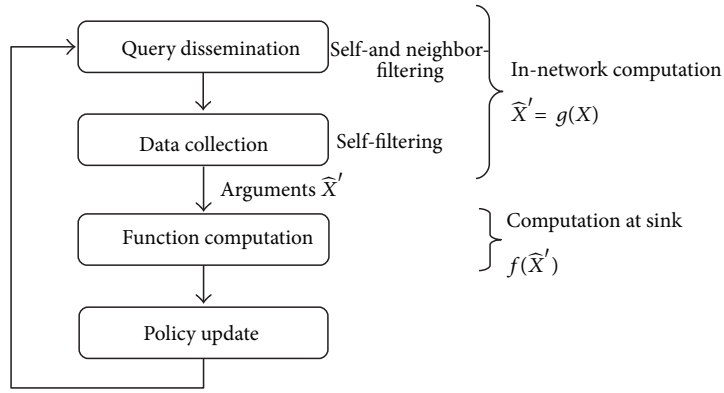

Figure 2: Function computation processes.

x_i obtained by the sensor node i is defined as the contribution of all events in the network, such as

$$x_i \propto \sum_{k=1}^{m} \frac{1}{d(i,k)^{\alpha}}, \tag{1}$$

where m is the number of events in the network, $d(\cdot)$ is the distance between sensor node i and event k, and α is the diffusion coefficient. In this work, we consider that all information sources emit the same amount of their physical magnitude. These sources could be objects such as light bulbs, air conditioners, heaters, and noise sources. For simplicity, we suppose that nodes are synchronized and sensors have infinite resolution.

In this context, a sink node, located at the center of the network, requires computing a specific function over time by injecting queries to the network. These queries are routed through the network to fetch relevant data $\widehat{X}'$. Thus, data flow in both directions: *downstream* (from sink to sensor nodes) and *upstream* (from sensor nodes to the sink). In the downstream direction, nodes can receive query messages, while, in the upstream direction, reply ones. In both directions, upon receiving any (query or reply) message, nodes decide both (i) if their readings are still worth to be reported to the sink and (ii) whether or not to broadcast the message to its neighbors.

In the *downstream direction*, the query message is first broadcasted by the sink node to all its neighbors who, after processing the query, can decide to forward it to their neighbors, and so on. Even if nodes can receive the same query multiple times, they can at most forward it once. The query message contains an identification (*msg-id*), a querier node identification (*querier-node-id*), a threshold value (*th*), and a hop counter (*hc*). Nodes keep track of the *ids* of already forwarded queries using local tables to avoid retransmissions. Initially, each node has no information of its distance in hops to the sink. This information is learned by each sensor node on each query iteration. In this sense, the distance to the sink (i.e., hop level) is calculated as the minimal hop count inside of received messages. Query messages are always processed even if received several times. Each time a node receives a query, the threshold value inside the message is read to determine if the node has relevant data (i.e., self-filtering). In *static filtering*, in which the query is routed through the whole network to select those nodes with readings above (below) a given (fixed) threshold value, each node can determine the relevance of its reading based on the first query message. Instead, in *dynamic filtering*, the threshold can be updated by nodes to implement maximum (minimum) function computation. Thus, even if a node may consider its data relevant on a first query arrival, it may learn on a successive query arrival that its data are not relevant at all as the threshold value has been updated. After processing the query, nodes must decide whether to forward or not the message to their neighbors (i.e., downstream neighbor-filtering). Even if a detailed discussion of the proposed schemes is presented in the next sections, we introduce its common principles: if a node believes that relevant data are present nearby, it will always relay the query message; otherwise, it will randomly decide whether to broadcast or not the query message. Since the function is computed over time, queries are periodically injected in the network by the sink node to calculate its value. Once the query process ends, only those nodes with relevant data (selected in downstream direction) report their readings ($\widehat{X}'$) to the sink after a short time interval Δt which is inversely proportional to the hop level of the selected node plus a small random time. In this sense, far selected nodes begin the response process first, filtering intermediate preselected nodes. As we will discuss later, each iteration feeds a learning process which helps to route next queries in a more efficient way. The query message conveys also a hop counter *hc*. This field is initialized to zero by the sink node and incremented by each receiving node to enable a *downstream learning* process. Since multiple messages from the sink may be received through different paths, nodes can discover their minimum distance (in hops) to the sink by this learning process.

In the *upstream direction*, selected nodes (i.e., not filtered) sent a reply message back to the sink containing their readings. This enhances the self-filtering process when

considering dynamic thresholds as preselected nodes in the downstream direction could be filtered in the upstream one. Even if each reply message is broadcasted, it is delivered to only one upstream node; thus, replies are routed back to the sink node through a single path. The broadcast mechanism helps to disable preselected nodes near the response path.

5. In-Network Filtering Schemes

Upon receiving either a query message or a reply one, nodes implement self-filtering in both directions. This is done comparing their readings with either thresholds in (downstream) query messages or arguments in (upstream) reply ones. Note that self-filtering does not introduce computation errors but can increase the communication cost due to the required message exchange to filter nodes with actually no relevant data. On the contrary, neighbor-filtering, which decides on whether it is worth broadcasting a query to neighbors, can introduce computation errors but tends to decrease the communication cost by avoiding querying nodes which may have useless data.

In this section, we describe two schemes for implementing the neighbor-filtering process. The first one considers that sensors nodes do not keep any state of previous filtering decisions; thus, it is referred to as *stateless filtering*. However, the sink node can learn from the information obtained from the network and modify the filtering rule over time by embedding some state information in the query messages that can be used to fetch only relevant data at a lower communication cost. Instead, in the second scheme, named *stateful filtering*, nodes maintain information about previous decisions while the sink node always keeps the same filtering rule. Both schemes can be described by (i) a downstream forwarding algorithm which implements the neighbor-filtering rule to route the query message through the network and (ii) an upstream learning process which actually adapts the dynamics of the neighbor-filtering rule in time.

5.1. Stateless Filtering. This scheme implements a version of the simulated annealing (SA) metaheuristic [20]. SA has the ability to explore beyond those areas where no relevant data are available due to its stochastic behavior. Since each query can be broadcasted to more than one node, this results in a natural parallelization of the algorithm as it can generate multiple forks of the same algorithm. The resulting algorithm is also known as *Parallel Adaptive Simulated Annealing (PASA)* [8]. Because PASA scheme is query-driven, it is conformed by two sequential phases, downstream forwarding and upstream learning. Both phases conform a computation iteration. The first one is used to forward the query to the sensor nodes following a one-to-many data flow model. The second one, instead, is implemented to send the arguments from selected nodes with relevant readings to the sink, following a one- or many-to-one data flow model, depending on the number of events in the network. These phases are detailed in the following subsections.

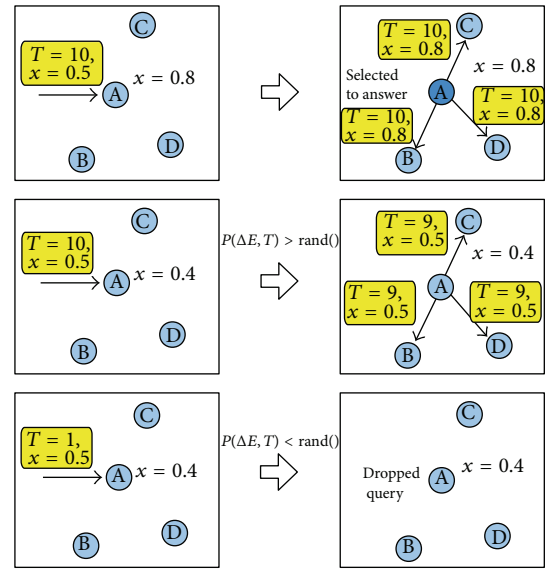

Figure 3: Downstream forwarding of the stateless scheme.

Downstream Forwarding. Each node can forward each query at most once in each computation iteration, in order to save as much energy as possible. The query message includes a T value, which represents the temperature of the algorithm, and the best found value (threshold) in the path followed by the query. Both parameters can be modified by each forwarding node. As a consequence, queries in the same iteration can have different T and threshold values depending on the sequence of broadcasts. A node having relevant data is configured to answer the query after a period of time and always broadcasts the query with the updated threshold to all its neighbors (Figure 3 top). Given that ΔE is the difference between the threshold value carried in the query message and the sensor reading, the condition $\Delta E \leq 0$ determines that relevant data are present, in case of maximum function computation. A node with no useful data forwards the query message to its neighbors with probability P computed as

$$P(\Delta E, T) = e^{-\Delta E/T}, \tag{2}$$

where T is the temperature parameter of the SA algorithm which is carried inside the query (Figure 3 center). Either the lower T value or the larger ΔE value, the lower probability P of forwarding the query to neighbors. In other words, the query is probably discarded by the forwarding node if the temperature of SA inside the query is low or nearby data are assumed irrelevant (Figure 3 bottom). A large T value encourages data exploration despite data relevance, thus increasing the probability P of forwarding queries. In our proposal, a large value of $T = T_0$ is initially used and then, in the query dissemination, linearly decreased by a D factor by intermediate nodes assuming irrelevant data, which is known as adaptive cooling. In the following iterations, a logarithmic decrement of T_0 is applied to accelerate the convergence of the algorithm. Instead, in the original SA algorithm, the cooling process is executed only at the beginning of each iteration. The value of D is computed by the sink node as $T_0/maxDist$,

```
(1)  Input Received query message
(2)  Output Response and forwarding decisions
(3)  id ← Msg.getId()
(4)  D ← Msg.getDec()
(5)  T ← Msg.getTemperature()
(6)  thresholdValue ← Msg.getThreshold()
(7)  ΔE ← thresholdValue − sensedValue
(8)  // Discard already forwarded queries
(9)  if idTable[id] == true then
(10)    delete(Msg)
(11) else
(12)    updateIdTable(id)
(13) end if
(14) // Query forwarding decision
(15) if ΔE ≤ 0 then
(16)    Msg.setValue(sensedValue)
(17)    Msg.setTemperature(T)
(18)    send(Msg)
(19) else
(20)    if P = e^{−ΔE/T} > rand() then
(21)       Msg.setTemperature(T − D)
(22)       send(Msg)
(23)    else
(24)       delete(Msg)
(25)    end if
(26) end if
```

Algorithm 1: Stateless Forwarding Scheme (PASA).

where *maxDist* is the maximum estimated distance from the sink in hops. This is due to the fact that there should exist a nonnull probability of exploring network borders even if irrelevant data are assumed; thus, it is required that $T > 0$ after traversing *maxDist* hops. Note that typically D will decrease with the network size. However, other cooling strategies could be considered. A detailed description of the forwarding algorithm of PASA is shown in Algorithm 1.

