LEQS: Learning-based Efficient Querying for Sensor Networks

Bhaskar Krishnamachari, Congzhou Zhou, Baharak Shademan
University of Southern California, Los Angeles, CA 90089

June 26, 2003

Abstract

We consider repeated one-shot queries for an identifiable object or named datum in a wireless sensor network. If there is no underlying pattern, querying for such data can be an expensive operation. However, if the object being queried for has an underlying probabilistic spatial description, though it may be unknown a-priori, we show that the energy cost of querying can be significantly reduced over time through reinforcement learning. We develop a simple localized and distributed learning-based efficient querying mechanism (LEQS). In this algorithm, sensor nodes maintain weights indicating the probability with which a given query is forwarded to each neighbor. The query response is used to update these weights on the reverse-path, effectively training the network to locate the object efficiently. Our extensive simulation results show that this in-network learning scheme can significantly reduce the energy expenditure of querying and that it is self-healing in case of node failures. With sufficiently many repeated queries and appropriate parameter settings, LEQS can provide near-optimal performance. In the scenarios we study, the learning process offers 50 to 75% gains in energy savings.

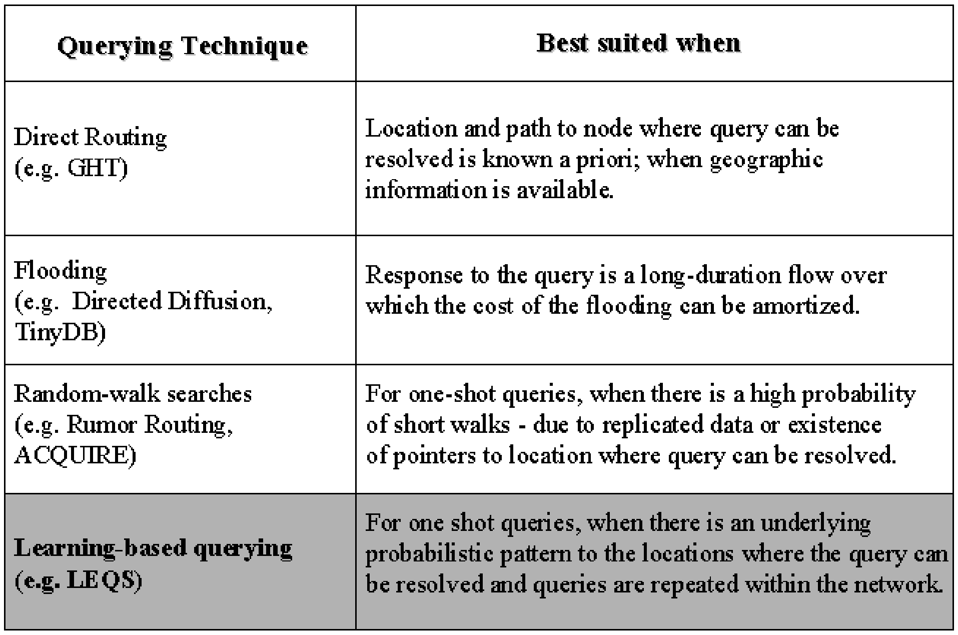
1 Introduction

Networks of embedded sensors can be used to obtain information about the physical world in several ways, depending on the application. In continuous data-gathering applications, all or some subset of nodes in the network may be tasked to provide data in a periodic manner, possibly in conjunction with data-aggregation techniques. It has been argued [5, 6] that one could view large-scale wireless sensor networks as distributed databases. The end user may be interested in some specific information instead of all data collected within the network. The basic problem of how to extract that information from the wireless sensor networks is that of querying. In prior work it has been shown that queries can be (i) flooded [1], (ii) forwarded randomly [2, 3], or (iii) routed directed to the object if the path is known [4]. We present an alternative approach in this paper – (iv) Learning-based Efficient Querying for Sensor networks (LEQS – pronounced "lex") .

First let us understand the circumstances for which flooding, random forwarding and direct routing are best suited. Direct query routing to the object (or to a rendezvous point) is the best solution for structured environments when (a) the location of the source which can provide a response to the query, and (b) a route to that source are both known a-priori. If the source is unknown, flooding is a good approach when the expense of the querying can be amortized over a large number of responses. Flooding is best suited when it is used to set up a long-standing information flow (this is the underlying assumption in the directed diffusion routing protocol).

Finally, random-walk-based search techniques are best suited for one-shot queries[1] when there is a reasonably high probability of locating the object or a pointer to the object with a small number of steps (e.g. when

[1]These are queries for which a single node in the network can provide an immediate response. An example of these are queries for the current location of objects known to be in the sensor network, such as "where is target X located?"



Querying Technique	**Best suited when**
Direct Routing (e.g. GHT)	Location and path to node where query can be resolved is known a priori; when geographic information is available.
Flooding (e.g. Directed Diffusion, TinyDB)	Response to the query is a long-duration flow over which the cost of the flooding can be amortized.
Random-walk searches (e.g. Rumor Routing, ACQUIRE)	For one-shot queries, when there is a high probability of short walks - due to replicated data or existence of pointers to location where query can be resolved.
Learning-based querying (e.g. LEQS)	For one shot queries, when there is an underlying probabilistic pattern to the locations where the query can be resolved and queries are repeated within the network.

Figure 1: A classification of querying techniques suitable for wireless sensor networks

there are multiple replicas of or pointers to the object in the network). If there is only a single location in the network where the query can be resolved, however, then a blind random-walk search will on average require the same order of energy expenditure as a flooded query. Scenarios in which the object has an underlying non-uniform location distribution (which does not have to be known a-priori to the querying node) have not been previously considered in the sensor network literature. For such scenarios, we develop the the LEQS mechanism where queries can be routed based on learning. This discussion is summarized in figure 1.

