Active Query Forwarding in Sensor Networks (ACQUIRE)

Narayanan Sadagopan[†], Bhaskar Krishnamachari[§†], Ahmed Helmy[§]
[†] Department of Computer Science
[§]Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California
{narayans, bkrishna, helmy}@usc.edu

June 11, 2003

Abstract

While sensor networks are going to be deployed in diverse application specific contexts, one unifying view is to treat them essentially as distributed databases. The simplest mechanism to obtain information from this kind of a database is to flood queries for named data within the network and obtain the relevant responses from sources. However, if the queries are a) complex, b) one-shot, and c) for replicated data, this simple approach can be highly inefficient. In the context of energy-starved sensor networks, alternative strategies need to be examined for such queries.

We propose a novel and efficient mechanism for obtaining information in sensor networks which we refer to as ACQUIRE. The basic principle behind ACQUIRE is to consider the query as an active entity that is forwarded through the network (either randomly or in some directed manner) in search of the solution. ACQUIRE also incorporates a look-ahead parameter d in the following manner: intermediate nodes that handle the active query use information from all nodes within d hops in order to partially resolve the query. When the active query is fully resolved, a completed response is sent directly back to the querying node.

We take a mathematical modelling approach in this paper to calculate the energy costs associated with ACQUIRE. The models permit us to characterize analytically the impact of critical parameters, and compare the performance of ACQUIRE with respect to other schemes such as flooding-based querying (FBQ) and expanding ring search (ERS), in terms of energy usage, response latency and storage requirements. We show that with optimal parameter settings, depending on the update frequency, ACQUIRE obtains order of magnitude reduction over FBQ and potentially 60 to 85% energy reduction over ERS (in highly dynamic environments and high query rates). We show that these energy savings are provided in trade for increased response latency. The mathematical analysis is validated through extensive simulations.

1 Introduction

Wireless sensor networks are envisioned to consist of large numbers of devices, each capable of some limited computation, communication and sensing, operating in an unattended mode. These networks are intended for a broad range of environmental sensing applications from weather data-collection to vehicle tracking and habitat monitoring [2, 3, 4]. The key challenge in these unattended networks is dealing with the limited energy resources on the nodes.

With a small set of independent sensors it is possible to collect all measurements from each device to a central warehouse and perform data-processing centrally. However, with large-scale networks of energy-constrained sensors this is not a scalable approach. It has been argued that it is best to view such sensor networks as distributed databases [9, 10, 11, 16]. There may be a central querier/data sink (or a collection of queriers/sinks) which issues queries that the network can respond to. Due to energy constraints it is desirable for much of the data processing to be done in-network. This has led to the concept of *data-centric* information routing, in which the queries and responses are for named data as opposed to the end-to-end address-centric routing performed in traditional networks.

Depending on the applications, there are likely to be different kinds of queries in these sensor networks. The types of queries can be categorized in many ways, for example:

- *Continuous queries*, which result in extended data flows (e.g. "Report the measured temperature for the next 7 days with a frequency of 1 measurement per hour") versus *One-shot queries*, which have a simple response (e.g. "Is the current temperature higher than 70 degrees?")
- *Aggregate queries*, which require the aggregation of information from several sources (e.g. "Report the calculated average temperature of all nodes in region X") versus *Non-aggregate queries* which can be responded to by a single node (e.g. "What is the temperature measured by node x?")
- *Complex queries*, which consist of several sub-queries that are combined by conjunctions or disjunctions in an arbitrary manner (e.g. "What are the values of the following variables: X, Y, Z?" or "What is the value of (X AND Y) OR (Z)" versus *simple queries*, which have no sub-queries (e.g. "What is the value of the variable X?") [1]
- *Queries for replicated data*, in which the response to a given query can be provided by many nodes (e.g. "Has a target been observed anywhere in the area?") and *queries for unique data*, in which the response to a given query can be provided only by one node.

Flooding-based query mechanisms such as the Directed Diffusion data-centric routing scheme [5] are well-suited for continuous, aggregate queries. This is because the cost of the initial flooding of the interest can be amortized over the duration of the continuous flow from

[1]We assume that each sub-query is a query for some variable tracked by the sensor network.

the source(s) to sink(s). However, keeping in mind the severe energy constraints in sensor networks, a one-size-fits-all approach is unlikely to provide efficient solutions for other types of queries.

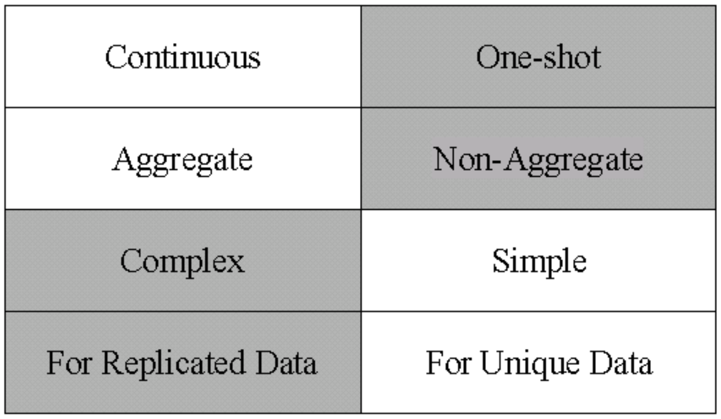
In this paper we propose a new data-centric querying mechanism, ACtive QUery forwarding In sensoR nEtworks (ACQUIRE). Figure 1 shows the different categories of queries and the kinds of queries in sensor networks that ACQUIRE is well-suited for: one-shot, complex queries for replicated data. As a motivation for ACQUIRE, we describe two scenarios which involve such queries:

- **Bird Habitat Monitoring Scenario:** Imagine a network of acoustic sensors deployed in a wildlife reserve. The processor associated with each node is capable of analyzing and identifying bird-calls. Assume each node stores any bird-calls heard previously. The task "obtain sample calls for the following birds in the reserve: Blue Jay, Nightingale, Cardinal, Warbler" is a good example of a *complex* (because information is being requested about four birds), *one-shot* (because each sub-query can be answered based on stored and current data) query, and is *for replicated data* (since many nodes in the network are expected to have information on such birds). Another example of a complex, one-shot query in this network might be "return 5 locations where a Warbler's call has been recorded" (the request for each location is a sub-query).
- **Micro-Climate Data Collection Scenario:** Imagine a heterogeneous network consisting of temperature sensors, humidity sensors, wind sensors, rain sensors, vibration sensors etc. monitoring the micro-climate in some deployed area. It is possible to put together a number of separate basic queries such as "Give one location where the temperature is greater than 80 degrees?", "Give one location where there is rain at the moment in the area?", and "Give one location where the wind conditions are currently greater than 20 mph?" can be combined together into a single batched query. This *complex* query is *one-shot* (as it asks only for current data) and is also *for replicated data* (since several nodes in the network may be able to answer the queries[2]).

The principle behind ACQUIRE is to inject an active query packet into the network that follows a random (possibly even pre-determined or guided) trajectory through the network. At each step, the node which receives the active query performs a triggered, on-demand, update to obtain information from all neighbors within a look-ahead of d hops. As this active query progresses through the network it gets progressively resolved into smaller and smaller components until it is completely solved and is returned back to the querying node as a completed response.

While most prior work in this area has relied on simulations in order to test and validate data-querying techniques, we have taken here a mathematical modelling approach that allows us

[2]There is an implicit assumption that the sub-queries can all be resolved. In this example, it is assumed that there are such locations. Without this assumption, it is not be possible to do anything more intelligent than querying all nodes in the network.



Continuous	▨ One-shot
Aggregate	▨ Non-Aggregate
▨ Complex	Simple
▨ For Replicated Data	For Unique Data

Figure 1: A categorization of queries in sensor networks: the shaded boxes represent the query categories for which the ACQUIRE mechanism is well-suited.

to derive analytical expressions for the energy costs associated with ACQUIRE and compare it with other mechanisms, and to study rigorously the impact of various parameters such as the value of the look-ahead parameter and the ratio of update rate to query rate. Our mathematical analysis is validated through simulations.

The rest of the paper is organized as follows: in section 2 we describe some of the related work in the literature. We provide a basic description of the ACQUIRE mechanism in section 3. In section 4 we develop our mathematical model for ACQUIRE and derive expressions for the energy cost involved as a function of the number of queried variables, the look-ahead parameter, and the ratio of the refresh rate to the query rate. We develop similar models and energy cost expressions for two alternative mechanisms: flooding based queries (FBQ) and expanding ring search (ERS) in section 5. We first examine the impact of critical parameters on the energy cost of ACQUIRE and then compare it to the alternative approaches in section 6. Our analytical models are validated by simulations in section 7. The average response latency incurred by ACQUIRE, ERS and FBQ is analytically modelled in section 8, while the caching storage requirements are discussed in section 9. We discuss these results and describe the future work suggested we are planning to undertake in section 10. Finally, we present our concluding comments in section 11.

2 Related Work

Bonnet, Gehrke, and Seshadri [10, 11] as well as Yao and Gehrke [16] present the COUGAR approach which treats sensor networks as distributed databases, with users tasking the network with declarative queries which are then converted by a front-end query processor into an efficient query plan for in-network processing. Similarly Govindan, Hellerstein, Hong *et al.* also argue in [9] that sensor networks ought to be viewed primarily as virtual databases, with query optimization performed via data-centric routing mechanisms within the network.

The efficient in-network computation of aggregate responses to queries is the subject of the paper by Madden, Szewcyk *et al.* [19]. The ACQUIRE mechanism we describe in this paper is compatible with this database perspective, and can viewed as a data-centric routing mechanism that provides superior query optimization for responding to particular kinds of queries: complex, one-shot queries for duplicated data.

Intanagonwiwat, Govindan, Estrin and Heidemann propose and study Directed Diffusion [5, 6], a data-centric protocol that is particularly useful for responding to long-standing/continuous queries. In Directed Diffusion, an interest for named data is first distributed through the network via flooding (although optimizations are possible for geographically localized queries), and the sources with relevant data respond with the appropriate information stream. The impact of aggregation in improving the energy costs of such data-centric protocols is examined by Krishnamachari, Estrin and Wicker in [7].

