A Probabilistic Query Routing Scheme for Wireless Sensor Networks

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

[1] Universidad Tecnológica Nacional, Córdoba
[2] Universidad Nacional de Córdoba - CONICET, Córdoba
griva@scdt.frc.utn.edu.ar,jfinochietto@efn.uncor.edu

Abstract. The use of wireless sensor networks for information discovery and monitoring of continuous physical fields has emerged as a novel and efficient solution. To this end, a query message is routed through the network to fetch data from sensor nodes and report it back to a sink node. As several applications only require a limited subset of the available data in the network, this query could be ideally routed to fetch only relevant data. In this way, much energy due to message exchange among nodes could be saved. In this paper, we consider the application of computational intelligence on nodes to implement a parallel adaptive simulated annealing (PASA) mechanism able to direct queries to relevant nodes. Besides, a reinforcement learning algorithm is proposed to adapt progressively the query process to the characteristics of the network, limiting the routing space to areas with useful data. Finally, the relevant data collection mechanism is also discussed to illustrate the complete process. We show by extensive simulations that the routing cost can be reduced by approximately 60% over flooding with an error less than 5%.

Keywords: Sensor Networks, Information Discovery, Energy Conservation, Computational Intelligence, Metaheuristics, Reinforcement Learning

1 Introduction

Wireless sensor networks (WSNs) consist of spatially distributed autonomous sensor (source) nodes with limited sensing, data processing, and communication capabilities. Nodes can sense physical parameters of their local environment (e.g. temperature, pollution, noise levels, etc) and send this information to a sink node via multi-hop wireless communication. Energy conservation is a key issue in the design of WSN applications because sensor nodes are powered by batteries. Since communication cost in terms of energy is much higher than computation one (e.g. the transmission of 1KB is equivalent to compute about 3 millons of instructions), it is preferred to implement simple processing tasks on nodes to reduce the message transmission. A WSN can be viewed as a huge distributed database system where users can query data [1]. These queries can be often processed (i.e. completed) by retrieving data from just a limited subset of nodes. Indeed, a query may not require fetching all data from the network

but only a few. Examples of these queries can be related to retrieving the number of nodes sensing values above a specific threshold or in a specific range, or to finding maximum/minimum values in the network, etc. These cases can be defined by threshold values which filter the actual relevant data to be reported back to the sink node. This process can be implemented as a raw data collection mechanism where all nodes report their sensed values to the sink node where data is processed in a centralized fashion. Node's computational intelligence (CI) is thus limited to disseminating queries and reporting messages from/to the sink node. This process can be enhanced by means of in-network data processing techniques [2] which aim at reducing the number of message exchange. Nodes can decide whether to forward a given message by considering the message content as well as local information available at the node.

Even if information discovery mechanisms in WSNs that make use of in-network processing have been proposed and widely studied, they are designed for event detection applications (i.e. *is there a specific event in the area?*) [3], [4], [5]. Instead, in this paper we consider the case of discovering relevant information and monitoring it afterwards in such a way that: i) if monitored nodes later result in sensing useless data they can be excluded from the query process, and ii) if unmonitored nodes happen to sense useful data they can be included in the query process. We are interested in data queries such as *what is the maximum value sensed in the network?*, as addressed in [6], [7]. In this context, our work considers the problem of routing the query to nodes with relevant information.

Since a priori no information is available regarding which nodes contain relevant data, our scheme implements a learning process to assess how much useful data are present in the network and self-configure to reduce the amount of message exchange. If source nodes, based on a distributed algorithm for localization, send also information about their geographic position to the sink in the response, a map with the location of relevant information can be obtained. For example, sensor nodes can be deployed over an oil spill in the ocean in order to determine the points of maximum oil concentration, and track the spill's direction of displacement over the time. Our scheme is fully distributed as each node decides whether to forward a message to its neighbors or not based on local available information. In this paper, we illustrate the case of discovery and monitoring max values of the network, which can be implemented by updating a threshold value inside the query message; however, other metrics can be implemented.