In the response process, single-path routing is implemented in order to reduce the communication cost, such as in [31]. Each selected node in PASA scheme uses the path of the first query arrival (lower latency path) to report arguments to the sink. Since PASA iteratively implements query and response phases, it can maintain an updated view of the network state. This allows dealing with nodes that can fail previous to an iteration. In case of a failure within an iteration, a simple *self-healing strategy* has been included to deal with this problem, which is discussed in detail in Section 6.4.

Upstream Learning. For a large value of T_0, downstream forwarding can behave as flooding as every node would always forward the query despite data relevance; thus, there is certainty in always finding most relevant data but at highest communication cost. At low T_0 values, it is equivalent to the gradient descent algorithm, where the query message is forwarded only through nodes with some relevant data. As a result, the probability of finding significant data is low, which can introduce computation errors. This feature is depicted in Figure 4 for the case of 10 events in the network. Different values for T_0 are considered and their impact is illustrated in terms of the computation error, the probability of success in computing the actual function value, and the query cost, which is defined by the average number of forwarded packets by each node in the network. From this analysis, we can conclude that there exists some T_0 value which can provide low or no computation error as well as low communication costs.

Therefore, it is necessary to find an optimal T_0 value to initialize T so that when linearly decreased it offers a good trade-off between communication costs and computation errors. For this purpose, a reinforcement learning (RL) algorithm is implemented in the sink node. The main idea behind this algorithm is that the sink can learn about the data distribution in the network based on its experience when receiving reply messages. It is implemented based on the *id* of response nodes. The sink node can apply a temperature update policy to adapt $T_0(i)$ value at each query iteration i to improve the search process, hence reducing energy consumption. The objective of the sink is to reduce the search depth iteratively by updating the $T_0(i)$ in order to reach only sensor nodes with relevant arguments.

This learning process is sketched in Figure 5. Even if nodes are scattered over an area A as illustrated, a first query iteration reaches nodes on area B. Nodes on (A-B) area were not queried since relevant data was not found nearby and the value of P became too small. On the second iteration, the initial $T_0(i)$ value is decreased to a $T_0(i+1)$ value, which reduces the search space to area C. This process is repeated till the algorithm finds out the value of $T_0(i)$, at some iteration i, that enables the query of all nodes with relevant data at the lowest cost, which in Figure 5 is represented by area E. At each query iteration, a decision on whether to decrease the previous $T_0(i-1)$ value or not is made by the sink. After sending the query for the first time with a high value of T_0, the sink records the number of node responses (i.e., the number of nodes which reported relevant data). As long as the sink gets data from the same number of responses, it is assumed that the previous $T_0(i-1)$ value can be reduced, so the sink decreases the injected $T_0(i)$ value in the next iteration i. A lower T_0 tends to narrow the search space, thus saving energy on nodes. If this number of responses decreases, the $T_0(i)$ value in iteration i is increased to an intermediate value between the $T_0(i-2)$, the last time without loss of information, and $T_0(i-1)$, the last iteration with loss of information. For example, $T_0(i-2) = 100$, $T_0(i-1) = 10$, and $T_0(i) = 45$ show this concept.

A description of logarithmic and linear cooling processes is sketched in Figure 6 in case of *max* function computation. The sink node is indicated as s node and a node selected to report its argument to the sink as n node. Query messages embed both the current T and threshold values. After i iterations, T_0 converges to a fixed $T_0(i)$ value which is used in successive queries, as shown in Figure 7. Note that if T_0 is decreased beyond a given value, the computation error increases as search space is significantly reduced and query forwarding becomes limited to areas with data gradients (i.e., $\Delta E \leq 0$). This last effect can be noticed by the number of wrong hops, which is defined as the average number of

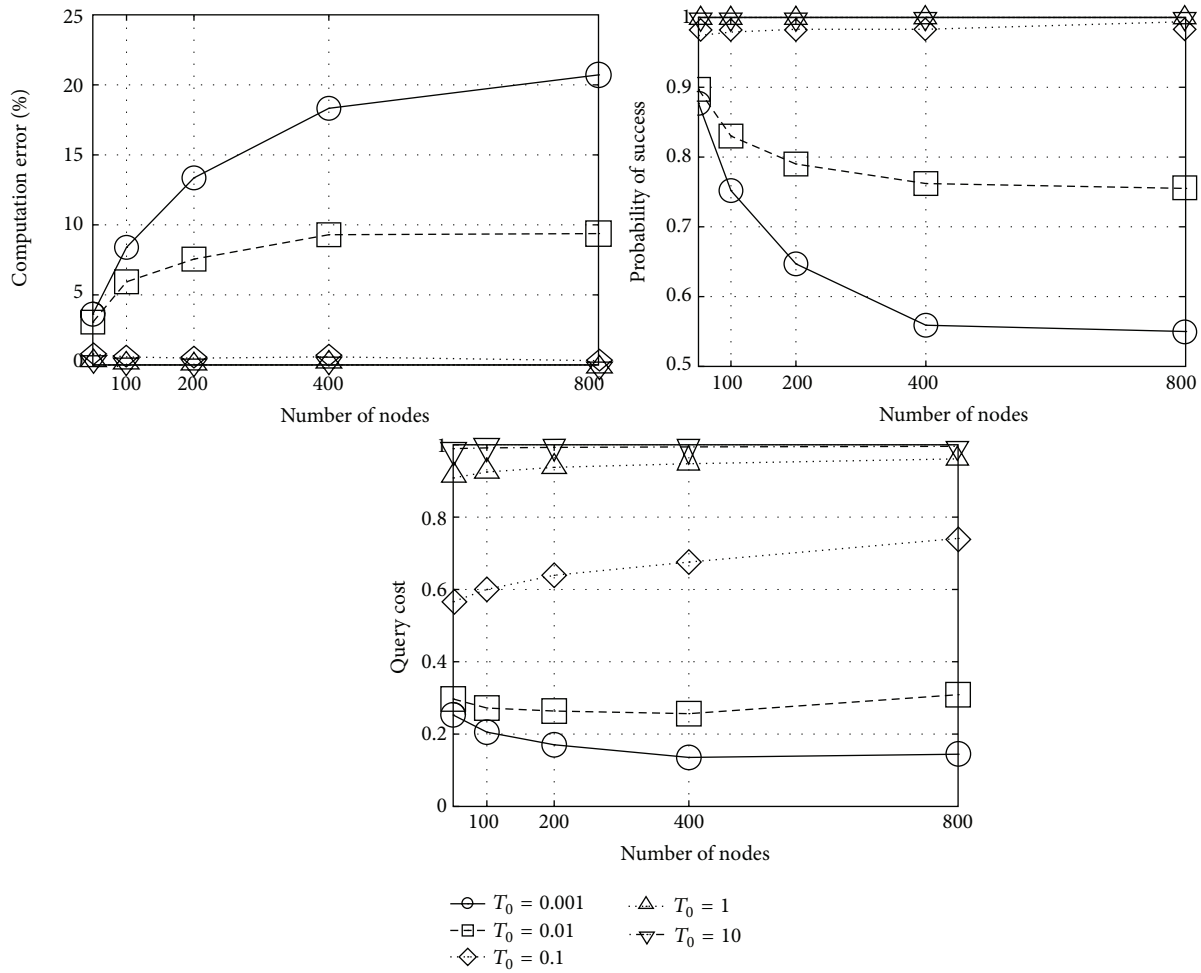

Figure 4: Performance of stateless scheme for different T_0 values.

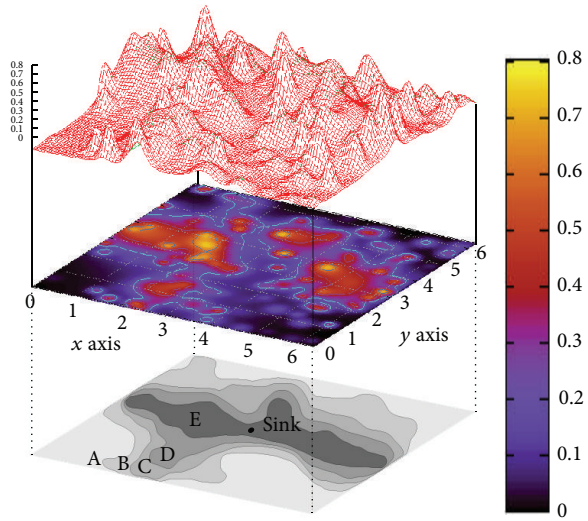

Figure 5: Iterative filtering with space reduction from A to E.

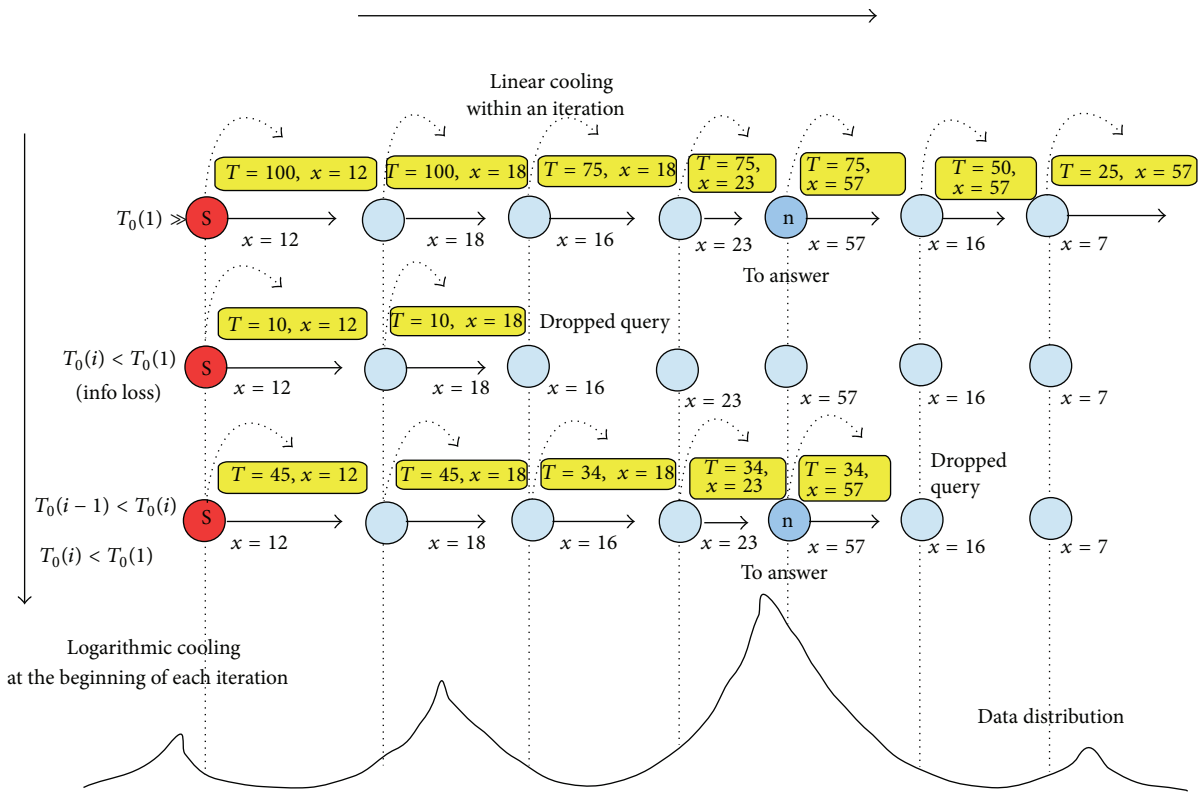

FIGURE 6: Cooling strategies used in stateless scheme.