Also related to our work are the Information Driven Sensor Querying (IDSQ) and Constrained Anisotropic Diffusion Routing (CADR) mechanisms proposed by Chu, Hausseker and Zhao [14]. In IDSQ, the sensors are selectively queried about correlated information based on a criterion that combines information gain and communication cost, while CADR routes a query to its optimal destination by following an information gain gradient through the sensor network.

One technique that is close in spirit to ACQUIRE is the rumor-routing mechanism proposed recently by Braginsky and Estrin in [21]. Their approach is quite interesting - sources with events launch mobile agents which execute random walks in the network resulting in event-paths. The queries issued by the querier/sink, in a manner somewhat similar to ACQUIRE, are also mobile agents that follow random walks. Whenever a query agent intersects with an event-path, it uses that information to efficiently route itself to the location of the event. Rumor routing is a mechanism to lower the interest-flooding cost for Directed Diffusion in situations where geographical information may not be available. Rumor routing is not, however, geared primarily towards complex one-shot queries for replicated data (as ACQUIRE is) and does not incorporate any look-ahead/update parameters. Moreover, if data is replicated, there might be multiple sources, each of which might initiate a random walk in the rumour-routing case. In such cases, rumour-routing may not be energy-efficient.

Other data-centric routing protocols proposed for sensor networks include SPIN for data dissemination by Heinzelman, Kulik and Balakrishnan [17], and LEACH for data collection by Heinzelman, Chandrakasan and Balakrishnan [18].

The recent work by Ratnasamy, Karp *et al.* [13] presents the geographic hash table technique for data-centric storage (DCS) in sensor networks. This approach is particularly useful for storing information to deal with historic queries (i.e. queries for non-current data). In estimating the cost of local storage the authors of [13] assume the use of flooding-based queries, to which we provide an alternative in this paper. It is also be possible to conceive of using our ACQUIRE scheme in conjunction with any DCS techniques that result in replication (e.g. for robustness reasons).

Our work also has some similarities to techniques proposed for searching in unstructured peer-to-peer (P2P) overlay networks on the Internet. In particular, [22] discusses the possibility of launching k-random walks through the unstructured P2P network for discovering required files/data. This differs from our work in three respects: one is that the cost-model is different in the two scenarios - in P2P networks one is primarily concerned with minimizing bandwidth usage and delay while we are primarily concerned with minimizing energy consumption; the second is that we incorporate the look-ahead parameter and allow for complex queries, which, as we show in this paper, significantly improves the performance of such a search; and finally, the trajectories followed by active queries in ACQUIRE need not necessarily be random walks, they could be directed and deterministically selected.

Our ACQUIRE mechanism combines a trajectory for active queries with a localized update mechanism whereby each node on the path utilizes information about all the nodes within a look-ahead of d hops. The size of this look-ahead parameter effects a tradeoff between the information obtained (which helps reduce the length of the overall trajectory) and the cost for obtaining the information. This look-ahead region is somewhat similar in spirit to the notion of zones in the Zone Routing Protocol (ZRP) [15] and to the notion of neighborhoods in the Contact-based Architecture for Resource Discovery (CARD) [12] developed for mobile ad-hoc networks. One key difference is that in ACQUIRE, only nodes on the active query trajectory need to have this look-ahead information and the neighborhood updates are triggered *on-demand*, if current information happens to be obsolete.

While the trajectory for the active queries is assumed to be random in our modelling in this paper, it is possible to envision pre-determined trajectories as well. One interesting new mechanism that we could combine ACQUIRE with is the idea of routing along curves, described by Nath and Niculescu in [20].

3 Basic Description of ACQUIRE

In order to explain ACQUIRE, it is best to begin first with an overview of traditional flooding-based query techniques. In these techniques, there is a clear distinction between the query dissemination and response gathering stages. The querier/sink first floods several copies of the query (which is an interest for named data). Nodes with the relevant data then respond. If it is not a continuous/persistent query (i.e. one that calls for data from sensors for an extended period of time as opposed to a single value), then the flooding can dominate the costs associated with querying. In the same way, even when data aggregation is employed, duplicate responses can result in suboptimal data collection in terms of energy costs.

By contrast, in ACQUIRE there are no distinct query/response stages. The querier issues an *active query* which can be a complex query, i.e. can consist of several sub-queries, each corresponding to a different variable/interest. The active query is forwarded step by step through a sequence of nodes. At each intermediate step, the node which is currently carrying the active query (the *active node*) utilizes updates received from all nodes within a lookahead of d hops in order to resolve the query partially. New updates are triggered reactively by the active node upon reception of the active query only if the current information it has is obsolete (i.e. if the last update occurred too long ago). After the active node has resolved the active query partially, i.e. after it has utilized its local knowledge to answer as much of the complex query as possible, it chooses a next node to forward this active query to. This choice may be done in a random manner (i.e. the active query executes a random walk) or directed intelligently based on other information, for example in such a way as to guarantee the maximum possible further resolution of the query. Thus as the active query proceeds through the network, it keeps getting "smaller" as pieces of it become resolved, until eventually it reaches an active node which is able to completely resolve the query, i.e. answer the last remaining piece of the original query. At this point, the active query becomes a *completed response* and is routed back directly (along either the reverse path or the shortest path) to the originating querier.

The difference between traditional querying techniques and ACQUIRE, and the lookahead scheme of ACQUIRE are illustrated in figures 2 and 3 respectively.

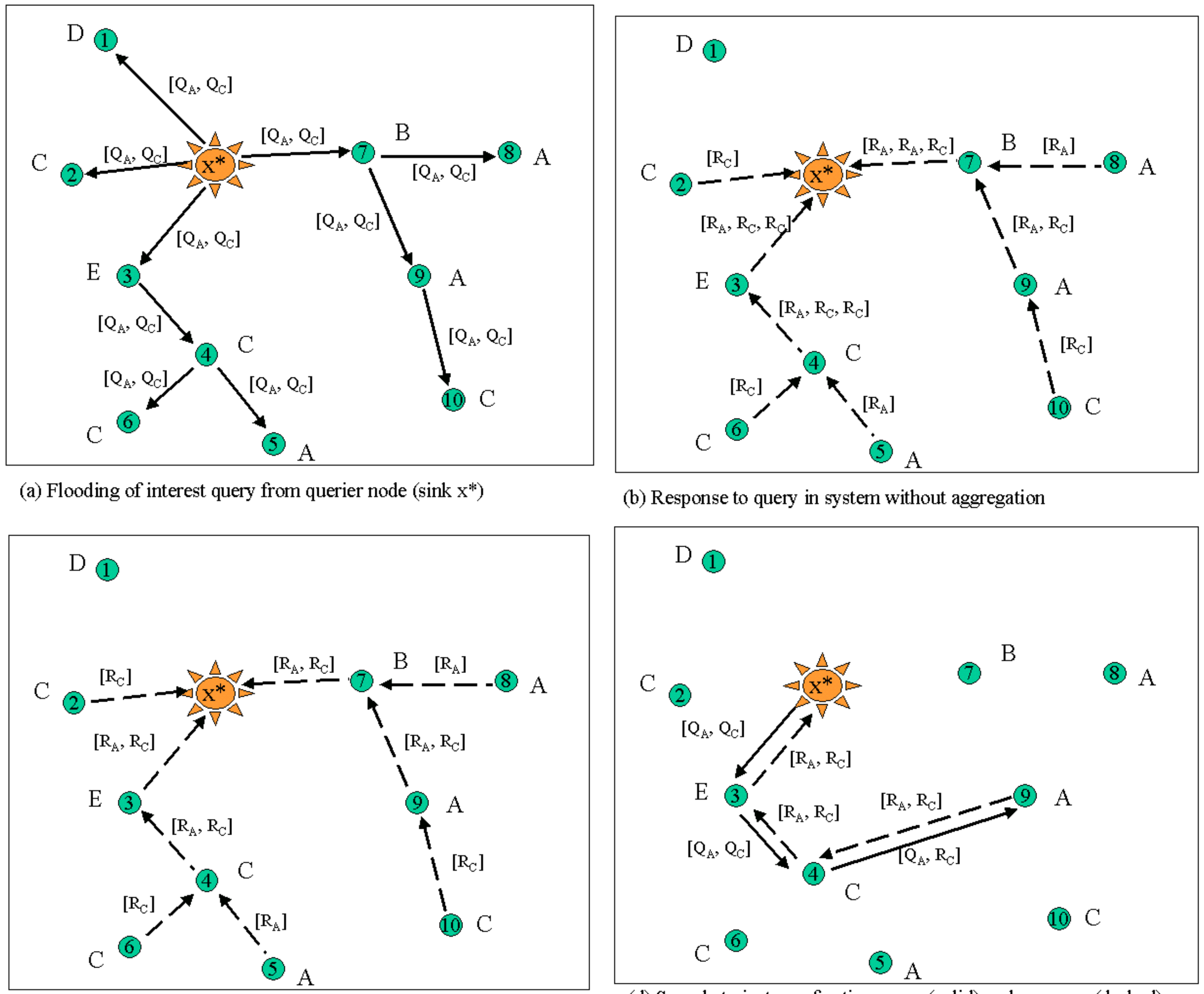
4 Analysis of ACQUIRE

We now build a mathematical model to analyze the performance of ACQUIRE in terms of its expected completion time and associated energy costs. This will also enable us to determine the optimal look-ahead parameter d. There are several metrics for energy costs. In our case, we focus on the number of transmissions as the metric for energy cost.

4.1 Basic Model and Notation

Consider the following scenario: A sensor network consists of X sensors. This network tracks the values of certain variables like temperature, air pressure, humidity, etc. Let $V = \{V_1, V_2, ...V_N\}$ be the N variables tracked by the network. Each sensor is equally likely to track any of these N variables. Assume that we are interested in finding the answer to a query $Q = \{Q_1, Q_2, ...Q_M\}$ consisting of M sub-queries, $1 < M \leq N$ and $\forall i : i \leq M, Q_i \in V$. Let S_M be the average number of steps taken to resolve a query consisting of M sub-queries. We define the number of steps as the number of nodes to which the query is forwarded before being completely resolved. Define d as the look-ahead parameter. Let the neighborhood of a sensor consist of all sensors within d hops away from it. In general the number of sensors in the neighborhood is dependent on the node density, the transmission range of the sensors, etc. However, we make the following assumptions about the sensor placement and their characteristics:

1. The sensors are laid out uniformly in a region.



(a) Flooding of interest query from querier node (sink x*)

(b) Response to query in system without aggregation

(c) Response to query in system with aggregation

(d) Sample trajectory of active query (solid) and response (dashed) in basic ACQUIRE (zero look-ahead)

Figure 2: Illustration of traditional flooding-based queries (a), (b), (c), and ACQUIRE (d) in a sample sensor network.