Routing mechanisms in WSNs are highly dependent on both the application and the network topology [8]. There is no optimal mechanism that fits all cases. In unstructured networks, where the sink has not knowledge about the target location (e.g., relevant data), query dissemination can be implemented based on flooding [4], [5], random walk [9], gossip, or gradients [6], [7]. Our proposal builds on ideas of the previous mechanisms. Indeed, query forwarding decisions are evaluated based on gradient information to assess the relevance of the surrounding data. If data can be assumed relevant, flooding is encouraged; otherwise, gossip routing is implemented. This scheme is similar to the well-known simulated annealing (SA) metaheuristic where randomness is used to explore distant areas

where relevant data can be present. Besides, as it will be explained in following sections, parallelization is natural when the broadcast nature of wireless transmissions is exploited; thus, multiple forks run simultaneously.

The rest of the paper is organized as follows. Section 2 introduces the network model and node behavior. Descriptions of the proposed query and data collection mechanisms are given in Section 3. The proposed learning algorithm is detailed in Section 4. Section 5 describes simulations and main results of the mechanisms. Finally, Section 6 discusses main conclusions and future work.

2 Network Model

We consider large, flat and unstructured WSNs of uniformly scattered sensor nodes over a square geographic area. A sink node, located at the center of this area, periodically injects on-demand query messages that are routed through the network to fetch relevant data. In flat networks all nodes have the same functionality. In unstructured networks the sink node has no clue where the relevant data reside, it uses blind sequential search for querying. Nodes exchange messages with all neighbor nodes within a fixed circular communication range.

A query message contains an identification number and a threshold value. This message is initially broadcasted by the sink node to all its neighbors who, after processing the query, can relay it to their neighbors, and so on. Even if nodes can receive the same query multiple times, they can at most forward it once. For this purpose, nodes keep track of the identification numbers of forwarded queries using local tables to avoid retransmissions. Query messages are always processed (even if received several times). In the simplest case, the threshold value in message is only read to determine if the node has relevant data. If so, the node configures itself to report its data to the sink node. In this case, the query is routed through the network to select those nodes above (below) a threshold value. This threshold value can be also updated (i.e., written) by nodes to implement, maxima (minima) search functions. Each node compares the read value from the query message to that locally sensed, if the node has a better value it updates the threshold value on the query message. Note that in this case, only those nodes that updated this value get configured to report data to the sink. However, since a node may receive the same query message more than once, it may happen that even if it first got configured to reply back, it could later dismiss its response due to receiving a query message with a better value than its own. From here on, we assume that the query process is used to find *maximum values* in the network.

After processing the query, nodes must decide whether to forward or not the message to its neigbors. Even if a detailed discussion of this decision process is described in the next section, we introduce its main principles:

• If a node finds out that its data are relevant, it will always relay the query message. This is due to the fact that more relevant data may be nearby the node.

• If a node concludes that it has useless data, it will randomly decide whether to relay the message or not.

Since network data are to be monitored over time, these queries are injected periodically by the sink node. Thus, each query iteration can result in fetching data from a different subset of nodes. On the one hand, data sensed by nodes may change driving the query to areas seen before as irrelevant and/or avoiding visiting nodes which now have useless information. Routing is data-driven: if data changes, routing paths may also change. On the other hand, randomness is introduced to explore areas where irrelevant data are present. As a result, new areas may be discovered at each query iteration. In general, we assume that the stochastic behavior of the network data does not change, which can be considered a valid assumption for several application scenarios.

3 Forwarding Algorithm

Each node must implement a forwarding algorithm that can process received messages. These messages can be either queries messages from the sink node or reply ones from source nodes. The former messages must be routed in the downstream direction (from sink to sources) while the latter ones, in the usptream direction (from sources to sink). In the downstream direction, queries should be routed over areas where relevant data can be found. Since no information is available a priori, a probabilistic routing based on the SA metaheuristics is proposed. In the upstream direction, replies should be directed to the sink node. Since this upstream routing is preceded by the downstream one, some information about the network topology can be used to drive replies to sink node. Next, we discuss the forwarding algorithm for both cases: downstrean and upstream.

3.1 Downstream Forwarding

To route queries through the network, a version of the well-known SA metaheuristic is used [10]. SA has the ability to explore beyond those areas where no relevant data are available. In our case, each node with no useful data can forward the query message to its neighbors with probability P computed as:

$$P(\Delta E, T) = e^{\frac{-\Delta E}{T}} \ .$$

where ΔE is the difference between the threshold value and the data sensed by the node, while T is the *temperature* parameter of the SA algorithm.

The largest ΔE, the lowest P as it should be less probable to accept relaying the query if data is completely irrelevant. A large value of $T = T_0$ is initially used and then decremented linearly. Note that a key issue is how to setup the initial value T_0. A value too large can flood the network with the query, while a value too small can limit discovering (hidden) relevant data.

