A Parallel and Adaptative Query Routing Scheme for Wireless Sensor Networks

Guillermo G. Riva  
Universidad Tecnológica Nacional  
Department of Electrical Engineering, Córdoba, Argentina  
griva@sctd.frc.utn.edu.ar

Jorge M. Finochietto  
Universidad Nacional de Córdoba – CONICET  
Digital Communications Lab, Córdoba, Argentina  
jfinochietto@efn.uncor.edu.ar

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 sink node disseminates a query message through the network to fetch data from sensor nodes. As several applications only require a limited subset of the available data in the network, the query could be ideally routed to obtain only relevant information, hence minimizing energy consumption. In this work, we apply computational intelligence algorithms to adapt the query process to the characteristics of the network in order to direct queries to relevant nodes, thus, limiting the routing space. This proposal is validated by extensive simulations which show that the routing cost can be reduced by approximately 60% over flooding with an error less than 5%.

I. INTRODUCTION

Wireless sensor networks (WSNs) consists of spatially distributed autonomous sensor nodes with limited sensing, processing, and communication capabilities. Nodes can sense physical parameters of their local environment and send this information to a sink node via multi-hop communication. Nodes are battery-powered, and the energy conservation is a key issue in the design of WSN applications [1]. Since communication cost in terms of energy is much higher than computation one (e.g. transmission of 1KB is equivalent to compute about 3 millions of instructions), it is preferred to reduce the message transmission count by processing data on nodes rather than sending all data to the sink. In this context, nodes can decide whether to forward a given query message by considering its content as well as local information available at the node. Even if in-network data processing in WSNs has been proposed and widely studied, it is designed for event detection applications (i.e. is there a specific event in the area?) [2], [3], [4]. Instead, we consider the case of monitoring general information of the network. We are interested in issues such as what is the top (i.e., maximum) reading?, as addressed in [5], [6] or more generally what are the 10% top readings?.

A WSN can be considered a huge distributed database system where users can query for specific information [7]. Most queries can be completed by processing data from just a limited subset of nodes. To this end some filtering of the available data on the network is required, which can be implemented by embedding a filter threshold on the query message. These filters can be either static as in the case of finding the number of nodes with readings within a specific range, or dynamic when searching for maximum (minimum) readings. Note that static filtering is based on fixed thresholds, so it can be implemented locally by each node despite the actual readings available at other nodes. On the contrary, dynamic filtering requires each node to evaluate whether to update or not the filter threshold based on local readings.

In this context, raw data (i.e., not filtered at all) collection mechanisms can waste much energy as not all fetched data can be actually required. Query mechanisms can perform much better as (static or dynamic) filtering thresholds can be disseminated through query messages to avoid reporting useless data to the sink node. For applications that require static filters, the query message can be flooded only once as thresholds remain unchanged, letting nodes to report back values only when their data match the filter. However, for applications requiring dynamic filters the query needs to be disseminated periodically to compute new thresholds resulting from current readings. A trivial solution is to periodically flood the query to all nodes. A more challenging solution is to try to learn where potential relevant data are located and route the query to these areas only. In this way, the cost of disseminating periodically the query to all nodes can be significantly reduced at the cost of allowing some error in the response. This paper proposes a novel mechanism that considers this challenge. The scheme implements computational intelligence (CI) processes to learn how much useful data are present in the network and self-configure to reduce the message exchange count.

The rest of the paper is organized as follows. Section II introduces the network model and the node behavior. Descriptions of the proposed query and data collection mechanisms are given in Section III, while the learning algorithm that adapts the query process to the network characteristics is detailed in Section IV. Section V describes simulations and main results of the mechanisms. Finally, Section VI discusses main conclusions and future work.

II. NETWORK MODEL

We consider large and unstructured WSNs composed of sensor nodes uniformly distributed over a square area. A sink node, located at the center of this area, injects on-demand query messages that are routed through the network to fetch relevant data. Nodes can receive messages, and process them to decide i) if their readings are relevant and need to be reported
to the sink, and ii) whether or not to broadcast the messages (i.e., forward the message to all neighbor nodes within their communication ranges).

A query message contains an identification number \( (id) \) 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 \( ids \) of forwarded queries using local tables to avoid retransmissions. Query messages are always processed (even if received several times). Each time a node receives the query, its threshold value inside the message is read to determine if the node has relevant data. In the case of static filters, where the query is routed through the network to select those nodes above (below) a given (fixed) threshold value, nodes can determine the relevance of its reading on the first query arrival. In dynamic filter cases, the threshold can be updated by nodes to implement, maximum (minimum) search functions. 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 is not relevant at all as the threshold value has been updated. Once the query process ends, only nodes with relevant data report their readings to the sink node.