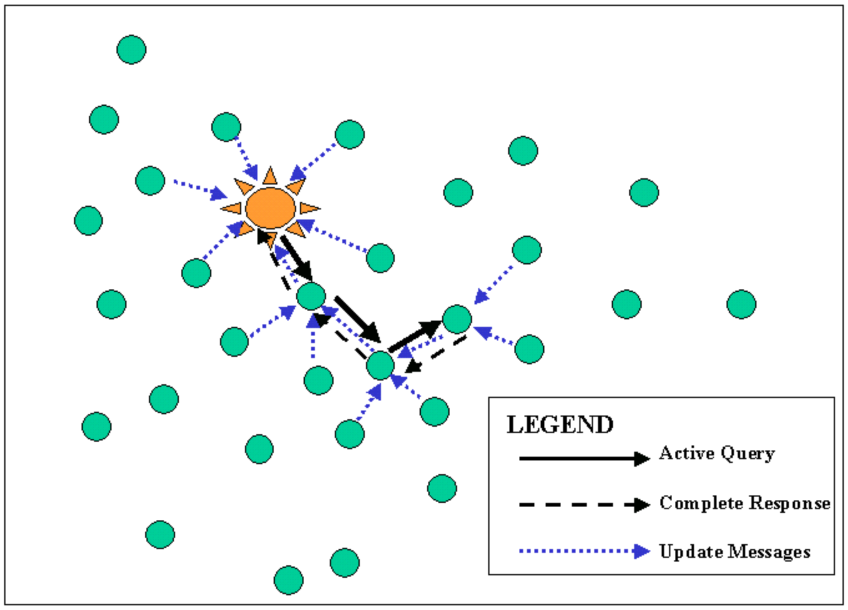


Figure 3: Illustration of ACQUIRE with a one-hop lookahead ($d = 1$). At each step of the active query propagation, the node carrying the active query employs knowledge gained due to the triggered updates from all nodes within d hops in order to partially resolve the query. As d becomes larger, the active query has to travel fewer steps on average, but this also raises the update costs. When d becomes extremely large, ACQUIRE starts to resemble traditional flooding-based querying.

2. All the sensors have the same transmission range.

3. The nodes are stationary and do not fail.

We model the size of a sensor's neighborhood (the number of nodes within d hops) as a function of d, $f(d)$, which is assumed to be independent of the particular node in question[3]. We also assume that all possible queries Q are resolvable by this network (i.e. can be responded to by at least one node in the network).

Mechanism of Query Forwarding:
Initially, let sensor x^* be the querier that issues a query Q consisting of M subqueries[4]. Let

[3]The size of the neighborhood is actually a measure of the number of different variables tracked by a node's neighborhood. For the sake of simplicity, we assume that each sensor tracks a single variable. However, this assumption does not affect the conclusions of our analysis

[4]One way to think of the size of the complex query M is to treat it as a "batch" parameter which effects a tradeoff between latency for batching and the latency and energy for query completion. Imagine independent sub-queries arrive at the central node at a fixed rate, then the time to put together a single batched query increases linearly with M, while the expected query completion time and energy increases only sub-linearly with M (as shown in section 4). In general the larger M is, the worse the batching latency, but better the average query completion time and energy expenditure (assuming subsequent queries are only sent out after the previous one has terminated).

d be the look-ahead parameter i.e each sensor can request information from sensors d hops away from it. In general when a sensor x gets a query it does the following:

1. *Local Update:* If its current information is not up-to-date, x sends a request to all sensors within d hops away. This request is forwarded hop by hop. The sensors who get the request will then forward their information to x. Let the energy consumed in this phase be E_{update}. Detailed analysis of E_{update} will be done in section 4.3.

2. *Forward:* After answering the query based on the information obtained, x then forwards the remaining query to a node that is chosen randomly from those d hops away.

Since the update is only triggered when the information is not fresh, it makes sense to try to quantify how often such updates will be triggered. We model this update frequency by an average *amortization factor* c, such that an update is likely to occur at any given node only once every $\frac{1}{c}$ queries. In other words the cost of the update at each node is amortized over $\frac{1}{c}$ queries, where $c \leq 1$. For example, if on average an update has to be done once every 100 queries, $c = 0.01$[5].

After the query is completely resolved, the last node which has the query returns the completed response[6] to the querier x^* along the reverse path [7]. We use α to denote the expected number of hops from the node where the query is completely resolved to x^*.

Let S_M be the average number of steps to answer a query of size M. Thus, the average energy consumed to answer a query of size M with look-ahead d can be expressed as follows:

$$E_{avg} = (cE_{update} + d)S_M + \alpha \tag{1}$$

Now, if $d = D$, where D is the diameter of the network, x^* can resolve the entire query in one step without forwarding it to any other node. However, in this case, E_{avg} will be considerably large. On the other hand, if d is too small, a larger number of steps S_M will be required. In general, S_M reduces with increasing d, while E_{update} increases with increasing d. It is therefore possible, depending on other parameters, that the optimal energy expenditure is incurred at some intermediate value of d. One of the main objectives of our analysis is to *analyze the impact of parameters such as M, N, c, and d upon the energy consumption E_{avg} of ACQUIRE.*

[5]It might be convenient to think of every datum having a time duration during which it is valid. During this period, all queries for the corresponding variable could be answered from the value cached from previous triggered updates. E.g. a sample bird call might have a longer validity period than a temperature reading.

[6]We note that it also makes sense to return partial responses back to the querier, as each sub-query is resolved along the way. This would reduce the energy and time costs of carrying partial responses along with the partial query. Our analysis thus overestimates the energy cost, and could be tightened further in this regard.

[7]If additional unicast or geographic routing information is available, the completed response can also be sent back along the shortest path back from the final node to the querier.

4.2 Steps to Query Completion

In this section, we present a simple analysis of the average number of steps to query completion as a function of M, N and $f(d)$. A more detailed analysis is in section 12 in the Appendix.

4.2.1 First-order Analysis

Consider the following experiment. Each sensor tracks a value chosen between 1 and N with equal probability. Fetching information from each sensor can be thought of as a trial. Define a "success" as the event of resolving any one of the remaining queries. Thus, if there are currently M queries to be resolved, then the probability of success in each trial is $p = \frac{M}{N}$ and the probability of failure is $q = \frac{N-M}{N}$. The number of trials till the first success i.e. the number of sensors from which information has to be fetched till one of the queries can be answered is a geometric random variable. Thus, the expected number of trials till the first success is $\frac{1}{p} = \frac{N}{M}$. Now the whole experiment can be repeated again with one less query. Thus, now, $p = \frac{M-1}{N}$ and $q = \frac{N-M+1}{N}$. The expected number of trials till the first success (i.e. another query being answered) is $\frac{N}{M-1}$ and so on.
Define the following:

1. σ_M = The number of trials till M successes i.e. the resolution of the entire query.
2. X_i: The number of trials (counted from the $(i-1)^{th}$ success) till the i^{th} success.

σ_M and X_i's are random variables.
Now,

$$\sigma_M = \sum_{i=1}^{M} X_i \tag{2}$$

By linearity of expectation,

$$E(\sigma_M) = \sum_{i=1}^{M} E(X_i) \tag{3}$$

$$E(\sigma_M) = N \sum_{i=1}^{M} \frac{1}{M-i+1} \tag{4}$$

Now, $\sum_{i=1}^{M} \frac{1}{M-i+1} = H(M)$ where $H(M)$ is the sum of the first M terms of the harmonic series.
It is known that $H(M) \approx \ln(M) + \gamma$, where $\gamma = 0.57721$ is the Euler's constant. Thus,

$$E(\sigma_M) \approx N(\ln M + \gamma) \tag{5}$$

Now, since we consider fetching information from $f(d)$ sensors as 1 trial (step) rather than $f(d)$ trials (steps)[8]:

$$S_M = \frac{E(\sigma_M)}{f(d)} \approx \frac{N(\ln M + \gamma)}{f(d)} \tag{6}$$

Eqn. 6 expresses the average number of steps to query completion (S_M) as a function of the total number of variables (N), the query size (M) and the neighborhood size ($f(d)$).

To answer more complex questions like "what is the probability that a complex query can be reduced in size in a single step?", we formulate the query forwarding process as a Markov Chain. Detailed analysis of this Markov Chain is in section 12 in the Appendix.

4.3 Local Update Cost

The energy spent in updating the information at each active node that is processing the active query E_{update} can be calculated as follows:
Assume that the query Q is at the active node x. Given a look-ahead value d, x can request information from sensors within d hops away. This request will be forwarded by all sensors within d hops except those that are exactly d hops away from x. Thus the number of transmissions needed to forward this request is the number of nodes within $d-1$ hops which is $f(d-1)$. The requested sensors will then forward their information to x. Now, the information of sensors 1 hop away will be transmitted once, 2 hops away will be transmitted twice,... d hops away will be transmitted d times. Thus,

$$E_{update} = (f(d-1) + \sum_{i=1}^{d} iN(i)) \tag{7}$$

where $N(i)$ is the number of nodes at hop i. $N(i)$ will be determined later in section 4.4.