Algorithm 1 describes the forwarding of queries in the downstream direction. Each node can only forward once each query in order to save as much energy as possible. Therefore, nodes keep updated an *id* table where recent messages *ids* are stored. Nodes that realize that have relevant data ($\Delta E < 0$) always broadcast the query to all its neghbors. Otherwise, nodes decide randomly whether to

Algorithm 1 Forwarding algorithm: Downstream only

```
1: id ← getId(Msg)
2: ...
3: T ← getTemperature(Msg)
4: thresholdValue ← getThreshold(Msg)
5: ΔE ← thresholdValue − sensedValue
6: if idTable[id] == true then {Discard already forwarded queries}
7:    delete(Msg)
8:    exit()
9: else
10:   updateIdTable(id)
11: end if
12: if ΔE ≤ 0 then {Decide forwarding of new queries}
13:   Msg ← setValue(sensedValue)
14:   Msg ← setTemperature(T)
15:   send(Msg)
16: else
17:   if P = e^{−ΔE/T} > rand() then
18:      Msg ← setTemperature(T − D)          /* D: Decrement value */
19:      send(Msg)
20:   else
21:      delete(Msg)
22:   end if
23: end if
```

forward or not the query based on the SA concept. Since each forwarded query is broadcasted more than one node can receive the same message. This results in a natural parallelization of the algorithm as each forwarded query can generate multiple forks of the same algorithm. Besides, nodes can forward the query with different values of T, which is known as adatpive cooling. Since we would like to encourage nodes to explore areas where relevant data can be assumed, nodes with relevant local data forward the query with the same temperature. Instead, nodes which assume irrelevant data decrement the T value linearly by a D parameter. We refer to the resulting algorithm as parallel adaptive simulated annealing (PASA).

3.2 Upstream Forwarding

During the downstream routing process, each node can learn about its minimum distance from the sink node (in number of hops). Besides, nodes can self-configure to reply to the query if they consider their data relevant. Each of these nodes sends a reply message back to the sink after a period of time related to the hop level plus a small random time. This message carries the relevant sensed data and the hop distance of the node. Besides, to identify if a message is addressed to the sink (reply) or from it (query), a special flag is used to mark messages as either downstream or upstream ones. Nodes receiving an upstream message forward the same message only if: i) their hop distance to the sink is smaller

than that carried on the message, and ii) the message has not yet been received or forwarded. All forwarded messages are updated with the hop distance of the node. In this way, nodes always forward a message if they realize that it gets closer to the sink node.

Algorithm 2 Forwarding algorithm: Downstream and Upstream

```
1: id ← getId(Msg)
2: type ← getType(Msg)
3: hops ← getHops(Msg)
4: if type == downstream then
5:    if hops < minHops then {Update minimum hop distance to sink node}
6:       minHops ← hops
7:    end if
8:    Msg ← setHops(minHops + 1)
9:    ... (Algorithm 1 - Downstream forwarding) ...
10:   if ΔE ≤ 0 then {Update reply status}
11:      replyQuery ← true
12:   else
13:      replyQuery ← false
14:   end if
15: else
16:   if idTable[id] == true then {Discard already received replies}
17:      delete(Msg)
18:      exit()
19:   else
20:      updateIdTable(id)
21:   end if
22:   if hops > minHops then {Decide forwarding of new replies}
23:      Msg ← setHops(minHops − 1)
24:      send(Msg)
25:   else
26:      delete(Msg)
27:   end if
28: end if
```

The complete forwarding algorithm is summarized in Algorithm 2. Figure 1 illustrates the different behaviors a node can have for the downstream case. In Figure 1(a) the case where relevant data are available at the node is shown. Since we consider the case of computing the maximum value, the query message is updated with the node's value and broadcasted to its neighbors. Note that the value of T remains unchanged as relevant data were found. On Figure 1(b) and 1(c) the case where no relevant data are available is illustrated. In both cases, the node compares the probability P with a random number to determine whether to forward the query. In Figure 1(b) the query is relayed and, as a result, the value of T decremented; while in Figure 1(c) the query is discarded. Note that nodes self-configure to reply back the query only when they recognize

themselves as having relevant data as shown in Figure 1(a). For this purpose, a timer is used to leave some time to the node to eventually process updated queries that can disabled its response.