queries forwarded by each node when data are considered not relevant ($\Delta E > 0$). Unlike traditional implementations of SA, we do not define a *stop condition* in PASA. Either a low T_0 value or a limit on the number of iterations is normally used as stop conditions in proposed SA-based solutions. In our scheme, it is possible under certain conditions that $T_0(i) \to 0$ for high i values, for example, in case of one event in the network. Since both network topology and data may change in time, this learning process can be run periodically to adapt the query dissemination to these variations.

In this work, we apply a more robust decision rule of the temperature update policy at the sink node with respect to our previous work [8]. In our previous version on PASA, we considered that the sink node takes advantage of the number of responses $\widehat{X}'$ obtained in the first iteration to apply the T adaptation policy. Instead, in this improved version, sink node applies a temperature update policy based on the *id* of those sensor nodes which report arguments in the first iteration. In this sense, sink decrements the temperature while those nodes report their readings and increments it otherwise, in order to avoid losing information.

5.2. Stateful Filtering. As for the stateless case, the goal of stateful scheme is to route queries towards nodes with relevant data to obtain the arguments for the computation of a given function. However, instead of centralizing learning on the sink node as just discussed, we consider a different scheme where sensor nodes can learn in a distributed fashion from previous decisions. The proposed scheme, also known as *Pheromone-based In-Network Processing (PhINP)* [9], implements an iterative procedure based on path reinforcement which results in search space reduction. Path reinforcement is achieved following a pheromone-based strategy similar to that used in ant colonies when searching for food [32], thus resulting in a probabilistic query routing towards nodes with relevant readings. PhINP, like PASA scheme, is a query-driven scheme conformed by two sequential phases, downstream forwarding and upstream learning, which are described in the following subsections.

Downstream Forwarding. Like in PASA, the query message is first broadcasted by the sink node to all its neighbors who, after processing the query, can decide to forward it to their neighbors, and so on. Even if nodes can receive the same query multiple times, they can at most forward it once. The message content was detailed in Section 4. In this case, each node maintains a state referred to as *pheromone level* λ whose value can range between 0 and 1. Nodes periodically decrement the value of λ by a factor λ_{dec} and can increase this value by a factor λ_{inc} under certain conditions but only after the upstream data collection process is over. The rate of the pheromone update policy is set up by the query process and in general it is assumed equal to the computation frequency (i.e., the frequency at which queries are disseminated). After each query iteration, some nodes tend to keep high pheromone levels, while others to decrease it. Queries are forwarded based on the pheromone level in a stochastic fashion such that

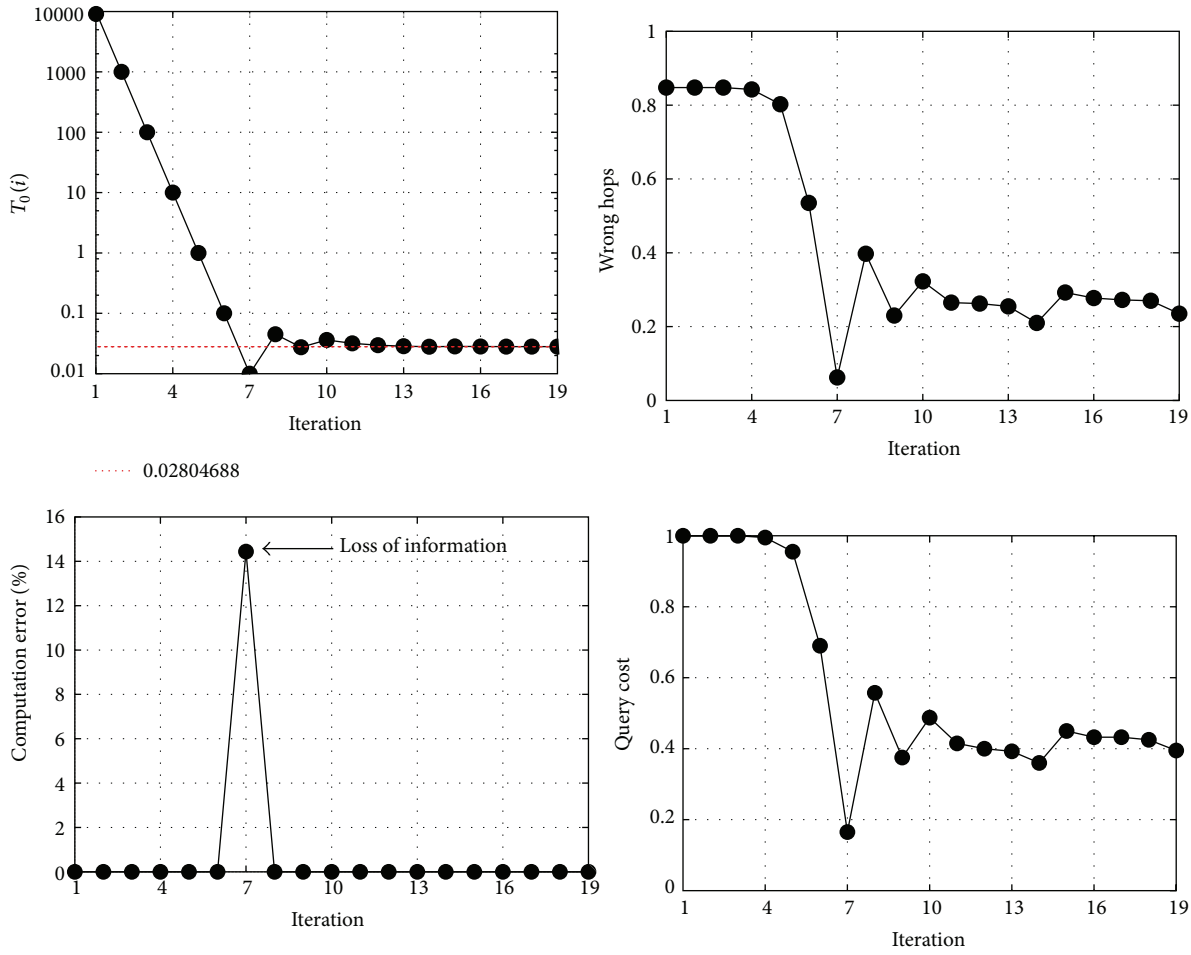

Figure 7: Stateless scheme convergence (400 nodes, 10 events).

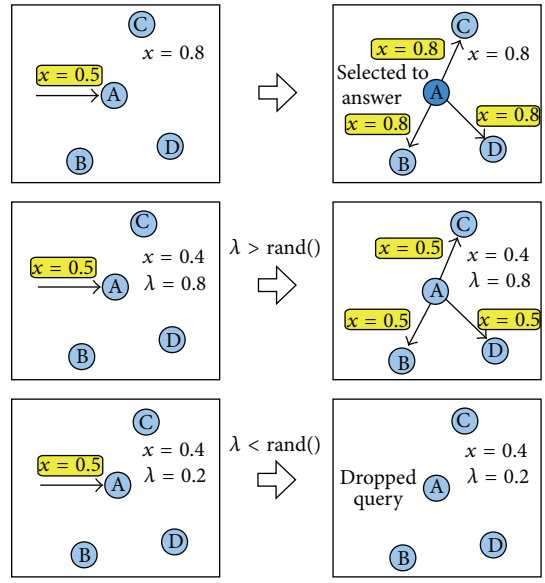

Figure 8: Downstream forwarding of the stateful scheme.

```
(1)  Input Received query message
(2)  Output Response and forwarding decisions
(3)  id ← Msg.getId()
(4)  thresholdValue ← Msg.getThreshold()
(5)  ΔE ← thresholdValue − sensedValue
(6)  // Discard already forwarded queries
(7)  if idTable[id] == true then
(8)      delete(Msg)
(9)  else
(10)     updateIdTable(id)
(11) end if
(12) // Query forwarding decision
(13) If ΔE ≤ 0 then
(14)     Msg ← setValue(sensedValue)
(15)     send(Msg)
(16) else
(17)     if λ > rand() then
(18)         Msg ← setValue(thresholdValue)
(19)         send(Msg)
(20)     end if
(21) else
(22)     delete(Msg)
(23) end if
```

Algorithm 2: Stateful Forwarding Scheme (PhINP).

```
(1) if λ > λmin then
(2)     λ = λ − λdec
(3) end if
(4) if answerQuery == true or forwardResp == true then
(5)     if λ < λmax then
(6)         λ = λ + λinc
(7)     end if
(8) end if
```

Algorithm 3: Pheromone update policy.

the more the pheromone is available, the more likely the message is forwarded. This process is described in Algorithm 2. The resulting node behaviors are shown in Figure 8, in which quite extreme conditions are considered (e.g., $\lambda = 0.2$ and $\lambda = 0.8$). The first query message sent by the sink encourages all nodes to participate in the forwarding phase by setting their pheromone level to its maximum value (i.e., $\lambda = 1$). This guarantees that all nodes receive the query and that all relevant data are properly selected during startup. Nodes can decrease their pheromone level up to a lower bound given by $\lambda_{\min}$ which determines the minimum exploration probability of searching for new data. Moreover, nodes can increase their pheromone level up to a maximal value of $\lambda_{\max}$ normally set to 1.