4.4 Total Energy Cost

We make the assumption that each active node forwards the resolved query to another node that is exactly d hops away, requiring d transmissions. Hence the average energy spent in answering a query of size M is given as follows:

$$E_{avg} = (cE_{update} + d)S_M + \alpha \tag{8}$$

[8]Here, we make an assumption that $f(d)$ new nodes will be encountered at every node where the query is forwarded. However, due to overlap, the number of new nodes actually encountered might be a fraction of f(d) i.e. $(1-\delta)f(d)$, where $0 < \delta < 1$, is a measure of the average overlap of the neighborhoods of nodes handling the query. It depends on algorithm used to route the query. For ACQUIRE to perform efficiently, this overlap should be small.

where α is the expected number of hops from the node where the query is completely resolved to the querier x^*.[9] This is the cost of returning the completed response back to the querier node. This response can be returned along the reverse path in which case α can be atmost dS_M. Thus,

$$E_{avg} \quad = \quad (cE_{update} + 2d)S_M \tag{9}$$

4.4.1 Special Case: $d = 0$ - Random Walk

If the look-ahead $d = 0$, the node x will not request for updates from other nodes. x will try to resolve the query with the information it has, and will forward the query to a randomly chosen neighbor. Thus, in this case, ACQUIRE reduces to a random walk on the network. On an average it would take $E(\sigma_M)$ steps to resolve the query and $E(\sigma_M)$ steps to return the resolved query back to the querier x^*. Thus,

$$E_{avg} \quad = \quad 2E(\sigma_M) \tag{10}$$

4.5 Optimal Look-ahead

As mentioned in section 4.4,

$$\begin{aligned} E_{avg} &= \{(f(d-1) + \sum_{i=1}^{d} iN(i))c + 2d\}S_M && \text{(from Eqn.7 and 9)} \\ &\approx \{(f(d-1) + \sum_{i=1}^{d} iN(i))c + 2d\}\frac{N(\ln M + \gamma)}{f(d)} && \text{(from Eqn.6)} \end{aligned} \tag{11}$$

If we ignore boundary effects, it can be shown that $f(d) = (2d(d+1)) + 1$ for a grid[10] Also,

$$\begin{aligned} N(i) &= f(i) - f(i-1) \\ &= 2i(i+1) - 2(i-1)i \\ &= 4i \end{aligned} \tag{12}$$

i.e. the number of nodes exactly i hops away from a node x on a grid is $4i$. Thus,

$$E_{avg} \quad \approx \quad \{(2(d-1)(d) + 1 + \sum_{i=1}^{d} 4i^2)c + 2d\}\frac{N(\ln M + \gamma)}{(2(d)(d+1)) + 1}$$

[9]Here, we are actually over-estimating E_{avg} by an additive amount of S_M as the query will not be forwarded at the last step, but will be returned back to the querier.

[10]Here we assume that at every node handling the query, there are $f(d)$ new nodes available in the neighborhood i.e. the query is routed such that there is minimal overlap between the neighborhoods of nodes handling the query. We will see the effect of overlap in simulation results.

$$\begin{aligned}
&\approx \{(2(d-1)(d)+1+\frac{4}{6}(d)(d+1)(2d+1))c+2d\}\frac{N(\ln M+\gamma)}{(2(d)(d+1))+1} \\
&\approx \{\frac{cN(\ln M+\gamma)}{3}\frac{4d^3+12d^2-4d+3}{2d^2+2d+1}+N(\ln M+\gamma)\frac{2d}{2d^2+2d+1}\} \qquad (13)
\end{aligned}$$

To find the value of d (as a function of c, N and M) that minimizes the E_{avg}, we differentiate the above expression w.r.t. d and set the derivative to 0. We get the following:

$$\begin{aligned}
\frac{2}{3}\frac{(N\ln M+\gamma)(4cd^4+8cd^3+22cd^2+6cd-5c-6d^2+3)}{(2d^2+2d+1)^2} &= 0 \\
4cd^4+8cd^3+22cd^2+6cd-5c-6d^2+3 &= 0 \qquad (14)
\end{aligned}$$

Thus, the optimal look-ahead d^* depends on the amortization factor c and is independent of M and N.
If $d = 0$:

$$E_{avg} \approx 2N(\ln M+\gamma) \qquad \text{(from Eqn. 5 and 10)} \qquad (15)$$

In this case, since no look-ahead is involved, E_{avg} is independent of c and d.

4.6 Effect of c on ACQUIRE

We first analytically study the behavior of ACQUIRE for different values of c and d and find the optimal look-ahead d^* for a given c, M and N. We used Eqn. 13 derived in section 4.5. N was set to 100 and M was set to 20. We varied c from 0.001 to 1 in steps of 0.001 and d from 1 to 10. For $d = 0$, E_{avg} is independent of c and d as shown by Eqn. 15 in section 4.5.

Figure 4 shows the energy consumption of the ACQUIRE scheme for different amortization factors and look-ahead values. Let d^* be the look-ahead value which produces the minimum average energy consumption. It appears that d^* significantly depends on the amortization factor.

Figure 5 shows that as the amortization factor c decreases, d^* increases. i.e. as the query rate increases and the network dynamics decreases it is more energy-efficient to have a higher look-ahead. This is intuitive because in this case, with a larger look-ahead, the sensor can get more information that will remain stable for a longer period of time which will help it to answer subsequent queries. Thus, in our study, for very small c ($0.001 \leq c \leq 0.01$), d^* is as high as possible ($d^* = 10$). On the other hand, for $0.08 \leq c < 0.9$ (approx.), the most energy efficient strategy is to just request information from the immediate neighbors ($d = 1$). It is also seen that there are values of c in the range from $[0.001, 0.1]$ such that each of $1, 2, ...10$ is the optimal look-ahead value. If $c \geq 0.9$ (approx.), the most efficient strategy for each node x is to resolve the query based on the information it has (without even requesting for information from its neighbors i.e. $d = 0$).

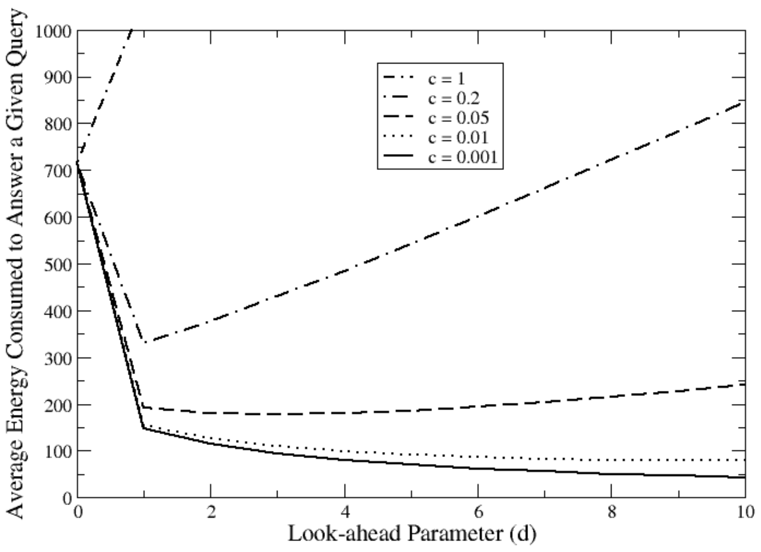


Figure 4: Effect of c and d on the Average Energy Consumption of the ACQUIRE scheme. Here, $N = 100$ and $M = 20$

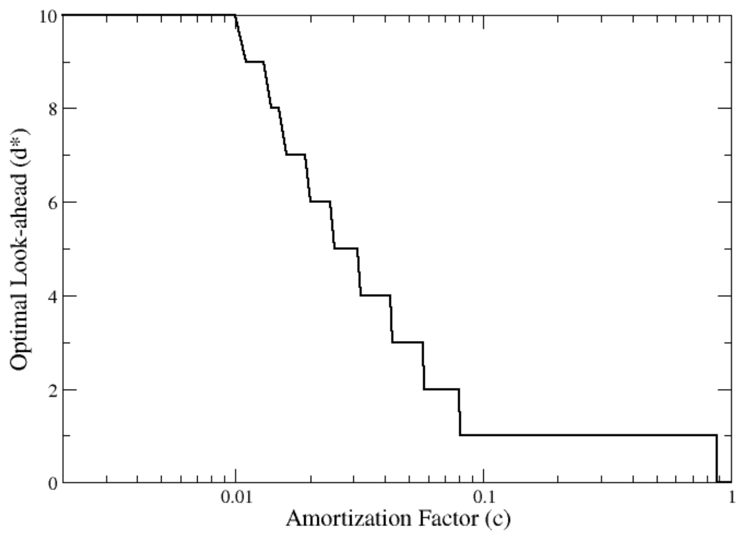


Figure 5: Effect of c on d^* for $N = 100$ and $M = 20$. The x-axis is plotted on a log scale.

5 Analysis of Alternative Approaches

In this section, we present the energy cost of expanding ring search and flooding based query mechanisms. While the expanding ring search is not currently implemented in any sensor querying protocols we are aware of, it is a better baseline for comparison than the clearly inefficient flooding-based querying, as far as one-shot queries are concerned.

5.1 Expanding Ring Search (ERS)

In an Expanding ring search, at stage 1, the querier x^* will request information from all sensors exactly one hop away. If the query is not completely resolved in the first stage, x^* will send a request to all sensors two hops away in the second stage. Thus, in general at stage i, x^* will request information from sensors exactly i hops away. The average number of stages t_{min} taken to completely resolve a query of size M can be approximately determined by the First order analysis in section 4.2.1:

$$\begin{aligned}
\sum_{i=1}^{t_{min}} N(i) &= N(\ln M + \gamma) \qquad \text{(from Eqn.5)} \\
4\sum_{i=1}^{t_{min}} i &= N(\ln M + \gamma) \qquad \text{(from Eqn.12)} \\
2t_{min}(t_{min}+1) &= N(\ln M + \gamma) \\
2(t_{min})^2 + 2t_{min} - N(\ln M + \gamma) &= 0 \qquad (16)
\end{aligned}$$

t_{min} can be determined by solving the above quadratic equation (taking the ceiling if necessary to get t_{min} as an integer).