The key insight we exploit is this: if there is an underlying distribution that describes the location of the object, and there are repeated queries for it, it should be possible to "learn" how to query for this object efficiently over time. We develop a simple localized and distributed learning algorithm for querying. In this algorithm, sensor nodes maintain weights indicating the probability with which a given query is forwarded to each neighbor. The query response is used to update these weights on the reverse-path, *effectively training the network* to locate the object more and more efficiently over time.

The distributed algorithm we present has several desirable features in the context of sensor networks. It involves only local interactions and makes minimal assumptions about the network; in particular, there is no need for global unique identifiers for nodes or geographical information or even an underlying routing mechanism for the purposes of querying (some of these may or may not be needed in the application-level content of the response).

Since sensor nodes are likely to be battery powered, energy conservation is a crucial requirement in sensor networks. For one-shot queries flooding is not a suitable method because of its high-energy cost. On the other hand, directing the query to the node containing the object may be the optimal method due to its low-energy cost. However that achievement requires additional geographic and content information that may not be available. Our learned querying mechanism starts by randomly forwarding the query. Through learning over time, our approach can achieve energy costs that are similar to the optimal directed query method.

The algorithm we proposed can be flexibly extended for multi-sink scenarios. It is also very scalable — if the number of neighbors of each node is at most k, and the total number of objects (there could be more than one) being queried for is T, then the in-network storage required is $O(kT)$ for each node. Finally, because of its inherently adaptive nature, the query learning algorithm we propose is very robust to node failures.

We next provide details on our assumptions, and the algorithm, followed by an extensive set of simulation results to show its performance under various sample scenarios. We will then discuss related work and present concluding comments.

2 Model, Assumptions and Algorithm

2.1 Assumptions

LEQS requires that the communication links be bidirectional. If it is not strictly true at the physical layer, this may be provided through a simple pruning of unidirectional links.

No node localization or geographical information is required. There may be multiple sinks querying for the same objects. The named object (there could be many such objects) being queried for can be found (or a response to the query can be obtained) at only one location in the network at any given time[2], but it may potentially be at one of many locations. There is assumed to be an underlying stationary spatial location distribution for each identifiable, queried object (but this distribution need not be known to the network or to any entity). This assumption about an underlying regularity/pattern in the object's location is crucial to LEQS – there must be an underlying location pattern that can be learned. In the results section, we will investigate both a scenario where the object is located in a single location with probability one (which is the most simple case), and the general case when it can be located at one of multiple locations in the network with different probabilities.

The LEQS algorithm does not require the use of global unique identifiers for nodes; it only requires unique identifiers for each identifiable objects being queried for – a more scalable requirement. Each node communicates through query forward and query response packets only with its immediate neighbors. Because the algorithm is independent of the location where the query is issued, it can be readily used for scenarios in which multiple sinks issue queries for the same object.

Finally, it is assumed that multiple queries are issued for the same object over time. Otherwise there will be no opportunity for improving the energy efficiency of the querying via learning.

2.2 LEQS Algorithm Description

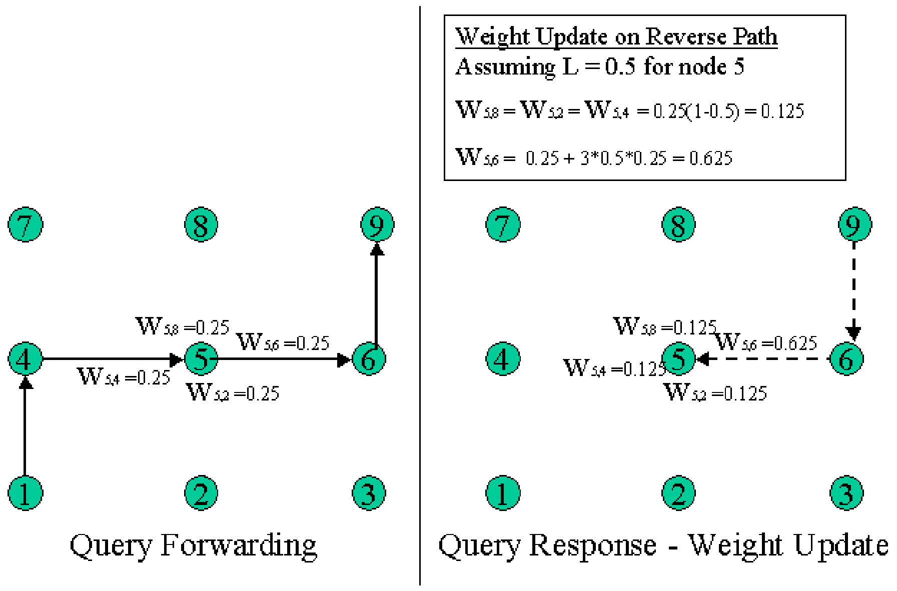
We now describe the LEQS algorithm.

Upon node deployment and setup, each sensor node i identifies its immediate neighbors and sets up a vector of weights $\overline{W_i}$ (one for each identifiable queried object A) in a querying table. Weight $W_{i,j}$ represents the probability that a query for object A that arrives at node i will be forwarded to node j.