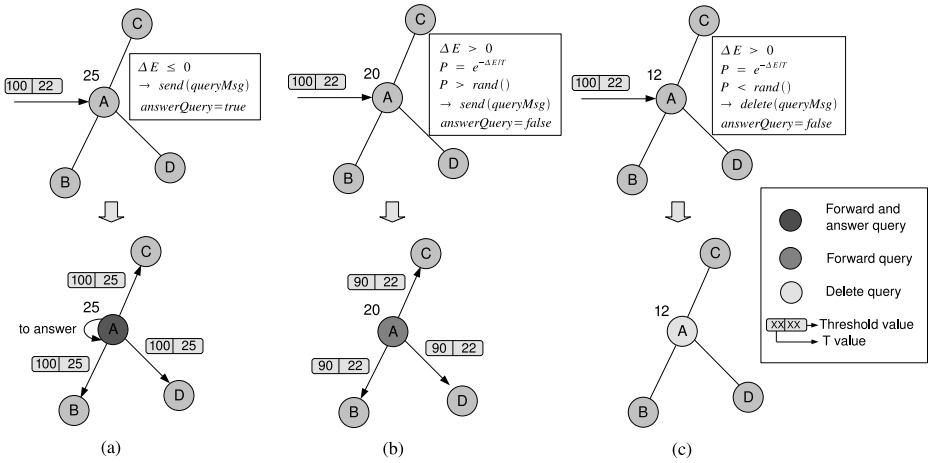


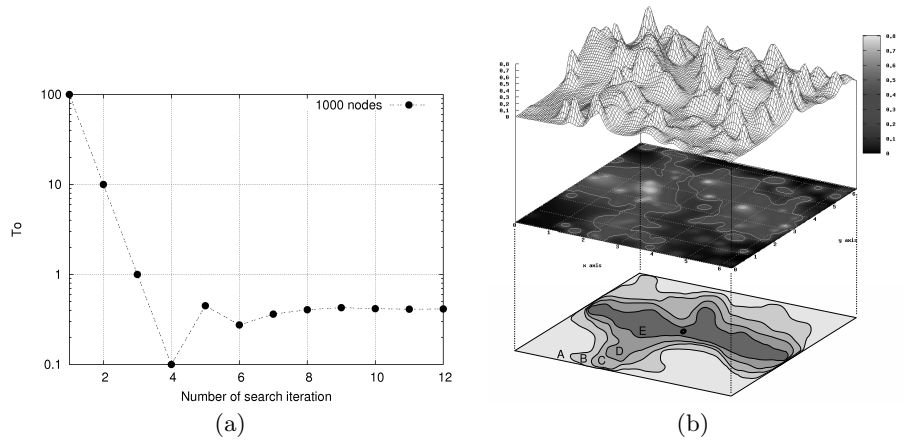
Fig. 1. Downstream query routing examples, (a) relevant, and (b), (c) irrelevant data.

4 Learning Algorithm

For a large value of T_0 our downstream mechanism can behave as flooding, as every node would always forward the query despite the data relevance; thus, there is certainty in always finding the most relevant data. At low T_0 values, it is equivalent to the gradient descent algorithm as the message is forwarded only through nodes which show some relevant data. As a result, the probability of finding the most significant data is low. Therefore, it is neccessary for our algorithm to find an optimal T_0 value to initialize T so that when linearly decremented (over bad moves) it offers a good trade-off between routing cost and query error. For this purpose, a reinforcement learning (RL) algorithm is implemented in the sink node. RL is a biologically inspired algorithm that acquires its knowledge by exploring its environment. It is easy to implement, highly flexible to topology changes, and well suited for distributed problems such as routing [11]. The main idea behind this algorithm is that the sink can learn about the distribution of information in the network based on its query experience. It is a simple strategy to improve the search for relevant information in large WSNs when the sink has no knowledge of the data distribution. By changing the initial T_0 value, the sink node can limit the search depth of each query iteration, reducing energy consumption. This process is sketched in Figure 2(a).

Even if nodes are scattered over an area A as illustrated in Figure 2(b), a first query iteration reaches nodes on area B. Nodes on (A-B) area where 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 value is decreased, which reduces the search space to area C. This process is repeated till the algorithm finds out the value of T_0 that is able to query all nodes with relevant data at the lowest cost, which in Figure 2(b) is represented by area E. At each query iteration, a decision on whether to decrease the initial T_0 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 nodes, we can assume that the initial T_0 value can be reduced. On succesive queries, the sink decreases the injected T_0 value while it gets the same number of responses. If this number decreases, the T_0 value is increased. A logarithm decrease-linear increase scheme is used to adapt the initial T_0 value. After a few iterations, T_0 converges to a fixed value which is used in succesive queries. Since both the network topology and data may change in time, this learning process is run periodically to adapt to network variations.