In ERS, at stage i, all nodes within $i-1$ hops of the querier x^* will forward the x^*'s request. Let $N_{avg}(i)$ be the expected number of nodes at hop i that will resolve some sub-query. The response from these nodes will be forwarded over i hops. There are a total of t_{min} stages. Thus, the total update cost is given as follows:

$$\begin{aligned}
E_{update} &= \sum_{i=1}^{t_{min}} (f(i-1) + iN_{avg}(i)) \\
&= \sum_{i=1}^{t_{min}} f(i-1) + \sum_{i=1}^{t_{min}} iN_{avg}(i) \qquad (17)
\end{aligned}$$

$N_{avg}(i)$ can be computed as follows:
At the i^{th} step, $f(i-1)$ nodes would already have been requested for their information. The expected number of queries resolved $M_r(i-1)$ before the i^{th} step can be given as follows:

$$\begin{aligned}
f(i-1) &= N(\ln(M_r(i-1)) + \gamma) \qquad \text{(from Eqn.5)} \\
M_r(i-1) &= e^{\frac{f(i-1)}{N} - \gamma} \qquad (18)
\end{aligned}$$

Thus, in the i^{th} step, the probability of "success" is given by

$$p_i = \frac{M - M_r(i-1)}{N} \tag{19}$$

Thus, in step i, the expected number of nodes that will resolve some sub-query is given by:

$$\begin{aligned} N_{avg}(i) &= N(i)p_i \qquad \text{(substituting } p_i \text{ for } p \text{ in Eqn.23)} \\ &= N(i)(\frac{M - e^{\frac{f(i-1)}{N} - \gamma}}{N}) \end{aligned} \tag{20}$$

Since the query is not forwarded to any other node,

$$\begin{aligned} E_{avg} &= E_{update}c \qquad \text{(substituting } d = 0, \;\; \alpha = 0, \;\; S_M = 1 \text{ in Eqn.8)} \\ &= (\sum_{i=1}^{t_{min}}(2i(i-1)+1) + \sum_{i=1}^{t_{min}} iN(i)(\frac{M - e^{\frac{f(i-1)}{N} - \gamma}}{N}))c \\ &= (\frac{1}{3}(t_{min})(t_{min}+1)(2t_{min}+1) - (t_{min})(t_{min}+1) + t_{min} + \sum_{i=1}^{t_{min}} iN(i)(\frac{M - e^{\frac{f(i-1)}{N} - \gamma}}{N}))c \\ &= (\frac{2}{3}(t_{min})(t_{min}+1)(t_{min}-1) + t_{min} + \sum_{i=1}^{t_{min}} iN(i)(\frac{M - e^{\frac{f(i-1)}{N} - \gamma}}{N}))c \end{aligned} \tag{21}$$

5.2 Flooding-Based Query (FBQ)

In FBQ, the querier x^* sends out a request to all its immediate neighbors. These nodes in turn, resolve the query as much as possible based on their information and then forward the request to all their neighbors and so on. Thus, the request reaches all the nodes in the network.

In general, as mentioned in Eqn. 8 from section 4.4,

$$E_{avg} = (cE_{update} + d)S_M + \alpha \tag{22}$$

In FBQ:

1. The request for triggered updates will have to be sent as far as R hops away from the querier x^* (near the center of the grid) where R is the "radius" of the network i.e. the maximum number of hops from the center of the grid.

2. $d = 0$, as the query is not forwarded.

3. $\alpha = 0$, as the query is completely resolved at the origin of the query itself.

4. $S_M = 1$.

Let $N_{avg}(i)$ be the expected number of nodes at hop i, that can resolve some part of the query. This can be determined along similar lines as in section 4.2.1:
As before, consider the fetching of information from a sensor as a "trial". In each "trial", the probability of success is $p = \frac{M}{N}$ and the probability of failure is $q = \frac{N-M}{N}$. The number of successes is a binomial random variable. The total number of "trials" at hop i is $N(i)$. Thus, the expected number of successes at hop i is given by

$$N_{avg}(i) \quad = \quad N(i)p \tag{23}$$

The response of each of the $N_{avg}(i)$ nodes will be forwarded over i hops.
Thus, for FBQ, E_{avg} is given as follows:

$$\begin{aligned} E_{avg} &= (f(R) + \sum_{i=1}^{R} iN_{avg}(i))c \\ &= (f(R) + \sum_{i=1}^{R} iN(i)\frac{M}{N})c \\ &= (f(R) + \frac{M}{N}\sum_{i=1}^{R} iN(i))c \\ &= (2R(R+1) + 1 + \frac{2}{3}\frac{M}{N}R(R+1)(2R+1))c \end{aligned} \tag{24}$$

For a grid with X nodes, $R = \sqrt{X}$. Thus, For a given M, N and c, $E_{avg} \propto X^{3/2}$.

6 Comparison of ACQUIRE, ERS and FBQ

6.1 Effect of c

These schemes were analytically compared across different values of c chosen in the range of $[0.001, 1]$ as in section 4.6. The total number of nodes X was 10^4. For ACQUIRE, the look-ahead parameter was set to d^* for a given value of c. We refer to this version of ACQUIRE as $ACQUIRE^*$. Eqn. 21 from section 5.1, Eqn. 24 from section 5.2 and Eqn. 13 from section 4.5 (with $d = d^*$) were used in the comparative analysis. For the initial comparisons, $N = 100$ and $M = 20$. Using these values for M and N in Eqn. (16) from section 5.1, we obtain $t_{min} = 13$. This value of t_{min} is then used in Eqn. 21.

As figure 6 shows that ACQUIRE with look-ahead 0 (i.e. random walk) performs at least as worse as ACQUIRE with the optimal look-ahead ($ACQUIRE^*$). $ACQUIRE^*$ seems to outperform ERS for higher values of the amortization factor. Moreover at $c = 1$, ACQUIRE gives about a 60% energy savings over ERS. In the case of $N = 100$ and $M = 20$, $ACQUIRE^*$ outperforms ERS if $c \geq 0.08$ (approx.). In this case, $d^* = 1$ as shown by figure 5.

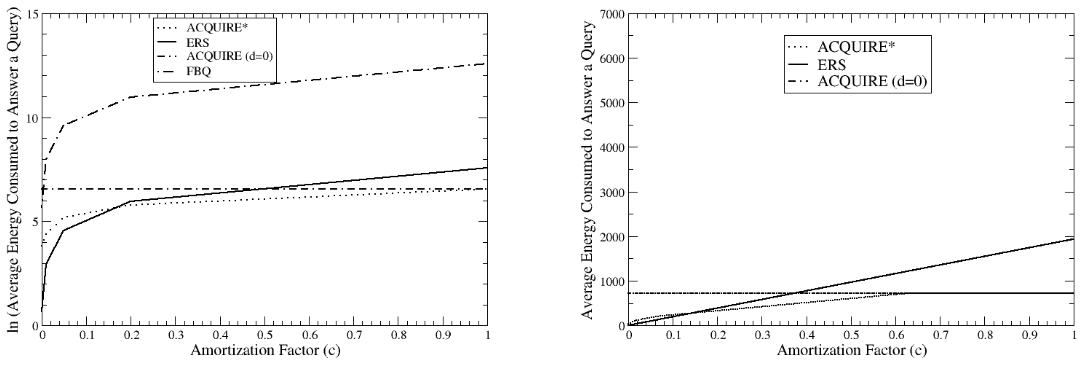


Figure 6: Comparison of $ACQUIRE^{*}$, ERS, ACQUIRE with $d = 0$ and FBQ with energy on a log scale (left) and a linear scale (right). (For $N = 100$ and $M = 20$)

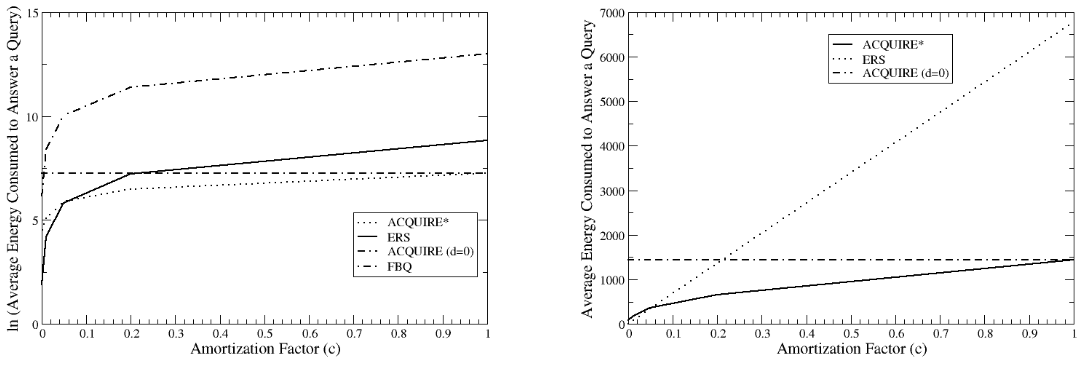


Figure 7: Comparison of $ACQUIRE^{*}$, ERS, ACQUIRE with $d = 0$ and FBQ with energy on a log scale (left) and on a linear scale (right) (for $N = 160$ and $M = 50$).

However, it is not always the case that $ACQUIRE^*$ outperforms ERS only with $d^* = 1$. If $N = 160$ and $M = 50$, $t_{min} = 19$ (from Eqn. 16). With these values of M and N as shown by the figure 7, $ACQUIRE^*$ outperforms ERS if $c \geq 0.06$ (approx.). In this case, $d^* = 2$ as shown by figure 5. Moreover in this case as figure 7 shows, for $c > 0.2$, even ACQUIRE with $d = 0$ would outperform ESR. Again, for $c = 1$, $ACQUIRE^*$ achieves more than 75% energy savings over ERS.

As figure 6 shows, FBQ, on an average, incurs the worst energy consumption which is several orders of magnitude higher than the other schemes. This is mainly because of a very large number of nodes ($X = 10^4$) used in our study.

From, the above analysis, it seems that the relative query size ($\frac{M}{N}$), seems to have a significant impact on the performance of ACQUIRE and ERS. Hence we analyze this effect in section 6.2. FBQ always incurs an order of magnitude worse energy consumption, and hence we do not study the impact of the relative query size on FBQ.

6.2 Effect of $\frac{M}{N}$

Intuitively, for a given N, as M is increased, ERS has to "expand" the ring more, while ACQUIRE will take more steps to resolve the query. In this section, we fix N at 100 and let M to take the values of 10, 20, 40, 60, 80 and 100. For each value of $\frac{M}{N}$, we observe the performance of $ACQUIRE^*$ and ERS for $c = 0.05$, $c = 0.2$ and $c = 1$. For ERS, the values of t_{min} for these values of $\frac{M}{N}$ are 12, 13, 15, 15, 16 and 16 respectively.