Initially, if a given node has k neighbors, each neighbor of the node is assigned an equal weight of $\frac{1}{k}$. At any given time, each query will start from the sink, and with the probabilities denoted by the weights at each node, will be forwarded randomly from node to node. Each node will forward the query to one of its neighbors, except the node it has received the query from, according to their weight. Each node that receives the query checks to see if the queried object is located at that node. A backtracking technique is incorporated to prevent looping[3]. Eventually the query will find its way to the node where the object is located. The

[2] This kind of query is useful, for example, to locate identifiable targets in the sensor network region

[3] In the LEQS algorithm, every attempt to sending a query to the object results in a success since the walk backtracks if a loop is encountered. With sufficient time, the entire network will be searched in the worst case. A TTL field must be added



Weight Update on Reverse Path
Assuming L = 0.5 for node 5
$W_{5,8} = W_{5,2} = W_{5,4} = 0.25(1-0.5) = 0.125$
$W_{5,6} = 0.25 + 3*0.5*0.25 = 0.625$
$W_{5,8} = 0.25$
$W_{5,6} = 0.25$
$W_{5,4} = 0.25$
$W_{5,2} = 0.25$
$W_{5,8} = 0.125$
$W_{5,6} = 0.625$
$W_{5,4} = 0.125$
$W_{5,2} = 0.125$
Query Forwarding
Query Response - Weight Update

Figure 2: Illustration of weight update in LEQS

response of the query is then sent back directly to the sink on what is essentially the reverse path of this query (using information recorded at each node along the way, bypassing any backtracked branches) and this is when the weight updates occur. Each node i on the reverse path increases the weight of its next-hop h_i (i.e. the next node on the forward path between i and the node where the query terminated successfully). The query response on the return path contains a counter d that is incremented hop by hop, so that all nodes on the reverse path get an indication of how many hops they were from the query termination location. This information is used in the weight update rule, which is described below.

Each node on the reverse path first calculates a learning factor L_i as follows:

$$L_i = \frac{p}{d_i^{\alpha}} \tag{1}$$

In the above equation, $p \in [0, 1]$ and α are learning parameters that determine the rate of learning as well as the dependence on d_i, the distance (in hops) of node i from the query termination point. Let h_i be the node from which i receives the query response (i.e. the node to which i had originally forwarded the query). Let $N(i)$ be the set of all neighbors of i. Then, the weights at i are updated as follows:

$$\begin{aligned} W_{i,j}(t+1) &= W_{i,j}(t) \cdot (1 - L) \quad , \forall j \in N(i) \backslash h_i \\ W_{i,h_i}(t+1) &= W_{i,h_i}(t) + L \cdot \sum_{j \in N(i) \backslash h_i} W_{i,j}(t) \end{aligned} \tag{2}$$

Note that this update rule ensures that the weight of all the neighboring nodes always sums up to 1. We refer to this weight update as the learning process.

In case of an object that may be potentially located at one of multiple locations, each with different probabilities, over time, the query will learn to optimize its path to visit all of these locations in turn so that it minimizes the expected number of hops in the query.

to the query packet if shorter latency constraints are required, but this may result in query failure. Backtracking does add additional overhead to LEQS, but our simulations confirm that backtracking occurs less frequently as the learning proceeds.

Note that the query table at each node includes a row for each identifiable object with a column for each neighbor. Thus the storage requirement per node is $O(kT)$, if k is the (max) number of neighbors per node and T the number of objects being queried.

2.3 Metrics

We briefly discuss different metrics that can be used to measure the performance of LEQS. Metrics A and B are shown in simulation results presented in this paper.

A. Average Total Number of Hops (in path from object to sink): This is the expected number of hops that a query response takes from the located object to the sink. (i.e. the number of hops on the reverses path). This can be measured instantaneously for each query, or cumulatively averaged over all preceding queries (which will amortize the higher cost of initially inefficient paths over future, more efficient queries). Note that this one-way metric does not take into account additional transmissions due to branching/backtracks during the query forwarding phase. Those are best captured by the following metric.

B. Average Number of Transmissions: A related metric is the average number of transmissions required to forward the query. Since query responses are sent in reverse of the original query, the two are almost the same. The difference is that there can be additional overhead of up to 50% in the forward direction due to the random walk branches in initial queries that result in backtracks (to prevent query looping). But the number of such backtracks decreases over time so that eventually the two metrics A and B become identical.

C. Convergence Time: This is the time needed for the network to converge to a weight distribution that does not change (which happens when queries settle to a single possible path). In our simulations, we measure time by the number of queries issued by the sink for the object. This metric is useful in theory, but not explicitly presented in our simulations.

3 Simulation Experiments and Results

We uniformly place 121 static nodes in a 1 x 1 square area. All nodes in the network have the same communication range R. The sink is located at the left lower corner[4]. All figures shown for the cumulative average number of hops are averaged over 20 simulation runs.

Figure 3 depicts a single LEQS run for the first, 20th, and 50th queries in case of a single object location. Starting from using random walk (the first query), the network learns through time about the distribution of object locations. The 20th query clearly performs better than the first query. The 50th query traverses the optimal path.

Our initial approach for studying the learning process was to change weights without considering the distance to the object, in other words, setting $\alpha = 0$. Figure 4(left column) shows the results for this approach. As we can see from the figure, if we change weights fast (for example, p=0.9), we will have a solution fast but that may not be optimal. On the other hand, if we change weights slowly (for example, p=0.1), we will have a solution after a long time but the solution is likely to be near-optimal.

Our second approach is to incorporate the distance from the point where the query was terminated into consideration when changing weights. Figure 4 (right column) shows the results for this approach. As we can see from the figure, if we change weights fast, we will have a solution fast and the solution is reasonably good. Comparing these two approaches, our second approach has a significant improvement in terms of the

[4]Note that while we use a single sink in our experiments, this is not at all a requirement for LEQS — it works equally well with multiple sinks issuing queries for the same identifiable object/target.