Upstream Learning. Only nodes which have been either selected to answer the query or have forwarded a response message to the sink node increase their pheromone level by λ_{inc} up to an upper bound given by $\lambda_{\max}$, as is described in Algorithm 3. In this way, paths from the sink node towards nodes with relevant data can be reinforced. On the following iterations, queries will be routed mostly over these paths, thus avoiding exploration of those areas where irrelevant data are assumed. To simplify the notation, we define a configuration vector $[\lambda_{\max}\ \ \lambda_{\text{inc}}\ \ \lambda_{\text{dec}}\ \ \lambda_{\min}]$ which is set in each sensor node and whose values are a function of the nature of the physical phenomena to be monitored. In this work, a reference configuration vector [1 0.2 0.1 0.1] is in general assumed.

In this work, we improve further the downstream forwarding decision with respect to our previous versions [9, 33, 34] to increase its robustness to changes in the sensed field. This is implemented by always forwarding queries if relevant data are present (i.e., $\Delta E \leq 0$), despite the pheromone level, which enhances the detection of new events.

Table 1: Simulation parameter setting.

Parameter	Value
Deployment area	200×200, 283×283, 400×400 m
Network size	100, 200, 400 nodes
Node density	2.5×10^{-3} nodes/sq m
Communication range	Circular, 50 m
Average neighbors per node	19.635
Node failure probability	0, 1, 5, 10%
Number of data sources	1 to 400
Movement of data sources	20 to 200% of comm. range
Number of simulations	2000
Random generator	Mersenne Twister

6. Simulation Results

The proposed schemes were evaluated in simulation environments developed in Omnet++ [35], where three metrics were analyzed to assess their performance under different scenarios: computation error, probability of success, and communication cost. Computation error represents the relative error resulting from the computed function at the sink and the actual optimal one. On the other hand, the probability of success represents the probability of computing the optimal value (i.e., zero relative error). In this sense, in case of computing the *max* function, this metric represents the probability of finding the node with the maximal reading in the network. Communication cost considers average number of nodes involved in forwarding both the query message (query cost) and the relevant readings to the sink (response cost). Note that the maximum query cost is equal to 1, which means that all nodes forwarded the query once (i.e., flooding). As each selected node sends its reading to the sink by a single path, this response cost is negligible with respect to the query cost. For this reason, we focus on the query communication cost only. Under the assumption that each sensor node can forward once the query in each iteration, the query cost can be also defined as the number of packets transmitted at each iteration. Note that this metric is closely related to the consumed energy in the network. The parameter set used in the simulations is shown in Table 1. We evaluate several aspects of the proposed schemes, in order to compare their performance and to determine the best

application fields of each scheme. Our analysis includes the algorithm convergence, event analysis, the robustness to node failure, packet loss and dynamic events, and the capability of readaptation to event changes in the sensor network. Finally, a comparison between the proposed schemes is addressed. These aspects are described in detail in the following sections.

6.1. Convergence. Since the stateless and stateful schemes learn in time, it is expected that their metrics will experience some variations during first iterations. In case of the stateless scheme, it needs to converge to a T_0 value, while for stateful one, the pheromone level needs to get stabilized on nodes. To simplify the analysis, we consider a network where both nodes and links are ideal; thus, nodes cannot fail and links are loss free. Besides, we assume that the sensed field, formed by 10 random data sources or events with the same amplitude following the diffusion law of heat ($\alpha = 1$), does not change neither their amplitudes nor their positions while the scheme converges.

In the stateless scheme, the sink node sets a high T_0 value to the query for the first iteration ($T_0(1) = 10000$ is in general considered in this work), in order to reach each sensor node in the network. Based on the set nodes which report data in the first iteration, the sink node can estimate if there is any information loss in the following iteration and adjust the $T_0(i)$ value in query to avoid it. As discussed in Section 5.1, this is based on reinforcement learning. The objective is to reduce the query dissemination cost, while there is no information loss. Due to the probabilistic nature of the proposed stateless scheme, it can incur in very low computation errors once the algorithm has converged, near 3% in case of this scenario. The computation error and query cost metrics obtained by simulation in this scenario are shown in Figures 9(a) and 9(b), respectively.

On the other hand, since the stateful scheme also floods the network in the first iteration, the computation error remains null in the following iterations, considering no major data changes in the sensed field. Under these conditions, the convergence of the algorithm can be analyzed as it tries to decrease the query cost while keeping the error null (i.e., retrieving the same relevant data) as shown in Figures 9(c) and 9(d). Indeed, the first iteration has always a cost equal to 1 as every node forwards the query once. After each iteration, paths to nodes with relevant data are reinforced based on local pheromone levels, and the probability P of using other paths is reduced, minimizing the communication cost. Each node's behaviour is described by 4 fixed parameters (λ_{max}, λ_{min}, λ_{inc}, and λ_{dec}) and a variable one λ, all of them ranging from 0 to 1. The collaborative work of the whole network defines in a decentralized way the performance of this scheme. In this analysis, different values of λ_{dec}, λ_{inc}, and λ_{min} are considered in order to understand the operation of this scheme. The larger these values, the faster the convergence to a minimum cost configuration; however, this cost tends to be higher for larger λ_{dec} and λ_{inc} values. A trade-off does exist between convergence time and this minimum cost. The value of λ_{max} is always set to 1 in order to avoid errors in the computation. The behavior of this scheme for several configuration values and network sizes is shown in Figures 9(c) and 9(d). Note that the performance of this distributed algorithm is independent of the network size.

From simulations we can see that the query cost for the stateless scheme can be decreased to more than 50% with respect to flooding while still keeping the error bounded to less than 2.5% with the considered scenario. Note that the stateful scheme can improve the communication cost with respect to stateless one by using lower λ_{min} values, at expense of lower robustness to dynamic events, as a consequence of the lower probability to escape from local minima, as we will discuss in following subsections. If $\lambda_{min} \to 0$, then the communication query cost and the response cost tend to have the same value, since the same path is used to forward both queries and responses.

Moreover, the stateless scheme tends to introduce more errors than the stateful one. However, the latter requires good synchronization of the setting parameters due to its distributed nature. In the stateless scheme, this condition is relaxed since the sink node configures and sends the cooling policy into the query.

6.2. Event Analysis. In this case, we evaluate the performance of the stateless and stateful schemes for different fields, that is, from the point of view of data distribution in the network. In this sense, we define the *event-sensor-rate* metric (esr), which is the relationship between data sources and sensor nodes presented in the network. In this sense, if we have n sensor nodes and e events (i.e., sources of information), the esr factor can be defined as $esr = e/n$. Normally, events and nodes are not at the same position, thus, we assume an independent random distribution for each of them. Moreover, to simplify the analysis, we consider that all events have the same amplitude and diffusion coefficient. Simulations with esr values ranging from 0.01 to 1 and configurations of $T_0(1) = 10000$ for stateless and [1 0.2 0.1 0.1] for stateful scheme are used for this purpose. Results obtained by simulation are shown in Figure 10. We can see that the stateless scheme has a very low computation error, which is normally the lower possible when $esr \to 0$, that is, the case of one or few events. On the other side, an $esr \approx 0.1$ introduces the higher error, in case of stateless scheme, with the higher communication cost in both schemes. As a consequence, we defined a scenario with 10 events as the case study in this work. Also, we can see that communication cost is rapidly reduced when $esr \to 1$ due to the filtering process in the network.

6.3. One-Event Detection. In this case, the capacity to detect an event randomly located in the network is analyzed. Note that this analysis is a particular case ($esr \to 0$) of the previous one. However, the goal is to compare the performance of the stateless and stateful schemes with respect to two traditional mechanisms such as *flooding* and *gossip*. The gossip mechanism is set with a *gossip probability = 0.25* in order to have the same query cost as the proposed schemes. The performance of these schemes is shown in Figure 11. We can see that flooding can always find those nodes with relevant arguments to compute the function, but it incurs in