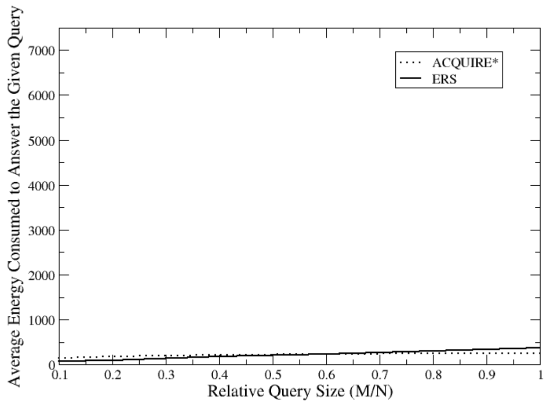
As the figures 8 shows, the average energy consumed to resolve a complex query increases with increasing M for a given N. For $c = 0.05$, ERS performs better than $ACQUIRE^*$ when $\frac{M}{N} \leq 0.5$, while in the other cases $ACQUIRE^*$ outperforms ERS across all the amortization factors and relative query sizes in our study. In these cases, the energy savings of $ACQUIRE^*$ over ERS are seen to range as high as 85% (e.g. when $c = 1$, $\frac{M}{N} = 1$).

Thus, both c and $\frac{M}{N}$ seem to have a significant impact on the performance of ACQUIRE and ERS. As c increases and $\frac{M}{N}$ increases, ACQUIRE achieves significant energy savings over ERS (and FBQ).

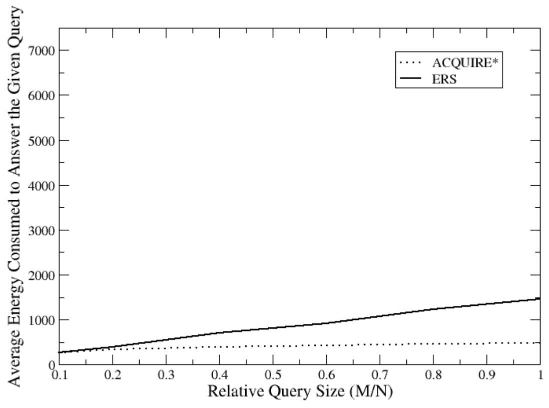
7 Validation of the Analytical Models

In this section, we validate our analytical models by conducting some high level simulations. Specifically, our objectives were the following:

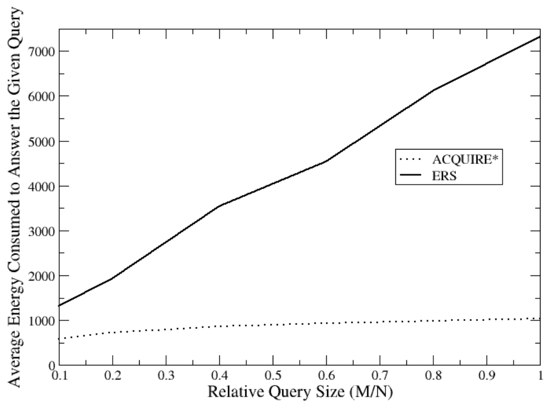
1. Validate the effect of c on ACQUIRE.
2. Validate the comparisons between ACQUIRE, ERS and FBQ for different values of c.



(a) $c = 0.05$



(b) $c = 0.2$



(c) $c = 1$

Figure 8: Comparison of $ACQUIRE^*$, and ERS ($N = 100$) with respect to $\frac{M}{N}$ for different amortization factors.

3. Perform a relative comparison between ACQUIRE and ERS across different relative query sizes ($\frac{M}{N}$).

As we shall show, our simulation results are more or less in line with our analysis. The minor differences can be ascribed to factors such as overlap in the query trajectory and boundary effects that are not modelled in our analysis.

7.1 Simulation Setup

Our setup consists of a 100m x 100m grid with 10^4 sensors placed at a distance of 1m from each other. The communication range of each sensor is 1m. The total number of variables in our simulations is set to 100. For all the querying mechanisms, a query is always injected at the center of the grid i.e. at sensor $(50, 50)$. We ran our simulations on 1000 queries. In order to take advantage of the caching, the query was made to follow a fixed trajectory in ACQUIRE. The first query out of the set of 1000 queries fixes the trajectory, while all the subsequent queries follow the same trajectory.

Our analysis assumed that there are no loops in the query trajectory. It turned out that this can very effectively achieved by ACQUIRE's local update phase at no additional cost. Each node maintains a flag called *queried*. Whenever a node is requested for an update, it sets this flag to *true*. Subsequently, whenever a node is requested for an update, it sends the value of the *queried* flag along with the variable. Once an *active* node has processed the query based on the information in its neighborhood, it forwards the remaining query to a node at d hops whose *queried* flag is *false*. Using this mechanism, most of our simulations have no loops in the query trajectory for $d \geq 1$. However, for the random walk ($d = 0$), there is no local update phase. Hence, in this case, there are loops in the query trajectory causing the random walk to revisit nodes. These loops lead to a $45 - 50\%$ degradation in the performance[11] as we will show in section 7.3. For ERS and FBQ, there is no trajectory as the query is never forwarded to other nodes.

7.2 Effect of c on ACQUIRE

In our simulations, we used 5 different values of c i.e. $0.001, 0.01, 0.05, 0.2$ and 1. We simulated the c as follows: Each variable has a validity time of $\frac{1}{c}$, where time is taken to be the number of queries. E.g. If $c = 0.001$, each variable is valid for 1000 queries. During this validity period, all queries for that particular variable can be answered using the cached copies. Beyond the validity period, the active node has to "refresh" variables from its "neighborhood". For each value of c, simulations were run using 100 different random seeds.

[11]The analysis of ACQUIRE with $d = 0$ case assumed that there is no looping. The significant performance degradation observed in simulations suggests that the use of straightening algorithms such as those described in [21] or geographically directed trajectories such as routing on curves [20] is necessary when ACQUIRE is used with $d = 0$ in practice.

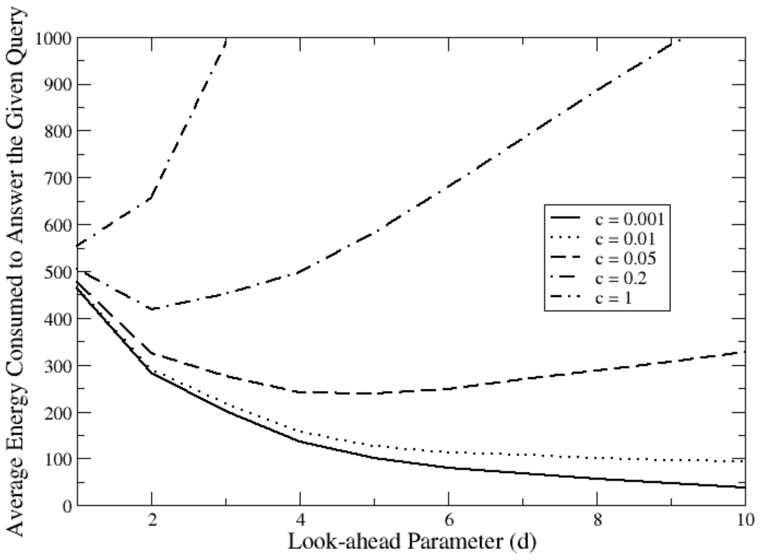


Figure 9: Effect of c on d^* by simulations ($N = 100$, $M = 20$). Compare with theoretical curves in figure 4

In each run, we used 1000 queries, each consisting of 20 sub-queries (or variables). For each run, the generated queries were stored in a query-file. At the same time the values chosen by each sensor were also stored in a grid-file for each run. The number of transmissions were averaged across all these runs for a given c.

As figure 9 shows, with increasing c, the optimal look-ahead d^* decreases. This concurs with our analysis in section 4.5. The simulations show that for $c = 0.001, 0.01$, the $d^* = 10$ (largest possible value used in our simulations), which is the same as shown by our analytical curves in figure 4. For $c = 0.05$, $d^* = 5$ from simulations, while the analytical $d^* = 3$. For $c = 1$, from simulations $d^* = 1$, while from the analytical $d^* = 0$. This is because in simulations, as mentioned in section 7.1, ACQUIRE with $d = 0$ has loops in its trajectory, which degrades its performance by around $45 - 50\%$.

7.3 Comparison of ACQUIRE, ERS and FBQ

7.3.1 Effect of c

For both ERS and FBQ, we use the same simulation setup as ACQUIRE. For both these mechanisms, we simulate 100 different runs. Each run consists of 1000 queries each containing 20 sub-queries. In each run, we use the query-files, grid-files and same values of c described in section 7.2.

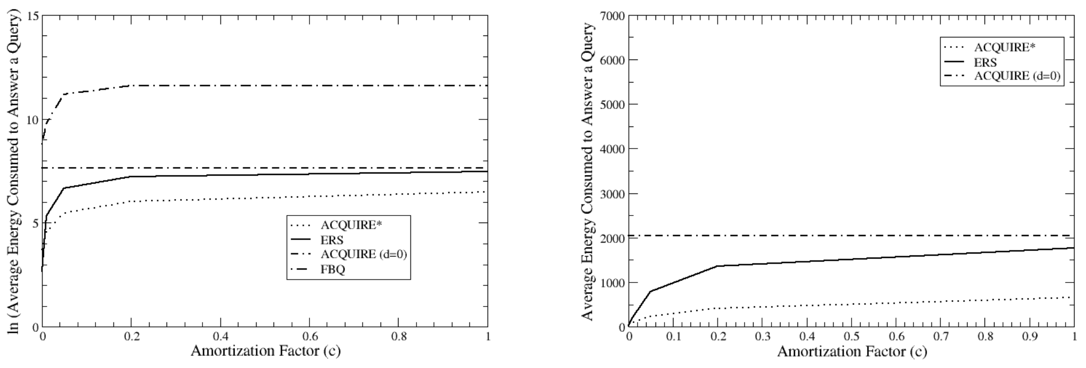


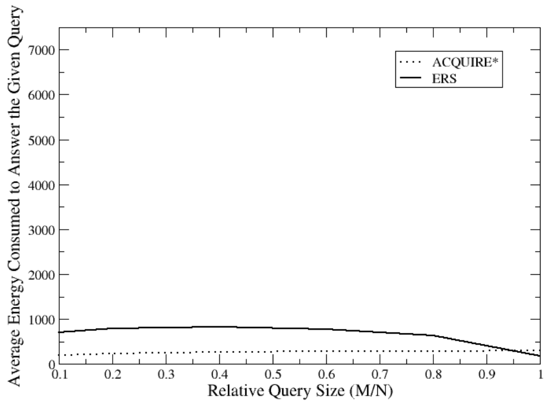
Figure 10: Comparison of $ACQUIRE^*$, ERS, ACQUIRE with $d = 0$ and FBQ by simulations ($N = 100,\ M = 20$) with energy costs on a log scale (left) and on a linear scale (right). Compare with theoretical curves in figure 6.