(a) (b)

Fig. 2. (a) T_0 value at each iteration. (b) Search space reduction at each iteration.

5 Simulation Results

In order to evaluate the proposed routing scheme two metrics were considered: query error and cost. A realistic network simulator based on the discrete event simulation package Omnet++ [12] was developed for this purpose.

The first represents the gap between the best reported data to the sink after a query and the actual optimal one. The latter considers the number of times on average each node forwarded the query. Note that the maximum query cost is equal to 1, which means that all nodes relayed the query once (i.e., similar

to flooding). These metrics were evaluated as a function of network size, node density, and initial T_0 value. Nodes are uniformly deployed over a square surface obtaining their readings as a function of their positions. This surface is given by the sum of 160 decreasing exponential functions with different values of position, amplitude, and decrement. For illustrative pruposes, an example of the resulting surface is shown in Figure 2(a). Reported results are the average of 1000 random simulations. A circular communication range of 40 meters is assumed for nodes. A detailed description about the number of neighbors needed to ensure good connectivity in multi-hop wireless networks is given in [13].

5.1 Performance

The performance of our mechanism as a function of network size is shown on Figures 3(a) and 3(b). A node density of $2.5\text{x}10^{-3}$ nodes/square meters is assumed. Different initial values of T_0 are considered to illustrate its impact; thus, in this case the sink does not implement the learning algorithm discussed in Section 4. Note that large T_0 values ($T_0 \geq 10$) result in always retrieving all relevant data (i.e., query error ≈ 0 as it can be appreciated from Figure 3(a). However, these values encourage almost all nodes to forward the query message, which is similar to simply flooding the network. On the contrary, too small T_0 values ($T_0 \leq 0.1$) result in significant query errors proportional to the network size. Therefore, it can be seen that there exist a trade-off between errors and cost where a range of T_0 values can offer good performance. For these values, errors can be bounded to less than 1%, while the query cost tends to decrease for larger networks.

Besides network size, node density impacts on the performance of our scheme as shown in Figures 3(c) and 3(d). Denser networks tend to decrease the error gap, requiring smaller initial T_0 values. This can be seen in in Figure 3(c) where for $T_0 = 0.1$ the query error diminishes as the node density increases. Increasing node density does not impact much on the query cost. Indeed, as shown in Figure 3(d), this cost has an asymptotic behavior, which means that larger densities result in slow increments of the query cost.

5.2 Adaptation

As discussed in Section 4, the sink can implement a simple reinforcement learning algorithm to find out initial T_0 values that can offer good performance. Figure 4 shows how this algorithm is able to setup initial T_0 values that result in both low error and cost.

The first query is sent with a pre-configured T_0 value, typically high enough to flood the network. On the second query message, this value is decreased which reduces the query cost while still offering a low error. After repeating this process in succesive queries, the error increases, which triggers the slowly (i.e., linear) increase of T_0. Recall that the sink cannot be aware of the error cost, which we show in Figure 4(a), but only of changes on the number of nodes that reply the query back. After a few iterations, the initial value of T_0 converges to a small value. Note that the query cost can be decreased to about 50% with respect to flooding (see Figure 4(b)) while still keeping the error bounded to less than 5%.