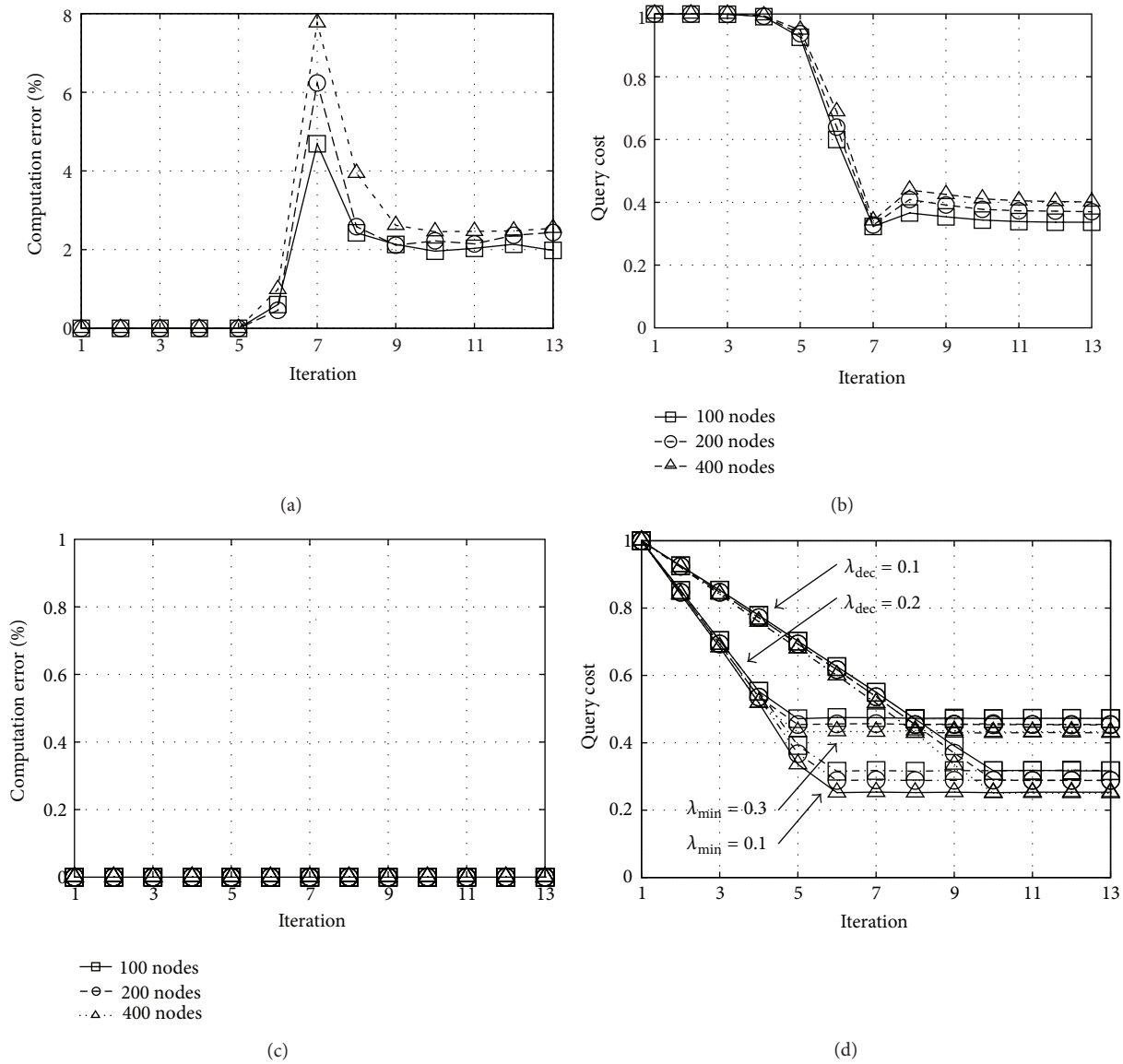

Figure 9: Convergence: stateless ((a), (b)) and stateful ((c), (d)) schemes.

a high communication cost, because each node has to forward the query (i.e., query cost = 1). Note that the performance of stateless and stateful schemes falls between the performance obtained by flooding and gossip mechanisms. On one hand, the proposed schemes incur in lower errors than gossip and, on the other hand, can reduce the communication cost greatly, in this case a reduction of 80%.

6.4. Robustness to Node Failure. Once the stateless and stateful schemes have converged after few iterations, it becomes critical for them to still work under faulty conditions. According to this, we consider the case of nodes with some failure probability. Figure 12 shows the performance of both schemes for different probabilities of node failure as the network size increases. To simplify the analysis, we suppose that the sensed field is static, thus, the sensed values do not change over time.

As expected, the stateful scheme, although has lower computation error in the network size range analyzed, is more susceptible to failure of nodes than the stateless scheme. The reason behind this is that the stateful one tends to maintain a single path (i.e., pheromone trail) for the query dissemination between sink and each of the nodes that provide relevant arguments to compute the function at the sink. In this sense, the failure of a forwarding node is critical, since this can affect the computation of a function. As we will see, this behavior of the stateful scheme to distributively form paths can be relaxed, as the width of paths is a function of the $\lambda_{\min}$ value. With the term *width*, we refer to the multipath characteristic of query dissemination. With a low value of $\lambda_{\min}$, that is, $\lambda_{\min} \rightarrow 0$, a unique single path is formed between sink and a selected sensor node. As $\lambda_{\min}$ increases, each disseminated query arrives to a node through more incoming paths (i.e., multiple paths). This effect is depicted in Figure 13 for a better

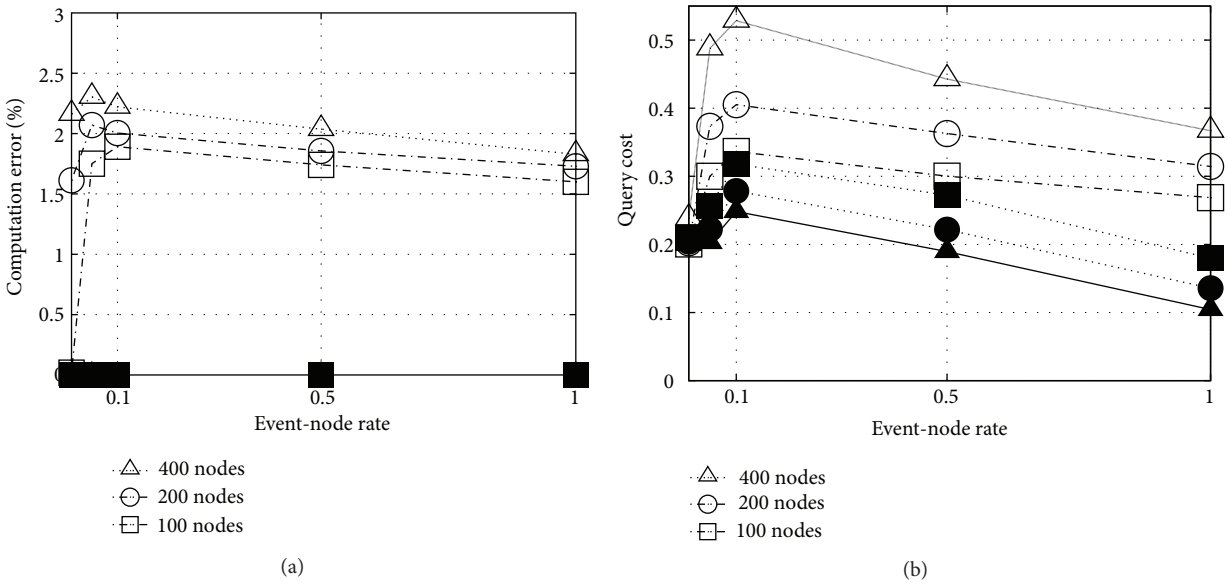

Figure 10: Event detection. Stateless (white) and stateful (black).

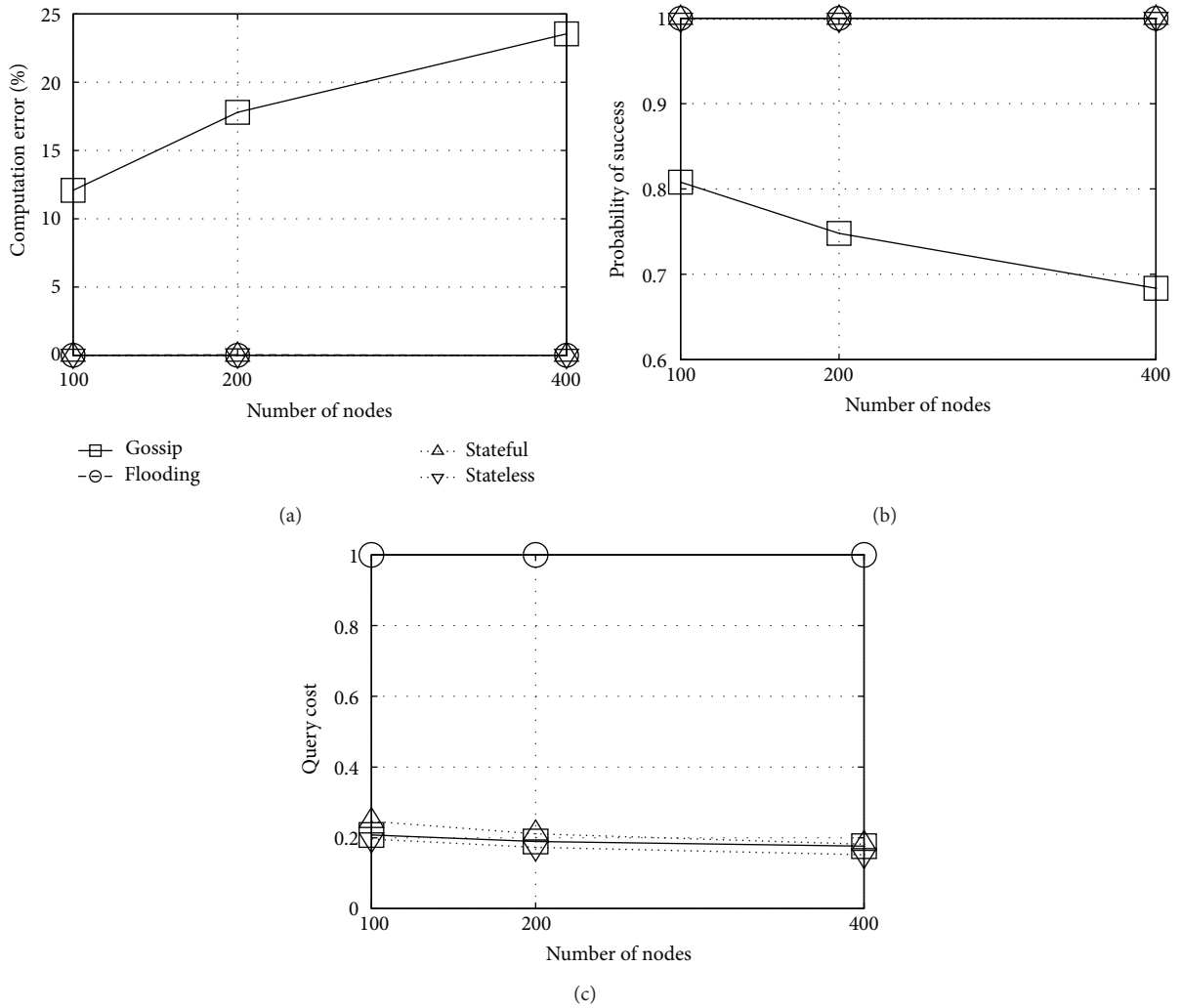

Figure 11: Capacity to detect a randomly deployed event.