As figure 10 shows, FBQ has the worst energy consumption in comparison with the other approaches. This concurs with the analytical curves in figure 6. ACQUIRE with $d = 0$ (random walk) incurs a greater energy consumption as compared to ACQUIRE with $d = 1$ as well as ERS, while the analytical curves show that at high values of c, ACQUIRE with $d^* = 0$ is better than ERS.This disrepancy is because of the loops in the query trajectory in the case of random walk as was mentioned in section 7.1. However, the energy savings of $ACQUIRE^*$ over ERS seem to be similar to that shown by the analytical curves in figure 6.

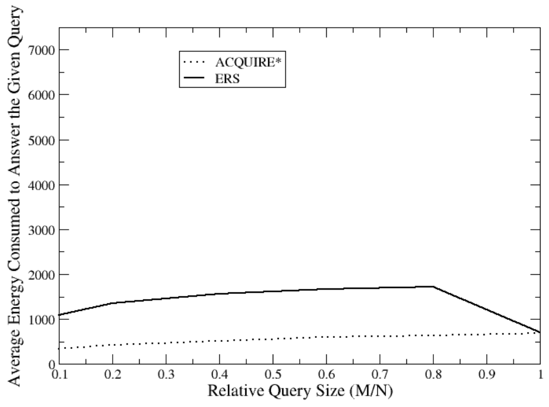
7.3.2 Effect of $\frac{M}{N}$

We use a similar set of simulation runs as mentioned in section 7.3.1 for values of c and $\frac{M}{N}$ as mentioned in section 6.2

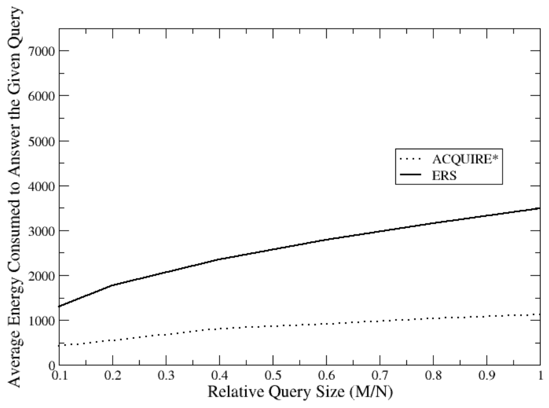
As figure 11 shows, for all values of c considered in this study, the average energy consumption of $ACQUIRE^*$ increases with increasing $\frac{M}{N}$. This behavior is also seen for ERS when $c = 1$. These observations are consistent with the analytical behavior in section 6.2. However, for $c = 0.05$ and $c = 0.2$, we should note that ERS seems to have a lower energy consumption at $\frac{M}{N} = 1$. This has to do with the way we modelled the caching behavior of ERS in section 5.1. For all mechanisms, we had modelled the update frequency as $\frac{1}{c}$; however in ERS the central querier caches only those variables which are part of the query. In such situations, in the simulations, an update would occur if even a single variable of a new query cannot be resolved from the cache. This makes our ERS analysis differ from the simulations when $c < 1$ and $M < N$. However, when $M = N$, all variables can be cached at the center, and one in every $1/c$ queries can answered directly from the cache, reducing the query cost. If $c = 1$, ERS will need to update every query and hence in this case, the energy consumption increases with increasing $\frac{M}{N}$. The analysis and simulation for ACQUIRE are better matched because because in ACQUIRE we cache all the variables within d hops (not just those queried



(a) $c = 0.05$



(b) $c = 0.2$



(c) $c = 1$

Figure 11: Comparison of $ACQUIRE^*$ and ERS by simulations ($N = 100$) with respect to $\frac{M}{N}$ for different amortization factors. Compare with theoretical curves in figure 8

for).

From simulations, at $\frac{M}{N} = 1$, $c = 0.05$, ERS performs better than $ACQUIRE^*$. On the other hand, for all other values of c used in our study, $ACQUIRE^*$ outperforms ERS across all relative query sizes. The energy savings of $ACQUIRE^*$ over ERS is around 65% for $c = 0.02$ (for all values $\frac{M}{N} < 1$) and $c = 1$ (for all values of $\frac{M}{N}$).

Our analysis of the energy cost of ACQUIRE, ERS and FBQ by analytical models and simulations illustrate that ACQUIRE achieves significant energy savings for moderate to high values of c depending on the relative query size. Do these energy savings come at a cost? We attempt to answer this question in the next section by modelling the average response latency incurred by these mechanisms.

8 Latency Analysis

In this section, we attempt to analytically compare the average latency in obtaining a response to a query by these 3 mechanisms. The metric for latency that we examine is the expected number of sequential transmissions required before a response is obtained to a given query. We should note that this is a network layer analysis that does not take into account MAC delay due to contentions.

8.1 ACQUIRE

In this section, we analyze the latency incurred by $ACQUIRE^*$ in answering a query.

$ACQUIRE^*$ takes S_M steps to answer a query of size M, where S_M is given by equation 6 in section 4.2.1. Each of these steps involve an update phase, where in a request is propagated within a neighborhood of d hops, while the responses are propagated over a maximum of d^* hops. Moreover, once the query is completely resolved, the response is sent back to the querier (modelled as $\alpha \leq d^* S_M$). Also, the update phase will be done only once every $\frac{1}{c}$ queries. Thus, the average latency of $ACQUIRE^*$ can be given as follows:

$$\begin{aligned} T_{avg} &= c\{S_M(2d^*) + 2S_M(d^*)\} + (1-c)\{2S_M(d^*)\} \\ &= 2S_M d^*(c+1) \\ T_{avg} &= 2d^*\{\frac{N(ln(M)+\gamma)}{f(d^*)}\}\{c+1\} \end{aligned} \tag{25}$$

For the random walk,

$$\begin{aligned} T_{avg} &= E_{avg} \\ &= 2N(ln(M)+\gamma) \end{aligned} \tag{26}$$

8.2 ERS

In ERS, on an average the ring has to expand till t_{min} hops to get updates, where t_{min} is given by Eqn. 16. The latency in executing a ring of size x is $2x$, x for the request and x for the reply. Moreover, these updates are sought once every $\frac{1}{c}$ queries. The remaining fraction of the queries are answered from the cached responses (which has a latency of 0). Thus, the average latency in ERS is given as follows:

$$\begin{aligned} T_{avg} &= c\{\sum_{i=1}^{t_{min}} 2i\} + (1-c)\{0\} \\ &= c(t_{min})(t_{min}+1) \qquad (27) \end{aligned}$$

8.3 FBQ

Similar to the latency analysis for ERS, in FBQ, the request for updates has to be forwarded for t_{min} hops on an average before all the sub-queries can be answered. Thus, the latency for issuing the request and getting the updates is $2t_{min}$. The updates are issued once every $\frac{1}{c}$ queries. For the remaining fraction, the queries are answered from the cache (incurring a latency of 0). Thus, the average latency incurred by FBQ is

$$T_{avg} = 2ct_{min} \qquad (28)$$

Figure 12 shows the analytical comparison of $ACQUIRE^*$, ERS and FBQ when $N = 100$, $M = 20$ and $X = 10^4$. The average latency seems to increase from FBQ to ERS to $ACQUIRE^*$ across all values of the amortization factor c. The latency for both FBQ and ERS increases linearly with c as is evident from Eqn. 28 and Eqn. 27 respectively. Interestingly, the latency in $ACQUIRE^*$ has a piece-wise linear behavior with respect to c. This is because the changing c alters d^* discontinuously which in turn alters $\frac{d^*}{f(d^*)}$ i.e. the slope of the line T_{avg} as shown by Eqn. 25. At the point where the latency is constant with respect to c, $ACQUIRE^*$ resembles a random walk ($d = 0$). The difference in the latency is significant (around 500 transmissions), when d^* for ACQUIRE goes from 1 to 0.

9 Storage Requirements

Our analysis in this paper assumes that all the mechansims i.e. ACQUIRE, ERS and FBQ utilize caching to answer queries. We now attempt to quantify the storage space requirements for the cache across these approaches.

In ACQUIRE, an active node requests an update from its "neighborhood" consisting of $f(d)$ (with $d = d^*$). In the worst case, the each of these $f(d)$ variables might be distinct, thus needing $O(f(d))$ storage space. For a grid and for most reasonable topologies, note that

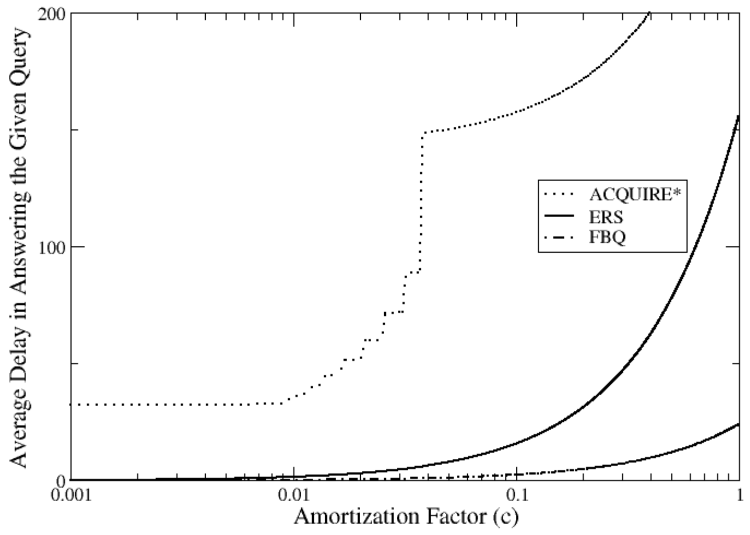


Figure 12: Comparison of the latency incurred by $ACQUIRE^{*}$, ERS and FBQ. Here, $N = 100$, $M = 20$, $X = 10^4$, $t_{min} = 13$. The x-axis is plotted on log-scale.

$f(d)$ would be polynomial in d. ACQUIRE distributes the cache at nodes which handle the query. In the case of ACQUIRE, the storage requirements depend on the optimal lookahead d^*, which in turn is dependent on the data dynamics. Higher c, higher the data dynamics, lower the optimal lookahead d^*, smaller the cache requirements at each node.