After processing the query, nodes must decide whether to forward or not the message to their neighbors. Monitoring with static filters requires always forwarding the message once to flood the network and disseminate the fixed threshold to all nodes. Instead, monitoring with dynamic filters requires periodically updating the thresholds to determine data relevance on nodes, which can be implemented by repeating the query dissemination process. However, these repetitions can benefit from a learning process which can help to route following queries in a more efficient way. Even if a detailed discussion of this routing process is described in the next section, we introduce its main principles:

- If a node finds out that its reading is relevant, it will always relay the query message. This is due the fact that more relevant data may be nearby the node.
- If a node concludes that has useless data, it will randomly decide whether to relay the query message or not.

Since data are monitored over time, queries are injected periodically by the sink node. Thus, each query iteration can fetch data from different subsets of nodes. On the one hand, data sensed by nodes may change in time driving the query to areas seen before as irrelevant and/or avoiding visiting nodes which now have useless information. Indeed, query 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. We assume that network data distribution changes slowly with respect to the query frequency (i.e., the frequency at which retrieved information is updated/monitored), which is a valid assumption for most application scenarios.

In this work, we consider the case of implementing a dynamic filter used to find the maximum (minimum) reading in the network. However, it is worth mentioning that other implementations are feasible like monitoring the top-p\% readings by embedding additional information in the query message.

III. Forwarding Algorithm

Each node must implement a forwarding algorithm in order to process received messages. These messages can be either queries messages from sink node or replies from source nodes. The former must be routed in the downstream direction (from sink to sources) while the latter, in the upstream one (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 simulated annealing (SA) metaheuristic 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 back to the sink node. Next, we discuss both downstream and upstream forwarding algorithms.

A. Downstream Forwarding

To route queries through the network, a version of the SA metaheuristic is used [8]. 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 \( \Delta E \) is the difference between the threshold value carried in the query message and the sensor reading, while \( T \) is the temperature parameter of the SA algorithm. The largest \( \Delta E \), 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.

Each node can only forward once each query in order to save as much energy as possible. Nodes that realize that have relevant data (\( \Delta E < 0 \)) always broadcast the query to all its neighbors. Otherwise, nodes randomly decide whether to 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 adaptive 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
behave as flooding, as every node would always forward the
resulting algorithm as parallel adaptive simulated annealing
(PASA).

B. Upstream Forwarding

During the downstream routing process, each node can learn
about its minimum distance to the sink node (in number of
hops). Nodes assuming their data relevant report their readings
to the sink after a random time inversely proportional to
the hop count. The larger the hop count, the shortest the
reply time. This encourages to propagate replies from leaf
nodes towards the sink node, which enables the filtering
of further data. Since nodes always receive (and process)
both downstream and upstream messages, they can determine
whether their readings are still worth of reporting or not.
Indeed, filtering is actually implemented in both downstream
and upstream directions.

The reply message carries the relevant sensed data and a hop
distance value. 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.

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 while relevant
data are 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 $T$ is decremented;
while in Figure 1(c) the query is discarded. Note that nodes
will reply the query as long as they recognize themselves as
having relevant data as shown in Figure 1(a).

IV. 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 query
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 necessary for our algo-

rithm 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 based on
reward and punishment. It is easy to implement, highly flexible
to topology changes, and well suited for distributed problems
such as routing [9]. The main idea behind this algorithm is that
the sink can learn about the data distribution in the network
based on its experience. It is a simple strategy to improve
the search for relevant information in large WSNs when the
sink has not knowledge of the data distribution. The sink
node adapts the initial $T_0$ value on successive queries (e.g.,
iterations) to improve the search process, hence reducing the
energy consumption. By changing the initial $T_0$ value, the
sink limits the search depth at each iteration. 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 decreases, which
reduces the search space to area C. This process is repeated
till the algorithm finds out the value of $T_0$ that enables 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 number of
responses, we can assume that the initial $T_0$ value can be
reduced, so the sink decreases the injected $T_0$ value. A lower
$T_0$ tends to narrow the search space and, thus, save energy on
nodes. If this number of responses decreases, the $T_0$ value is
increased. A logarithmic decrease and linear increase scheme

Fig. 1. Downstream query routing examples.