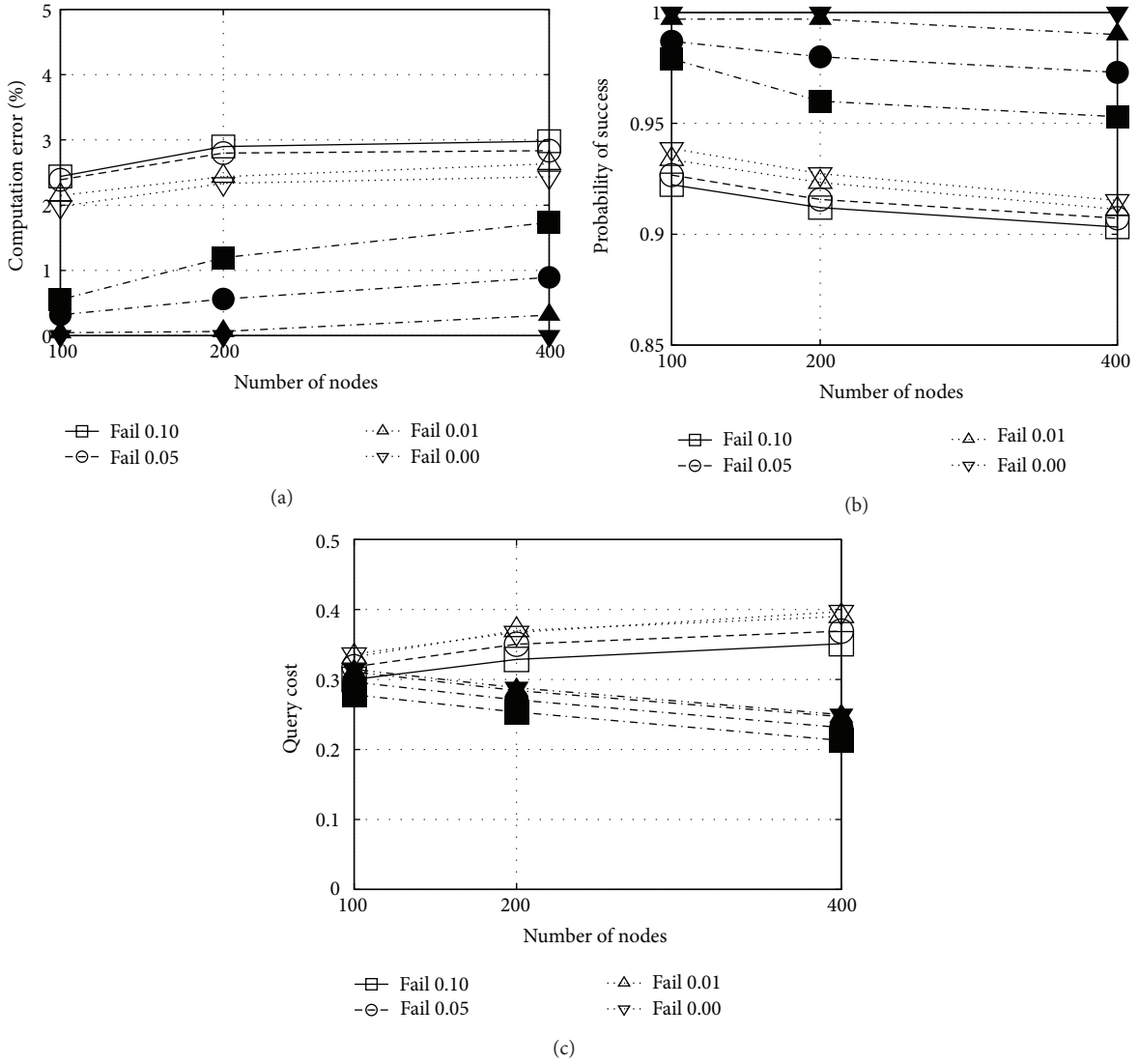

Figure 12: Lossy network. Stateless (white) and stateful (black).

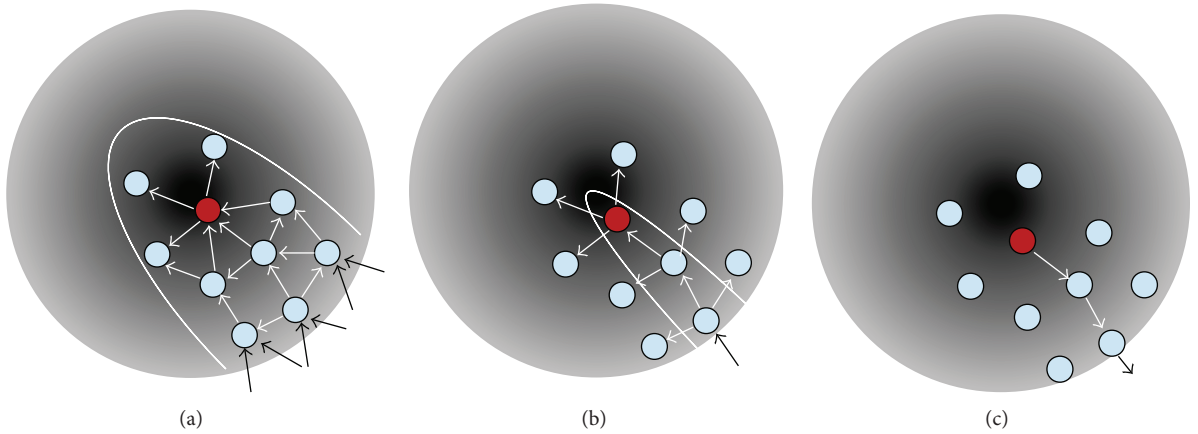

Figure 13: Downstream forwarding: stateless (a) and stateful (b) schemes and upstream forwarding (c) after convergence.

understanding. In case of the stateless scheme, the query arrives through multiple incoming paths. So, a failure in an forwarding node is not critical (see Figure 13(a)). Instead, in case of stateful scheme, if a low $\lambda_{\min}$ value is used, the scheme after convergence forms narrow paths which are more affected by failures in nodes as depicted in Figure 13(b). For higher $\lambda_{\min}$ values, the stateful scheme tends to have similar behavior from the point of view of incoming query messages as the stateless one, at the expense of a higher communication cost. A more detailed description is given in Section 6.8. As explained above, both schemes report the arguments of selected nodes by using the inverse path of the first arrival path as shown in Figure 13(c).

In order to increase the robustness of the proposed schemes to node failures, a simple *self-healing mechanism* is implemented to avoid loss of information. In this sense, each node sending information in upstream forwarding, based on reinforcement learning by active hearing, can notice if the transmitted packet was forwarded by its one-hop neighbor towards the sink. The same mechanism is followed by intermediate forwarding nodes in the path. When a sending node realizes that its one-hop destination neighbor does not forward the response packet (see Figure 14(a)), it sends the same response to the following neighbor of its neighbor's vector as depicted in Figure 14(b). In this respect, as seen above, the neighbor's vector is constructed by each sensor node in each query iteration. This is an ordered array in which the *ids* of one-hop neighbor nodes sending the query are stored. Note that the stateless scheme can also benefit from this improvement, not so for the case of stateful schemes set with low $\lambda_{\min}$ values.

6.5. Robustness to Loss of Packets. In this case, the robustness of the stateless and stateful schemes to packet loss is analyzed. The same scenario and parameters setting as in the case of node failure analysis are considered. Results are shown in Figure 15. As expected, the stateful scheme is more sensible to packet loss than the stateless one, since it is able to conform paths using fewer query forwarding nodes.

6.6. Robustness to Dynamic Events. Even if the proposed schemes have shown to be robust enough to lossy networks, it becomes also crucial to analyze their behavior as the sensed field changes in time. For this purpose, we consider now a loss free network where all field sources (events) may change their positions. We are interested in evaluating if, after the schemes have converged to a given sensed field, they can adapt to a different one which has still some correlation with the original one. In this sense, a random position change is inserted to each event in the network, which is defined as a percentage of the communication range. The simulation considers the case of 10 events in the network, and the statistics are reported just after the change, that is, in the following iteration without considering readaptation as will be analyzed above for the stateful scheme. Results obtained for several position change values are shown in Figure 16. More clearly, if the communication range r of sensor nodes is fixed to 50 meters, a position change of 100% implies that each event in the network is randomly moved at most the same value of the communication range from its initial position. In this sense, a change of 20% has the effect that the node with the best reading to report to the sink (i.e., near to the event) could be the same; instead with a change of 200% the node with the best reading is usually another. Based on the simulation results, we can see that both schemes have similar tendencies as the percentage of position change of events is increased.

6.7. Readaptation Capability. As an extension of the previous analysis, in which we only report the metrics after applying the changes, we analyze the readaptation capability of the stateful scheme in several iterations after those changes. Recall that the stateless scheme does not have this readaptation feature due to the learning process followed by the centralized control of sink. As seen above, the configuration parameters λ_{inc} and λ_{dec} define the dynamic of this scheme. In this case, a scenario with 50 randomly deployed events was considered. The scheme was configured as $[1 \;\; X \;\; 0.2 \;\; 0.1]$ and the results obtained through simulations are shown in Figure 17. Intuitively, a user would try to set the λ_{inc} value as high as possible, but this depends on the application.

6.8. Schemes Comparison. Based on the discussed results, we can notice that each scheme has its own scope; that is, each scheme is more efficient for a given scenario. We make this analysis considering that both schemes have already converged to a low-energy state. In this sense, the stateful scheme is more appropriate than the stateless one in scenarios with dynamic events. However, the stateless scheme is more robust to failure of sensor nodes due to the fact that the same query message more probably arrives to a node through multiple paths. In this sense, in the stateful scheme there is a trade-off between the communication cost and the robustness to node failure. If a low $\lambda_{\min}$ value is set, the scheme tends to a low-energy state, with a minimal communication cost, at the expense of a low robustness to node failure, and vice versa. Figure 18 shows the queried areas for both schemes after the convergence in case of max function computation. In this case, a low $\lambda_{\min}$ value was set in the stateful scheme in order to obtain well-defined paths between sink (red node) and selected nodes (white nodes) to provide arguments to compute the function. In case of setting a large $\lambda_{\min}$ value, these paths are widened, and, as consequence, the queried area tends to resemble that of the stateless scheme.

A substantial difference between both schemes is the probability of exploration of sensor nodes (i.e., query forwarding) related to the neighbor-filtering rule. In the stateful scheme this probability is limited to a $\lambda_{\min}$ value which is independent of the position of the node. Instead, in the stateless scheme, this probability is a function of the distance to the sink, so the farther the node, the lower the probability of forwarding the query (i.e., exploration). This is due to the degradation of the temperature T of a query when it is forwarded through the network using a centralized control. In this sense, the temperature $T_0(i)$ inside the query, which is set by the sink node when the query is injected to

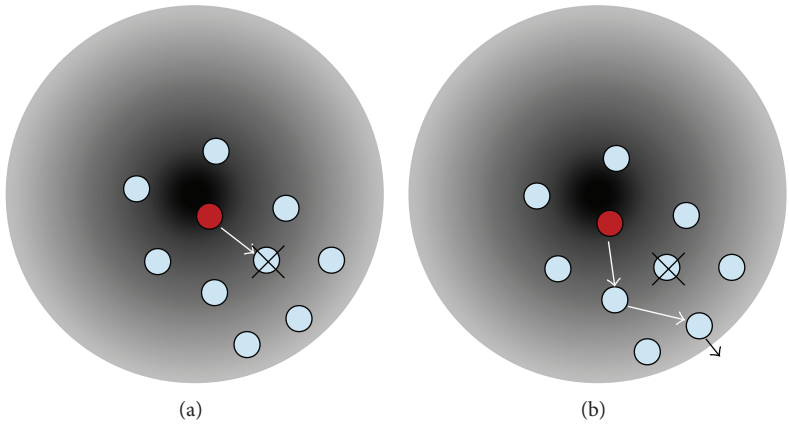

Figure 14: Self-healing mechanism to deal with node failures.