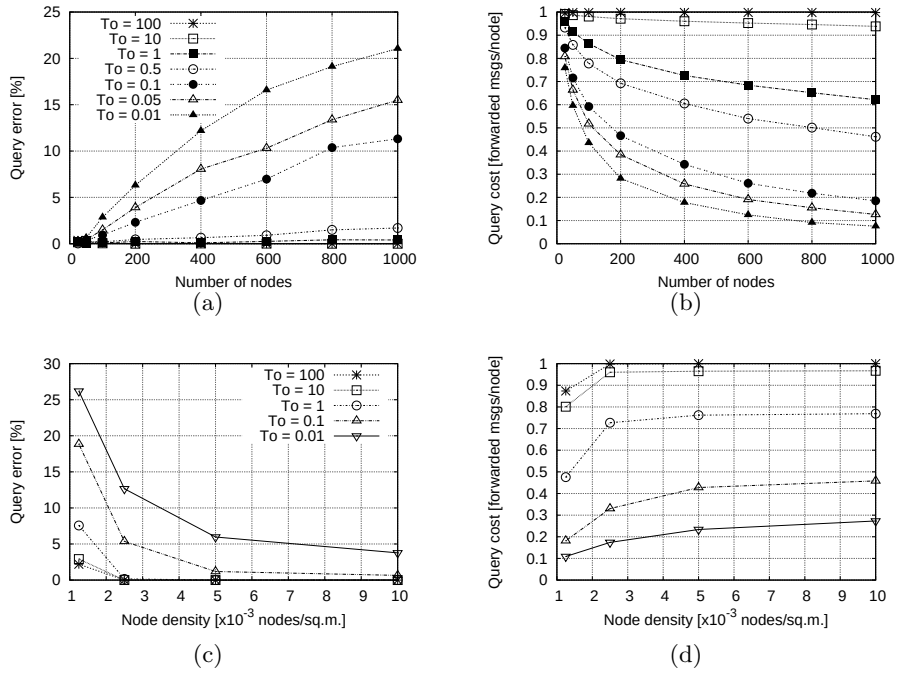


Fig. 3. Query error and cost in terms of network size (a),(b), and node density (c),(d).

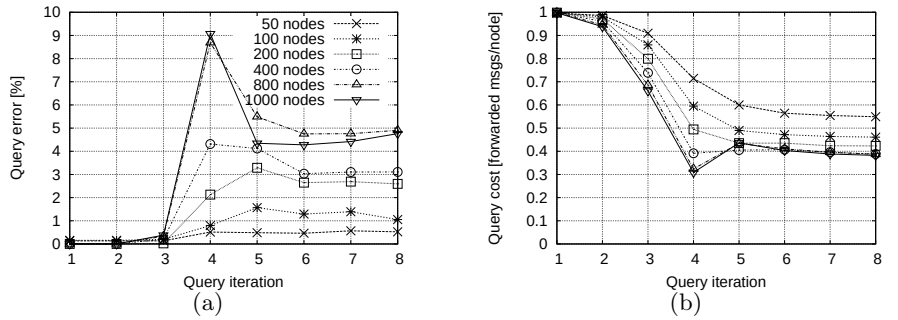


Fig. 4. Adaptation process: (a) Query error and (b) number of transmitted messages in query dissemination in terms of query iteration and network size.

5.3 Comparison

A comparison of the proposed mechanism respect to flooding and random walk is shown in Figures 5(a) and 5(b). In the case of random walk, we consider a maximal hop count value equal to the number of nodes. A query cost reduction of 60% with an error less than 5% can be obtained by using PASA respect to the other mechanism.

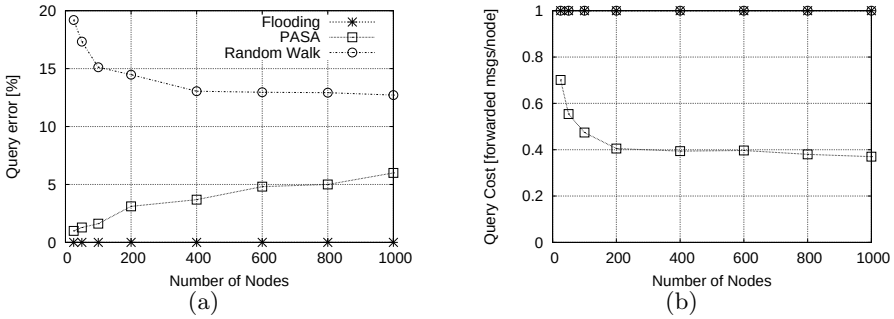


Fig. 5. Query (a) error and (b) cost for different routing mechanisms

5.4 Robustness

Finally, the robustness of the scheme is evaluated considering three possible scenarios. The first one takes into account variations on the surface that represents the continuous field to monitor. For this purpose, after the adaptation process, at each new query iteration we randomly change each of the 160 building functions up to a maximum value of 10%, 50% or 90%. Figure 6(a) shows that resulting query error increments as the network size increases. However, the error can be kept bounded to less than 5-10%. The second one considers node failures after adaptation process. Nodes are selected uniformly to fail with different percentages of failures (between 0 and 10%), obtaining errors less than 10% in networks with 5% of failed nodes. This is shown in Figure 6(b). The third one takes into account the packet loss probability in the network. We show in Figure 6(c) the performance with packet loss probability between 1% and 10%. All the figures were generated with 1000 random simulations. We conclude that this mechanism is robust as it can tolerate data changes, node failures, and packet loss.