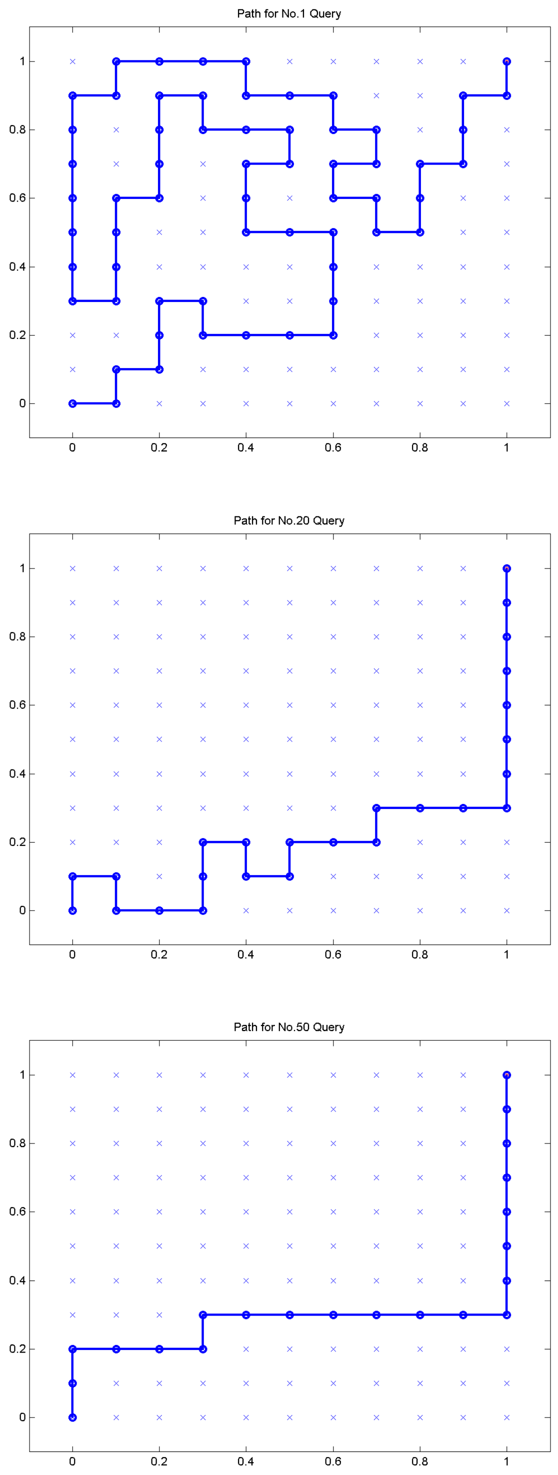


Figure 3: Sample run showing how learning improves the query efficiency (single location case) over multiple queries.

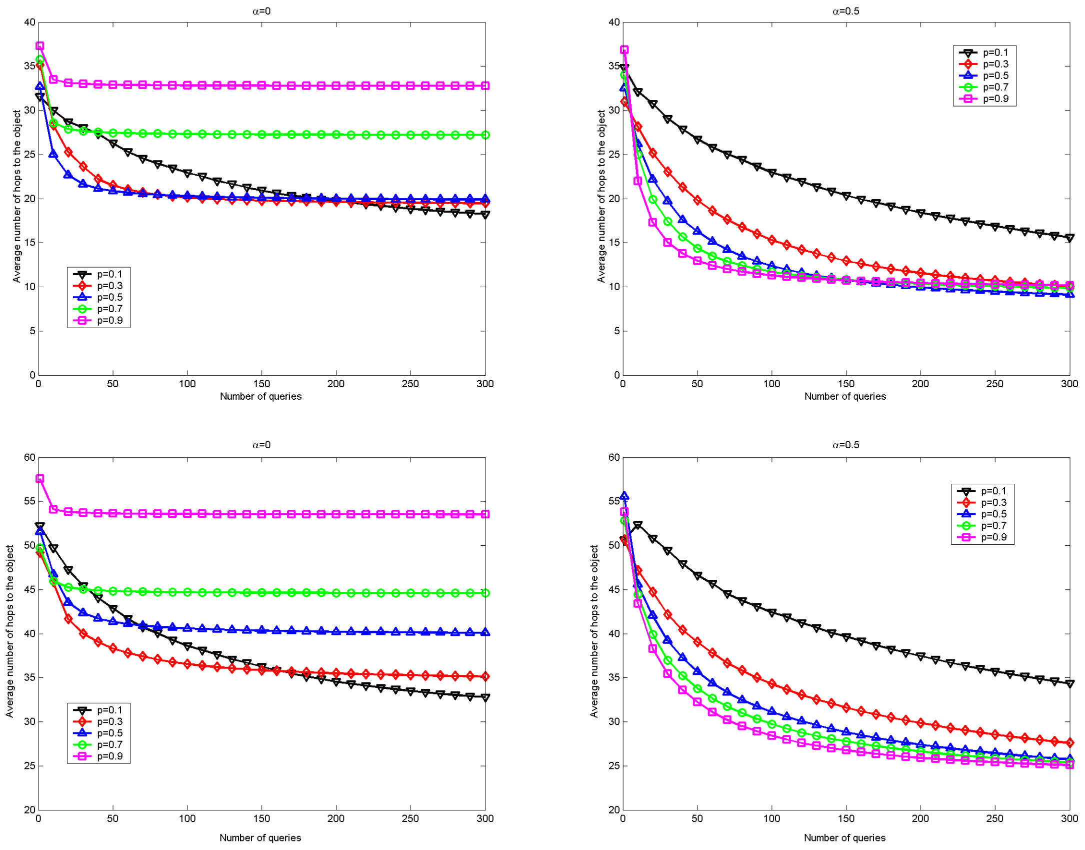


Figure 4: Decrease in average cost of querying over time for different learning rates for $\alpha = 0$ (left) and $\alpha = 0.5$ (right) for random deployment (top row) and grid deployment (bottom row)

solution quality except when the learning parameter p is too small. We will further examine the impact of α in detail below.

Figure 4 also confirms that the trends are similar regardless of whether the deployment is random or on a grid. Due to the similarity in their results, we restrict our remaining results to scenarios involving grid deployment.

Figure 5 shows that the performance curves for different settings of p can cross-over. If p is chosen to be very high (0.95), LEQS performs well initially as the learning rate is higher; however it may settle down to a non-optimal value, as seen by the fact that the lower learning rate curve eventually provides a lower average cost. This suggests that there is a tradeoff between the speed of convergence and the optimality of the converged solution that can be achieved by a careful selection of this parameter p.