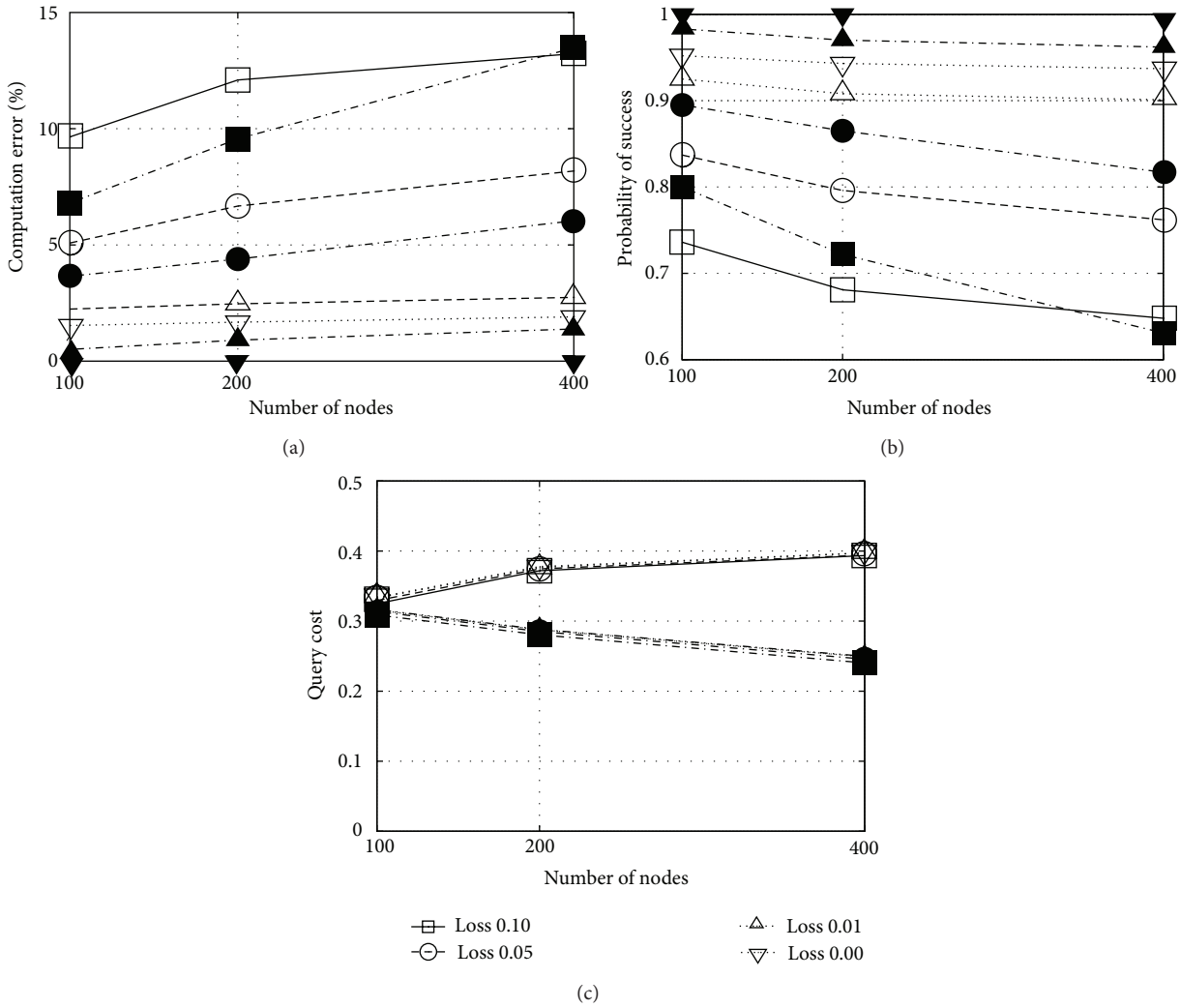

Figure 15: Loss of packets. Stateless (white) and stateful (black).

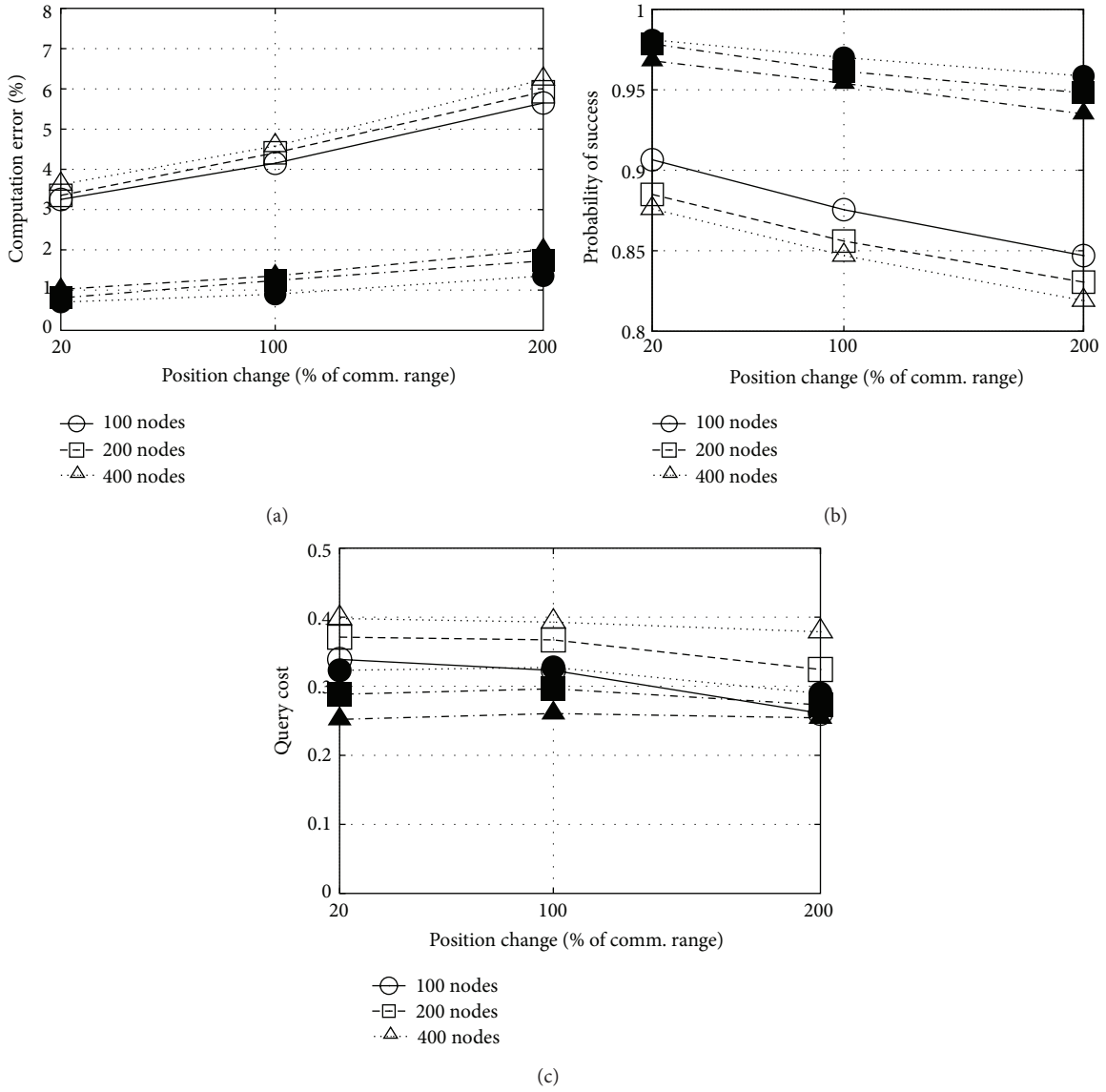

Figure 16: Dynamic events. Stateless (white) and stateful (black).

the network, is the highest that it can have during iteration i. This is a consequence of a centralized control. This feature makes more reliable stateful scheme for large-scale WSNs. A summary of the main features and setting parameters of the proposed schemes is shown in Table 2.

7. Conclusion

In this work, we analyzed the problem of in-network filtering to compute efficiently type-threshold functions in a WSN, minimizing both communication cost and computation error. In this context, in-network filtering schemes can be used to forward only relevant data towards a sink node for processing purposes as an alternative to data aggregation. To this end, two nature-inspired schemes were proposed that can drive this filtering process. The first one considered a stateless filtering scheme where the sink node implemented a learning algorithm to feed a decision rule based on the well-known simulated annealing search process. Instead, the second approach, dubbed stateful scheme, considered a distributed learning scheme inspired by the ant colony

Table 2: Comparison of proposed schemes.