Fig. 2. (a) $T_0$ at each iteration. (b) Search space reduction at each iteration.
is used to adapt the initial $T_0$ value. After a few iterations, $T_0$ converges to a fixed value which is used in successive queries. Since both network topology and data may change in time, this learning process can be run periodically to adapt to network variations.

V. SIMULATION RESULTS

The proposed routing scheme was implemented in Ommet++ [10] in order to evaluate its performance. Two metrics were considered: query error and cost. The first one represents the gap between the best reported data to the sink after a query and the actual optimal one (i.e., the relative error). The latter one considers the number of times on average each node forwarded the query message. Note that the maximum query cost is equal to 1, which means that all nodes relayed the query once (i.e., 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. The sensed data surface is the result of summing $S$ decreasing exponential functions with different values of position, amplitude, and decrement. Indeed, our data model is based on multiple independent sources which can represent air conditioners, lights, speakers, etc. For illustrative purposes, an example of the resulting surface is shown in Figure 2(b). Our simulations consider the case of $S = 160$ and report the average results of 1000 random simulations. A circular communication range of 40 meters is assumed for nodes and a node density of 2.5x10^{-3} nodes/square meters.

A. Performance

The performance of our mechanism as a function of network size is shown in Figures 3(a) and 3(b). 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 IV. Note that large $T_0$ values result in flooding the network and retrieving all relevant data (i.e., query error $\approx 0$ and query cost $\approx 1$). On the contrary, too small $T_0$ values result in significant query errors proportional to the network size. Therefore, it can be seen that there exist a trade-off between error and cost where $T_0$ 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 (Figure 3(c)). 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.

B. Adaptation

As discussed in Section IV, the sink implements a reinforcement learning (RL) algorithm to find out the initial $T_0$ value 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 high $T_0$ value, enough to flood the network. On the second query, this value is decreased which reduces the query cost while still offering a low error. After repeating this process, 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 to the query. After a few iterations, the initial value of $T_0$ converges to a small and constant value. Note that the query cost can be decreased to about 50% with respect to flooding while still keeping the error bounded to less than 5% as shown in Figure 4(b).

C. 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. This comparison considers only query disseminating cost (i.e., query cost) as the cost of replies is negligible with respect to the query one.

A totally different alternative is to consider flooding once the network to first retrieve all readings and then let sensor nodes report their data only when a significant change in their readings happen. Since the query is disseminated only
once (during the first iteration) we refer to this scheme as single-query (SQ). To bound the error to some threshold we assume that nodes report new data to the sink only when the data changes more than 5%. To compare the performance of both PASA and the SQ approach, we need to compare all (downstream and upstream) messages as in the SQ case only upstream (replies) messages are present. This analysis is shown in Figure 5(c) considering that at each iteration (i.e., monitoring interval) each data source changes uniformly between a given percentage. It is clear that if data sources do not change (0% case), SQ outperforms PASA as it does not require new data from nodes after the first iteration. However, if sources do change a little bit some nodes can sense differences above the 5% threshold and report new data. After some iterations the overall cost converges to about 4 messages per node for the case of 1% source variations. It is worth noticing how PASA performs much better by only sensing differences above the 5% threshold and report new data.

After some iterations the overall cost converges to about 4 messages per node for the case of 1% source variations. It is worth noticing how PASA performs much better by only directing the query to relevant data areas and thus limiting the overall cost to about 1 message per node.

D. Robustness

The robustness of the scheme is evaluated considering three possible scenarios after the adaptation process. Figure 6(a) shows the impact of variations on the data model while Figure 6(b) the case for node failures. Finally, the impact of the packet loss probability is illustrated in Figure 6(c)). In general, the proposed scheme is much resilient to variations on the network conditions.

VI. Conclusion

In this paper, we proposed PASA, a self-learning mechanism to monitor relevant data in WSNs. We showed how simple algorithms can be implemented distributely to route both upstream and downstream messages efficiently. The scheme significantly outperforms flooding-based and random walk-based models in terms of the message exchange cost, obtaining more than 60% reduction as compared to flooding. Cost savings can be made even higher at the cost of increasing the query error. Future work considers implementing much more complex dynamic filtering techniques under the same strategy.

VII. Acknowledgments

This work was partially funded by the National Technological University - FONCyT IP-PRH 2007 Postgraduate Grant Program and by the SECYT-UNC 2010-2011 Research Program.

REFERENCES