Intuitively, if the average number of neighbors for a node is large, the probability of choosing the right neighbor to the object is small. Therefore it will take time to have a solution and that solution may be a bad solution. Figure 6 gives the results for different network densities/communication radius for both single and multiple location scenarios. As we can see, LEQS works well when the density is low. In general, however, as the radio range increases, initially the costs increases but then it starts to decrease when the densities are so high that there are very few hops between any node and an object location.

Figure 7 shows the effect of the learning factor p at different query number (times). We fix α at 0.5 in this experiment. The line for the 50^{th} query gives a measurement of the short-term solution quality while the line

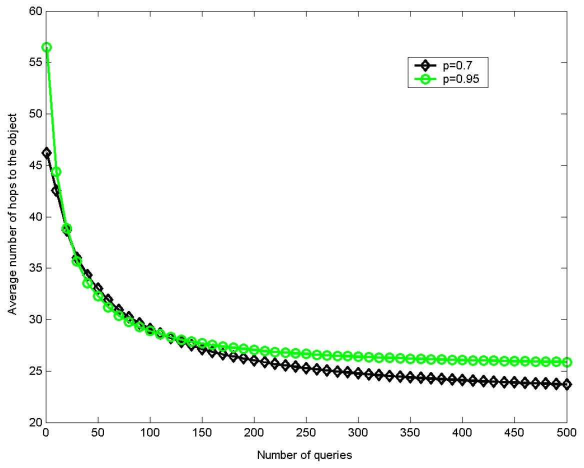


Figure 5: Performance of Query Learning for different learning rates; the crossover shows that for lower values of p the learning may be slower but converge to a better value than with high p.

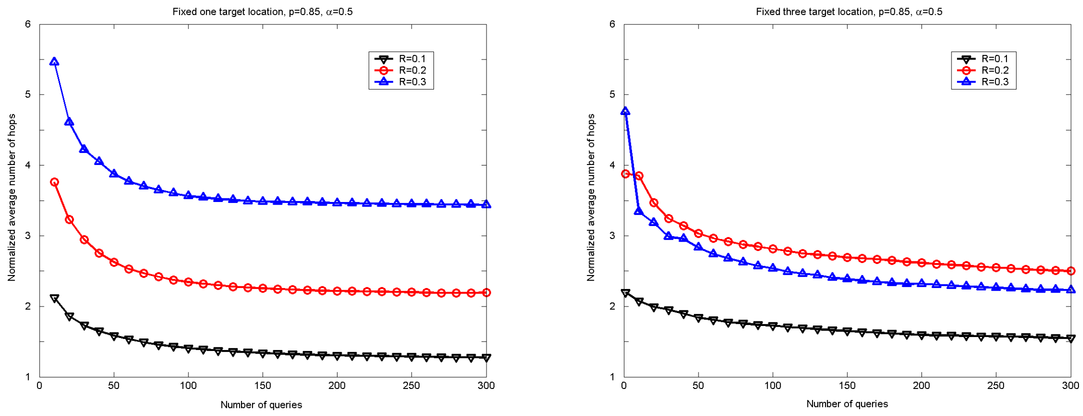


Figure 6: Comparison of Query learning with different radio ranges for single location (left) and three-location scenarios.

for query number 300 gives a measurement of the long-run solution quality. From this figure, we find that for our experimental settings when p equals 0.85, it yields good solutions for both short-term and long-term performance. This kind of analysis helps identify optimal parameter settings for LEQS on a given sensor network topology.

Figure 8 shows the effect of the learning factor α at different times (query numbers). We fix p at 0.85 in this experiment according to the previous experiment. The line for query number 50 gives a measurement of the short-term solution quality while the line for query number 300 gives a measurement of the long-run solution quality. From this figure, we find that when α equals about 0.45 or 0.5, it yields good solutions both in the short-term and in the long-term.

We now consider what happens if the object is not always located at one node, but rather in one of several locations with an underlying probability distribution. Figure 9 shows the experiment results for different number (k) of possible object locations. In this experiment, we set the object locations along the diagonal line of the square area with equal probabilities. (After careful calculation, we find that the optimal number of hops should be 10 for k=1 and 20 for all other cases.) From this figure, we can see that it will take much

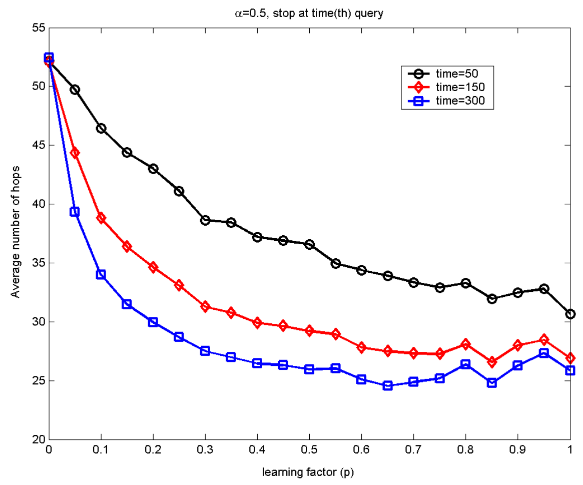


Figure 7: Performance (for different numbers of queries) as a function of the learning parameter p.

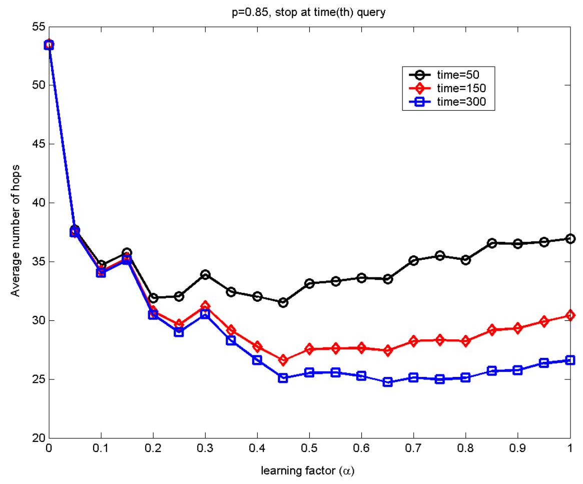


Figure 8: Performance (different numbers of queries) as a function of the learning parameter α

longer for the network to learn the distribution of the object locations when the number of possible object locations is large. Here we fix p at 0.85 and α at 0.45.