Feature	Scheme	
	Stateless (PASA)	Stateful (PhINP)
Type of control	Centralized	Distributed
Control parameter	T	λ
Parameter location	Inside the query	Local at each node
Flooding behavior	$T \gg$	$\lambda_{\min} = 1$ or $\lambda_{\text{dec}} = 0$
Convergence speed	Logarithmic, linear D	$\lambda_{\text{inc}}, \lambda_{\text{dec}}, \lambda_{\min}$
Readaptation capability	No	Yes
Readaptation speed	—	$\lambda_{\text{inc}}/\lambda_{\text{dec}}$

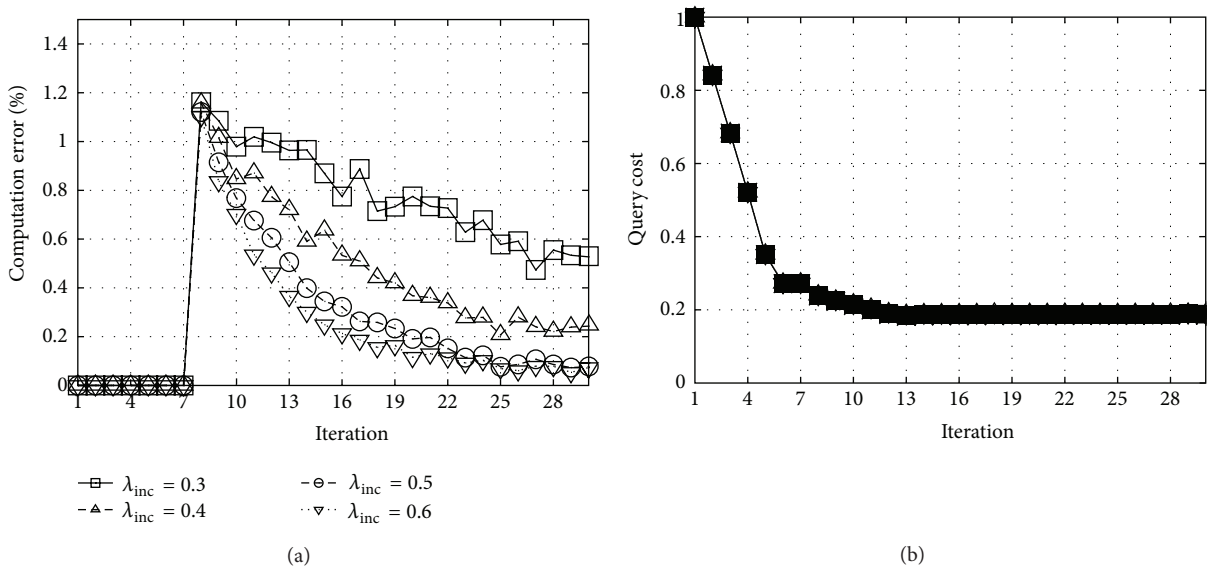

Figure 17: Readaptation capability. Stateful scheme.

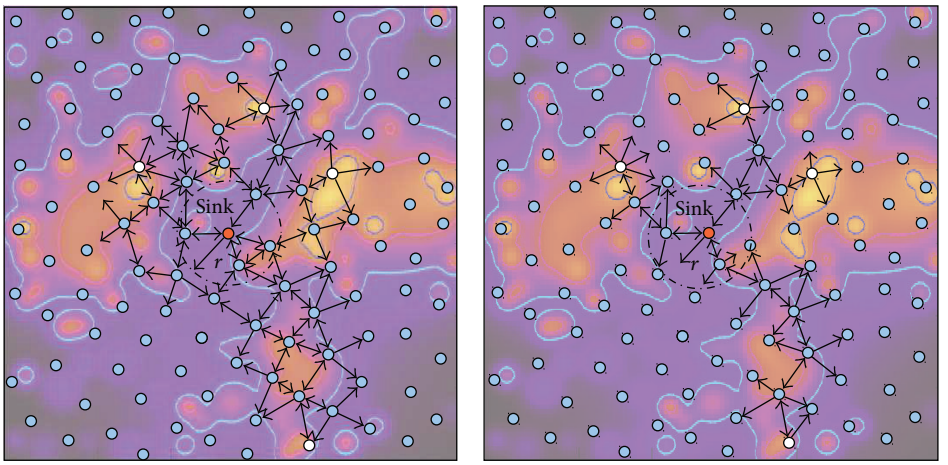

Figure 18: Queried area. Stateful (a) and stateful (b).

behavior where nodes kept a state to reinforce paths to the sink. We show by simulation that communication costs can be significantly reduced with respect to traditional schemes while keeping the computation error bounded.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is partially funded by Universidad Tecnológica Nacional, FONCyT IP-PRH Postgraduate Grant Program, SECYT-UNC Research Program, and CONICET Postgraduate Grant.

References

[1] S. Tilak, N. Abu-Ghazaleh, and W. Heinzelman, "A taxonomy of wireless micro-sensor network models," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, no. 2, pp. 28–36, 2002.

[2] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, 2008.

[3] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks," *Ad Hoc Networks*, vol. 3, no. 3, pp. 325–349, 2005.

[4] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: a survey," *Ad Hoc Networks*, vol. 7, no. 3, pp. 537–568, 2009.

[5] R. Rajagopalan and P. Varshney, "Data-aggregation techniques in sensor networks: a survey," *IEEE Communications Surveys & Tutorials*, vol. 8, no. 4, pp. 48–63, 2006.

[6] A. Giridhar and P. R. Kumar, "Computing and communicating functions over sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 4, pp. 755–764, 2005.

[7] A. Giridhar and P. R. Kumar, "Toward a theory of in-network computation in wireless sensor networks," *IEEE Communications Magazine*, vol. 44, no. 4, pp. 98–107, 2006.

[8] G. Riva and J. Finochietto, "A parallel and adaptative query routing scheme for wireless sensor networks," in *Proceedings of the IEEE International Conference on Communications (ICC '12)*, pp. 243–247, Ottawa, Canada, June 2012.

[9] G. G. Riva and J. M. Finochietto, "Pheromone-based in-network processing for wireless sensor network monitoring systems," in *Proceedings of the IEEE International Conference on Communications (ICC '12)*, pp. 6560–6564, Ottawa, Canada, June 2012.

[10] T. C. Aysal, M. E. Yildiz, A. D. Sarwate, and A. Scaglione, "Broadcast gossip algorithms for consensus," *IEEE Transactions on Signal Processing*, vol. 57, no. 7, pp. 2748–2761, 2009.

[11] M. Franceschelli, A. Giua, and C. Seatzu, "Distributed averaging in sensor networks based on broadcast gossip algorithms," *IEEE Sensors Journal*, vol. 11, no. 3, pp. 808–817, 2011.

[12] A. Meliou, C. Guestrin, and J. M. Hellerstein, "Approximating sensor network queries using in-network summaries," in *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN '09)*, pp. 229–240, San Francisco, Calif, USA, April 2009.

[13] Y. Xu, T. Fu, W. Lee, and J. Winter, "Processing K nearest neighbor queries in location-aware sensor networks," *Signal Processing*, vol. 87, no. 12, pp. 2861–2881, 2007.

[14] Y. Xu, W. Lee, J. Xu, and G. Mitchell, "Processing window queries in wireless sensor networks," in *Proceedings of the 22nd International Conference on Data Engineering (ICDE '06)*, pp. 270–280, Atlanta, Ga, USA, April 2006.

[15] H. Huang, J. Hartman, and T. Hurst, "Efficient and robust query processing for mobile wireless sensor networks," *International Journal of Sensor Networks*, vol. 2, no. 1-2, pp. 99–107, 2006.

[16] J. Ahn, S. Kapadia, S. Pattem, A. Sridharan, M. Zuniga, and J. Jun, "Empirical evaluation of querying mechanisms for unstructured wireless sensor networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 17–26, 2008.

[17] J. Faruque and A. Helmy, "RUGGED: Routing on fingerprint gradients in sensor networks," in *Proceedings of the IEEE International Conference on Pervasive Services (ICPS '04)*, pp. 179–188, Novi Sad, Servia and Montenegro, July 2004.

[18] J. Zhang, X. Zhu, and H. Peng, "Bi-filtered forwarding: a quasi-optimal routing algorithm for query delivery in wireless sensor networks," *International Journal on Smart Sensing and Intelligent Systems*, vol. 6, no. 3, pp. 993–1011, 2013.

[19] J. Heidemann, F. Silva, and D. Estrin, "Matching data dissemination algorithms to application requirements," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys '03)*, pp. 218–229, November 2003.

[20] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[21] A. Zimmerman and J. Lynch, "A parallel simulated annealing architecture for model updating in wireless sensor networks," *IEEE Sensors Journal*, vol. 9, no. 11, pp. 1503–1510, 2009.

[22] R. V. Kulkarni, A. Förster, and G. K. Venayagamoorthy, "Computational intelligence in wireless sensor networks: a survey," *IEEE Communications Surveys and Tutorials*, vol. 13, no. 1, pp. 68–96, 2011.

[23] K. A. Yau, P. Komisarczuk, and P. D. Teal, "Reinforcement learning for context awareness and intelligence in wireless networks: review, new features and open issues," *Journal of Network and Computer Applications*, vol. 35, no. 1, pp. 253–267, 2012.

[24] F. Dressler and O. B. Akan, "A survey on bio-inspired networking," *Computer Networks*, vol. 54, no. 6, pp. 881–900, 2010.

[25] M. Saleem, G. A. Di Caro, and M. Farooq, "Swarm intelligence based routing protocol for wireless sensor networks: survey and future directions," *Information Sciences*, vol. 181, no. 20, pp. 4597–4624, 2011.

[26] A. M. Zungeru, L. Ang, and K. P. Seng, "Classical and swarm intelligence based routing protocols for wireless sensor networks: a survey and comparison," *Journal of Network and Computer Applications*, vol. 35, no. 5, pp. 1508–1536, 2012.

[27] B. Park, S. Park, E. Lee, S. Noh, and S. Kim, "Large-scale phenomena monitoring scheme in wireless sensor networks," in *Proceeding of the 71st IEEE Vehicular Technology Conference (VTC '10)*, pp. 1–5, Taipei, Taiwan, May 2010.

[28] M. Li and Y. Liu, "Iso-Map: energy-efficient contour mapping in wireless sensor networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 5, pp. 699–710, 2010.

[29] J. Sun, "Multi-threshold based data gathering algorithms for wireless sensor networks," *Journal of Networks*, vol. 4, no. 1, pp. 30–41, 2009.

[30] F. Xue and P. R. Kumar, "The number of neighbors needed for connectivity of wireless networks," *Wireless Networks*, vol. 10, no. 2, pp. 169–181, 2004.

[31] Z. Cheng and W. B. Heinzelman, "Flooding strategy for target discovery in wireless networks," *Wireless Networks*, vol. 11, no. 5, pp. 607–618, 2005.

[32] L. F. M. Vieira, U. Lee, and M. Gerla, "Phero-trail: A bio-inspired location service for mobile underwater sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 4, pp. 553–563, 2010.

[33] G. Riva and J. Finochietto, "Pheromone-based in-network processing for wireless sensor network monitoring systems," *Journal of Network Protocols and Algorithms*, vol. 4, no. 4, pp. 156–173, 2012.

[34] G. G. Riva, J. M. Finochietto, and G. Leguizamon, "Bio-inspired in-network filtering for wireless sensor monitoring systems," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '13)*, pp. 3379–3386, Cancun, Mexico, June 2013.

[35] Omnet ++ simulation library, http://www.omnetpp.org/.