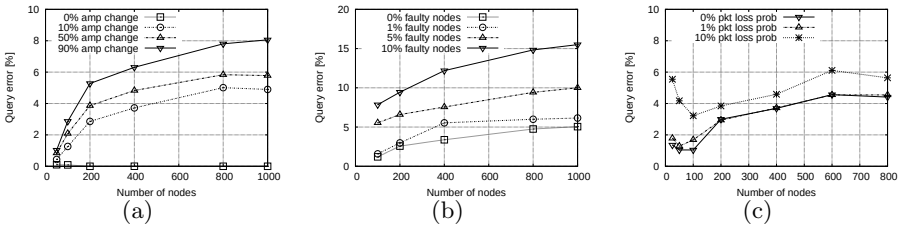


Fig. 6. Query error in terms of (a) amplitude change after adaptation, (b) percentage of failed nodes, and (c) packet loss probability.

6 Conclusion

In this paper, we proposed PASA, an energy-efficient probabilistic mechanism for routing queries in WSNs where only some nodes have relevant data. We showed how simple algorithms can be implemented distributely to forward both upstream and downstream messages. The scheme significantly outperforms flooding-based and random walk-based models in terms of energy cost. Energy consumption due to message exchange can be reduced by more than 60% as compared to flooding. Energy savings can be made higher at the cost of increasing the query error. Future work considers the implementation of proofs-of-concept of this routing mechanism in real sensor networks.

Acknowledgment. This work is partially funded by the Universidad Tecnolgica Nacional - FONCyT IP-PRH 2007 Posgraduate Grant Program and by the SECYT-UNC 2010-2011 Research Program.

References

1. Gehrke, J., Madden, S.: Query Processing in Sensor Networks. IEEE Pervasive Computing, Vol. 3, No 1, pp. 46-55 (2004)
2. Anastasi, G., Conti, M., Di Francesco, M., Passarella, A.: Energy Conservation in Wireless Sensor Networks: A Survey. Ad Hoc Networks 7, pp. 537-568 (2009)
3. Ahn, J., Kapadia, S., Pattem, S., Sridharan, A., Zuniga, M., Jun, J.: Empirical Evaluation of Querying Mechanisms for Unstructured Wireless Sensor Networks. In ACM SIGCOMM Computer Comm. Review, Vol. 38, No 3, pp. 19-26 (2008)
4. Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. MobiCom, pp 56-67. (2000)
5. Cheng, Z., Heinzelman, W.: Flooding Strategy for Target Discovery in Wireless Networks. In ACM International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems, USA (2003)
6. Chu, M., Haussecker, H., Zhao, F.: Scalable Information-Driven Sensor Querying and Routing for ad hoc Heterogeneous Sensor Networks. International Journal of High Performance Computing Applications, Vol. 16, No 3, pp. 293-313 (2002)
7. Faruque, J., Helmy, A.: RUGGED: Routing on fingerprint gradients in sensor networks. In IEEE International Conference on Pervasive Services, Lebanon (2004)
8. Al-Karaki, J. N., Kamal, A. E.: Routing Techniques in Wireless Sensor Networks: A Survey. In IEEE Wireless Communications, Vol. 11, No 6, pp. 6-28, (2004)
9. Braginsky, D., Estrin, D.: Rumor Routing Algorithm for Sensor Networks. In ACM Intern. Workshop on Wireless Sensor Networks and App., pp. 22-31, USA (2002)
10. Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P.: Optimization by Simulated Annealing, Science, New Series, Vol. 220, No. 4598, pp. 671-680 (1983)
11. Kulkarni, R. V., Förster, A., Venayagamoorthy, G. K.: Computational Intelligence in Wireless Sensor Networks: A Survey. IEEE Communications Surveys & Tutorials, Vol. 13, No 1, First Quarter (2011)
12. Omnet++ simulation library, http://www.omnetpp.org/
13. Xue, F., Kumar, P. R.: The Number of Neighbors needed for Connectivity of Wireless Networks. Wireless Networks, Vol. 10, No 2, pp. 169-181 (2004)