As illustration of the performance of LEQS for multiple location scenarios, we examine some specific examples. Figure 10 shows the results for a three-possible-object-locations case: showing both the learned improvement over time and a sample run after 300 queries. We can see that the object is located at the lower right corner with a probability of 0.6, at the center with a probability of 0.3, and at the upper left corner with a probability of 0.1.

Similarly figure 11 shows the results for a five-object-locations case. We can see the distribution of the five object locations with the corresponding probabilities. The figure also depicts a sample run for the 1000th query, when the object is located at (0.8,0).

Figure 12 compares the performance of LEQS with respect to random walk, flooding, and the global optimum solution for both single and 5-location scenarios. The performance of LEQS is shown for both the cumulative average (which takes into account the cost of the initial inefficient queries) as well as the instantaneous value

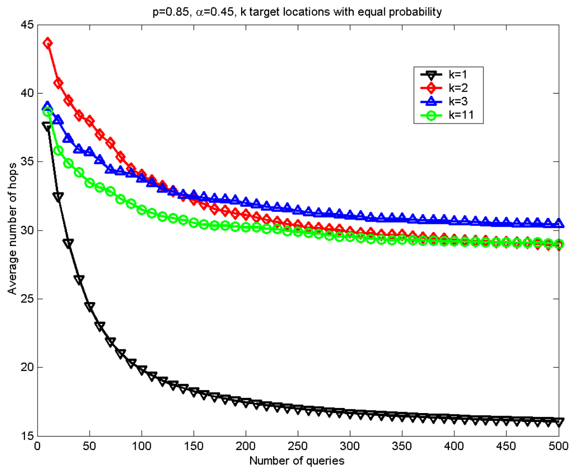


Figure 9: Performance of query learning for different numbers of possible locations for the queried object

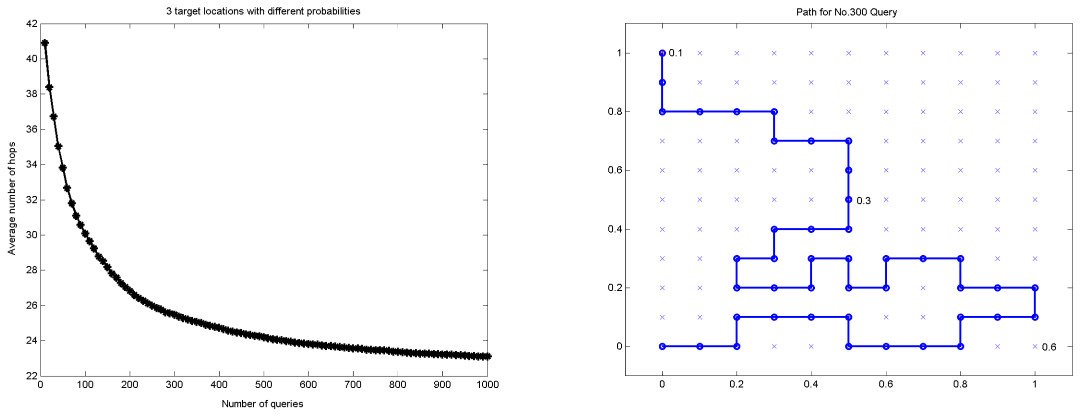


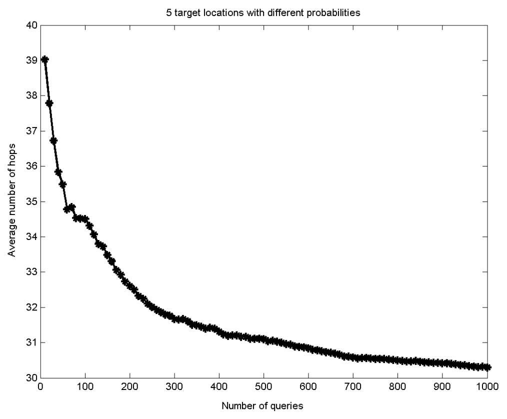
Figure 10: Performance of Query Learning (left) and a sample run after 300 queries for scenario involving three possible locations with different probabilities.

of the number of transmissions taken to reach the object. The global optimum shown is the optimal sequential solution — this is obtained by performing an exhaustive search that evaluates paths involving visiting each location (all permutations).

In the case of a single object location, the performance of LEQS is near-optimal, and shows that it offers nearly 75% gains with respect to a random walk. In the five-location case, it takes longer to converge to the optimal solution (this has not yet occurred after 1000 queries), but the performance of LEQS still shows more than 50% improvements. Note that these gains can be even higher for larger networks.

In cases where there is a node failure, our scheme has the ability to be self-healing. As long as there exists a path between the sink and the object, the network will find it and adapt to it. Figure 13 shows the no-failure cases as well as a scenario in which there is a node failure (node 26) at query number 50. Although there is a jump at query number 50 in the figure (seen in both the instantaneous and the average curves) the learning process continues after the failure, and the number of hops to success still keeps decreasing towards the optimal value as the number of queries increase.

Note that although we have not explicitly compared LEQS with flooding or random walks in this paper



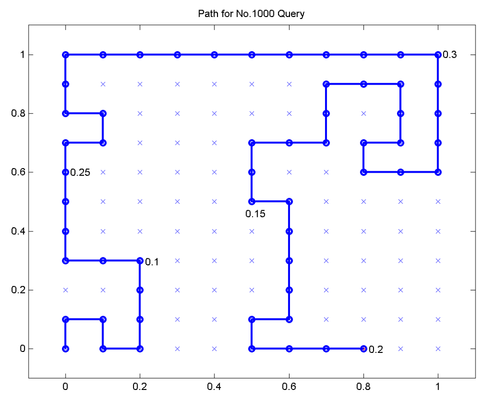
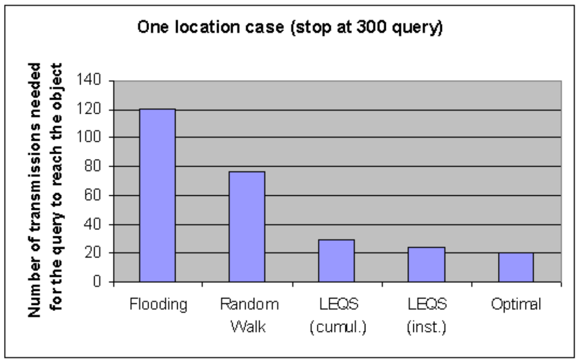


Figure 11: Performance of Query Learning (left) and a sample run after 1000 queries for scenario involving five possible locations with different probabilities.



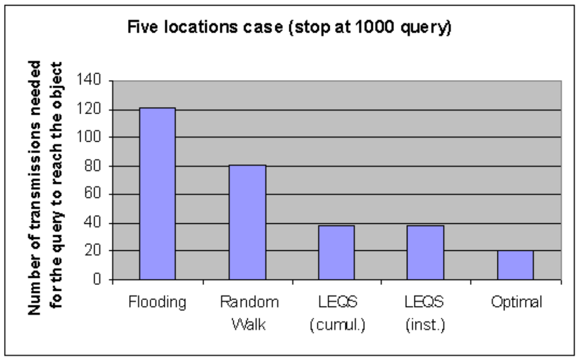


Figure 12: Comparison of LEQS, both cumulative (which amortizes initial cost of learning) and instantaneous, with respect to Random Walk, Flooding and the global Optimal solution, for one-location (left) and five-location scenarios (right)

these comparisons are implicit – flooding in this network has a cost of $n = 121$ transmissions. A random walk may take about $n/2 \approx 60$ hops to locate the object, and this cost is shown implicitly as the cost of LEQS at query number 1, when no learning has taken place. In the scenarios we considered in this section, we found that LEQS can result in energy efficiency improvements of up to 75% given sufficient learning time and optimal parameter settings.

4 Related Work

The Directed diffusion protocol [1] is a data-centric communication protocol for sensor networks. In this protocol, the sink first sends out a query in the form of an interest which is flooded throughout the network; gradients are set up along with the interest propagation and data is sent to the sink from the source along the interest gradient path. Reinforcement is used for the path from the sources to the sink with short latency. The gradients updates in directed diffusion bear some similarity to the weight updates in LEQS but it is important to note critical differences: The weight in LEQS indicates the probability with which a query (executing a random walk) is forwarded to the neighbors on its way *from the sink*, whereas gradients in directed diffusion affect the rate at which response data is forwarded from the source *to the sink*. Again, the querying in directed diffusion is performed using flooding and no gradient update occurs to improve the

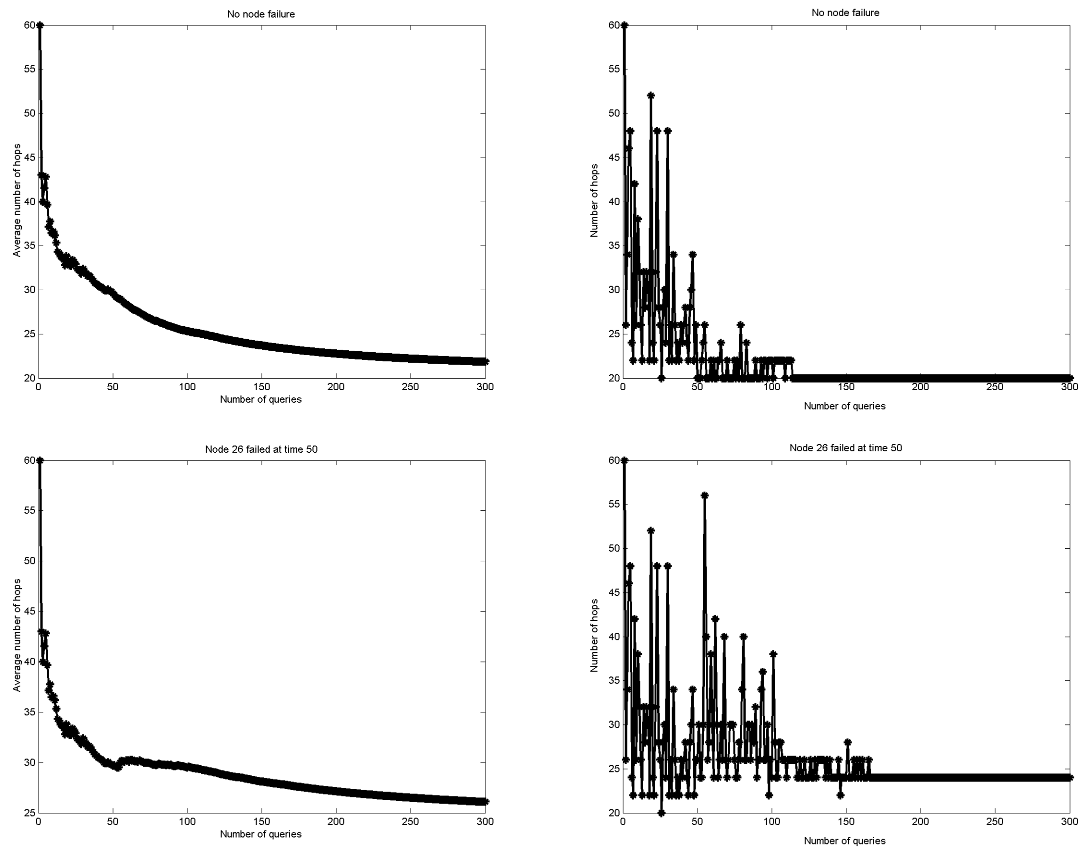


Figure 13: Average (left column) and instantaneous (right column) query cost with respect to query number without (top row) and with (bottom row) a node failure on the query path at query number 50, demonstrating the self-healing nature of LEQS.

querying process over multiple interests/queries as in LEQS.

The flooding of queries in Directed Diffusion is efficient only if the flow that is set up in response to the interest flood is long-standing. In such a case, the flooding cost for the query can be amortized over time. Thus, while LEQS is primarily a search-like query mechanism for one-shot queries, Directed Diffusion is a full-fledged routing mechanism for establishing long-standing source-sink flows.

Paper [2] describes the Rumor routing mechanism for the application where the queried data is small. In this mechanism, mobile agents are sent out for every detected event to create event paths. Queries are forwarded randomly to the source. If a query intersects with an event path, the route between the event location and the querying node is established. The authors also show that the probability that a query intersects with an event path depends on the number of agents created and some other factors. This Rumor routing mechanism has a lower querying cost in comparison to Directed Diffusion. However, if there is no query generated for a detected event, the energy spent on the agents for that event may be wasted.

ACQUIRE [3] is an efficient mechanism for obtaining information in sensor networks. It forwards an active query through the network, and each intermediate node uses its cached local information (within a look-head of d hops) to solve or partially solve the query. If the query is not fully solved, it will be forwarded to a node that is d hops away in a random manner or directed intelligently based on other information. The authors use a random manner in their analysis. Our learned querying mechanism can be easily incorporated with ACQUIRE to achieve a better network performance.

We should also point out that there are some parallels between and work on sequential paging in cellular telephone networks. In sequential paging problems, the objects being queried for are mobile users in the network. If the mobile users have an underlying mobility pattern that can be represented probabilistically, sequential paging algorithms have been developed to query for these users with minimum cost [7]. While the concept of querying for objects with underlying distribution is similar to the assumption in LEQS, there are some key differences. The primary difference is that proposals for cellular networks are all based on techniques where location probabilities are stored and decisions are made in a centralized manner, which is not a feasible or scalable technique for sensor networks; further, to our knowledge there has been little prior work on paging based on learning, with the exception of the LeZi update scheme [8].

5 Conclusions

We develop a learning-based efficient querying mechanism (LEQS) that is simple, localized and distributed. LEQS is suitable for one-shot queries for objects with underlying location patterns and in a context where the object is queried for repeatedly. In this algorithm, sensor nodes maintain weights indicating the probability with which a given query is forwarded to each neighbor. The query response is used to update these weights on the reverse-path, effectively training the network to locate the object efficiently.

Our extensive simulation results show that this in-network learning-based query scheme can significantly reduce the energy expenditure of querying over time (between 50 and 75% savings in some of the scenarios we studied compared to random walks without learning). We examined the critical learning parameters such as p and α which can be chosen to optimize the tradeoff between learning rate and the quality of the learned solution. We also studied the impact of network density by varying the radio range.

Our results demonstrates that LEQS is suitable for scenarios where the queried object may be located in many possible locations with different probabilities. Finally, the learning process has been shown to be robust and self-healing in case of node failures.

LEQS has several desirable features in the context of sensor networks. It involves only local interactions and makes minimal assumptions about the network; in particular, there is no need for global unique identifiers for nodes or geographical information or even an underlying routing mechanism for querying (though some of these may be needed in the application-level content of the response). It also works when there are multiple sinks querying for the same object. Finally, we should note that the in-network storage per node required by LEQS also scales very well — it is linear in both the number of neighbors and the number of queried objects.

For future work, we would like to see LEQS tested in a more realistic environment such as an experimental sensor test-bed. Other possible extensions include performing mathematical analysis to better understand the optimal settings of the learning parameters α and p and network settings such as the density and query rate on the performance of LEQS. Also of interest would be to test LEQS on dynamic scenarios in which the underlying distributions of the object location can change over time (intuitively, the learning mechanism in our proposed algorithm should be able to adapt to such changes). We would also like to develop and compare alternative update rules for learning in a systematic framework.

References

[1] C. Intanagonwiwat, R. Govindan and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," *MOBICOM'00*, August 2000.

[2] D. Braginsky and D. Estrin, "Rumor Routing Algorithm for Sensor Networks," *First ACM Workshop on Sensor Networks and Applications (WSNA)*, September 2002.

[3] N. Sadagopan, B. Krishnamachari, and A. Helmy, "The ACQUIRE Mechanism for Efficient Querying in Sensor Networks," *First IEEE International Workshop on Sensor Network Protocols and Applications (SNPA), in conjunction with IEEE ICC 2003*, Anchorage, AK, May 2003.

[4] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "GHT – A Geographic Hash-Table for Data-Centric Storage," *First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, September 2002.

[5] R. Govindan, J. Hellerstein, W. Hong, S. Madden, M. Franklin, and S. Shenker, "The Sensor Network as a Database," *Technical Report 02-771, Computer Science Department, University of Southern California*, September 2002.

[6] Y. Yao and J. Gehrke, "The Cougar Approach to In-Network Query Processing in Sensor Networks," *SIGMOD Record*, Volume 31, Number 3, September 2002.

[7] B. Krishnamachari, R.-H. Gau, S. B. Wicker, and Z. J. Haas, "Optimal Sequential Paging in Cellular Networks," to appear in *ACM/Baltzer Wireless Networks*, 2003.

[8] A. Bhattacharya and S. K. Das, "LeZi-Update: An Information-theoretic Approach to Track Mobile Users in PCS Networks," *Mobicom '99*, August 1999.