For ERS and FBQ, each querier (there may be only one, always located at the same node) requires a cache that is between $O(M)$ and $O(N)$ in size. This is because this querier node must cache all responses to all valid queries.

10 Discussion and Future Work

Partly for ease of analysis, we have described and modelled a very basic version of the ACQUIRE mechanism in this paper. One of our major next steps is to convert ACQUIRE into a functional protocol that can be validated on an experimental sensor network test-bed. There are a number of ways in which our analysis can be improved, and a number of additional design issues need to be considered in our future work, some of which we outline here.

Our analytical model of ACQUIRE assumes that the query packet is always of a fixed size consisting of all the individual sub-queries and their responses. The entire packet circulates in the network till the answer to the last query is obtained. The packet is then sent back to the querier. This simplifies the analysis as we need to only count the number of transmissions in order to quantify E_{avg}. However, it may be more efficient to send the answers to sub-queries to the querier node as and when they are obtained. Our analysis could be tightened to take this into consideration.

The efficiency of ACQUIRE can also be improved if the neighborhoods of the successive active nodes in the query trajectory have minimal overlap. This may potentially be best accomplished by using some deterministic trajectory as opposed to random walks, possibly making use of additional topological or geographical information. These would also aid in minimizing inefficiency due to the walk revisiting nodes (looping). Guided trajectories may also be helpful in dealing with non-uniform data distributions, ensuring that active queries spend most time in regions of the network where the relevant data are likely to be. In the analysis, we ignored the issue of overlap (although this was taken into account in the simulations we presented).

One interesting result of our analysis is that the performance of ACQUIRE and the optimal choice of the look-ahead parameter d^* are functions of the amortization factor c and (somewhat surprisingly) independent of M, N, and the total number of nodes X. This lends itself to the possibility of using distributed algorithms in which localized estimates of c are used to determine the value of d at each step without global knowledge of system parameters. This would significantly improve the scalability of ACQUIRE.

As presented here, ACQUIRE is meant to be used in situations where there is replicated data. At the very least there should be one node in the network that can resolve each component sub-query. One way to deal with other situations might be to equip the active queries with a time-to-live (TTL) field which is decremented at each hop. This would permit ACQUIRE to gracefully terminate with a negative response if a solution is not found within a reasonable period of time, to be followed up (for example) by a flooding-based query.

Our analysis has assumed a regular grid topology. This helped us in gaining considerable insight into the performance of ACQUIRE, ERS and FBQ. In reality the topology of a sensor network might not only be irregular but also dynamic, due to failures and mobility. Exploring the behavior of ACQUIRE on such topologies is a focus of our ongoing effort. We should mention, however, that our results do already have some generality in this regard: so long as a reasonable model for $f(d)$ can be developed for the network topology, the analysis presented here can be extended in a straightforward manner.

In our modelling we have only counted the number of transmissions for energy costs, although it is true that receptions can also influence energy consumption. This is the case especially for broadcast messages, where there's no channel reservation and all the direct neighbors receive the message. We believe that some of the alternatives to active querying, such as FBQ and ERS will in fact incur even more energy consumption under an energy model that incorporates receptions because all their query messages are broadcast. Moreover, these broadcasts would also lead to an increased latency in FBQ and ERS due to higher contention. We would like to examine such richer energy cost models in the future.

In our analysis of delay, we looked only at response latency at the network layer (by examining the number of maximum sequential transmissions required). These results must be taken with a grain of salt, because they do take into account MAC-layer delay. For broadcast-based querying techniques such as FBQ and ERS, there could be far greater MAC layer contention than in ACQUIRE. This deserves worth further investigation.

We have also ignored the possibility of aggregate queries in this paper. Our assumption has been that each sub-query is independent. This would be another direction for future work.

11 Conclusions

In this paper, we have proposed ACQUIRE - a novel mechanism for data extraction in energy-constrained sensor networks. The key feature of ACQUIRE is the injection of active queries into the network with triggered local updates. We first categorized sensor network query types and identified those for which ACQUIRE is likely to perform in an energy-efficient manner: complex, one-shot, non-aggregate queries for replicated data.

We have developed a fairly sophisticated mathematical model that allows us to analytically evaluate and characterize the performance (in terms of energy costs and response latency) of ACQUIRE, as well as alternative techniques such as flooding-based queries (FBQ) and expanding ring search (ERS). As far as we are aware, there are very few similar results in the literature that provide similar mathematical characterizations of the performance of query techniques for sensor networks. We validated our analysis through extensive simulations and also identified ways in which the models can be extended and improved.

In our analysis we defined an amortization factor c to meaningfully capture the relationship between the query rate and data dynamics. When c is low, more queries can be processed in the time that a given datum remains "fresh." Our analysis revealed that this parameter has a significant impact on the energy costs of cached update schemes such as the one used in ACQUIRE. Indeed, we showed that the optimal look-ahead in ACQUIRE depends solely upon c, not on other parameters such as the size of the network or the size of the queries.

We found that ACQUIRE with optimal parameter settings outperforms the other schemes for complex, one-shot queries in terms of energy consumption. Specifically, optimal ACQUIRE performs many orders of magnitude better than flooding-based schemes (such as Directed Diffusion) for such queries in large networks. We also observed that optimal ACQUIRE can reduce the energy consumption by more than $60 - 75\%$ as compared to expanding ring search (in highly dynamic environments and high query rates). The energy savings are highest when c is high and $NlnM$ is high. However, this energy savings come at the cost of increased average latency in answering a query.

To conclude, we believe that there is no one-size-fits-all answer to the question: "How do we efficiently query sensor networks?" We propose ACQUIRE as a highly scalable technique, energy-efficient at solving complex one-shot queries for replicated data. We argue that

ACQUIRE deserves to be incorporated into a portfolio of query mechanisms for use in real-world sensor networks.

References

[1] D. Estrin, L. Girod, G. Pottie, and M. Srivastava, “Instrumenting the World with Wireless Sensor Networks,” *International Conference on Acoustics, Speech and Signal Processing (ICASSP 2001)*, Salt Lake City, Utah, May 2001.

[2] J. Warrior, “Smart Sensor Networks of the Future,” *Sensors Magazine*, March 1997.

[3] G.J. Pottie, W.J. Kaiser, “Wireless Integrated Network Sensors,” *Communications of the ACM*, vol. 43, no. 5, pp. 551-8, May 2000.

[4] A. Cerpa *et al.*, “Habitat Monitoring: Application Driver for Wireless Communications Technology,” *2001 ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, Costa Rica, April 2001.

[5] C. Intanagonwiwat, R. Govindan and D. Estrin, “Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks,” *ACM/IEEE International Conference on Mobile Computing and Networks (MobiCom 2000)*,August 2000, Boston, Massachusetts

[6] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann, “Impact of Network Density on Data Aggregation in Wireless Sensor Networks” , In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS’02)*, Vienna, Austria. July, 2002.

[7] B. Krishnamachari, D. Estrin, and S. B. Wicker, “”The Impact of Data Aggregation in Wireless Sensor Networks,” *International Workshop on Distributed Event-Based Systems, (DEBS ’02)*, Vienna, Austria, July 2002.

[8] D. Estrin, R. Govindan, J. Heidemann and S. Kumar, “Next Century Challenges: Scalable Coordination in Sensor Networks,” *ACM/IEEE International Conference on Mobile Computing and Networks (MobiCom ’99)*, Seattle, Washington, August 1999.

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

[10] P. Bonnet, J. E. Gehrke, and P. Seshadri, “Querying the Physical World, ” *IEEE Personal Communications*, Vol. 7, No. 5, October 2000.

[11] P. Bonnet, J. Gehrke, P. Seshadri, “Towards Sensor Database Systems,” *Mobile Data Management*, 2001.

[12] S. Garg, P. Pamu, N. Nahata, A. Helmy, “Contact Based Architecture for Resource Discovery (CARD) in Large Scale MANets”, USC-TR, July 2002. (Submitted for Review).

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

[14] M. Chu, H. Haussecker, F. Zhao, "Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks." Int'l J. High Performance Computing Applications, to appear, 2002. Also, Xerox Palo Alto Research Center Technical Report P2001-10113, May 2001.

[15] Zygmunt J. Haas, Marc R. Pearlman, and Prince Samar, "The Zone Routing Protocol (ZRP) for Ad Hoc Networks," *IETF MANET Internet Draft*, July 2002.

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

[17] W.R. Heinzelman, J. Kulik, and H. Balakrishnan "Adaptive protocols for information dissemination in wireless sensor networks," *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99)*, pp. 174-185, Seattle, WA, August 1999.

[18] W.R. Heinzelman, A. Chandrakasan, and H. Balakrishnan "Energy-efficient communication protocol for wireless microsensor networks," *33rd International Conference on System Sciences (HICSS '00)*, January 2000.

[19] S. Madden, M. J. Franklin, J. M. Hellerstein and W. Hong, "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks", *Proceedings of the Fifth Annual Symposium on Operating Systems Design and Implementation (OSDI)*, to appear, December 2002.

[20] Badri Nath and Dragos Niculescu, "Routing on a curve," *HotNets-I*, Princeton, NJ, October, 2002.

[21] David Braginsky and Deborah Estrin, "Rumor Routing Algorithm For Sensor Networks," *First Workshop on Sensor Networks and Applications (WSNA)*, September 2002.

[22] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker "Search and replication in unstructured peer-to-peer networks," In *ICS'02*, New York, USA, June 2002.

12 Appendix: Detailed Analysis of Steps to Query Completion

Assume that a query Q consisting of M sub-queries is at a sensor x. If d is the look-ahead, then sensor x will have information about the values stored by $f(d)$ sensors. Thus, x can resolve at most $\min(f(d), M)$ out of the M sub-queries. In the worst case, x cannot resolve any of the M sub-queries. After resolving the possible queries, x will forward the remaining query $Q' \subseteq Q$ to a sensor which is chosen uniformly at random from those exactly d hops away. Assuming that whenever a sensor gets the query, it can always get information from $f(d)$ new nodes (i.e. there are no loops in the query forwarding process and the topology of the network is regular), the probability of answering K' of the M sub-queries is dependent only on the information obtained from the $f(d)$ nodes. This characteristic of the query

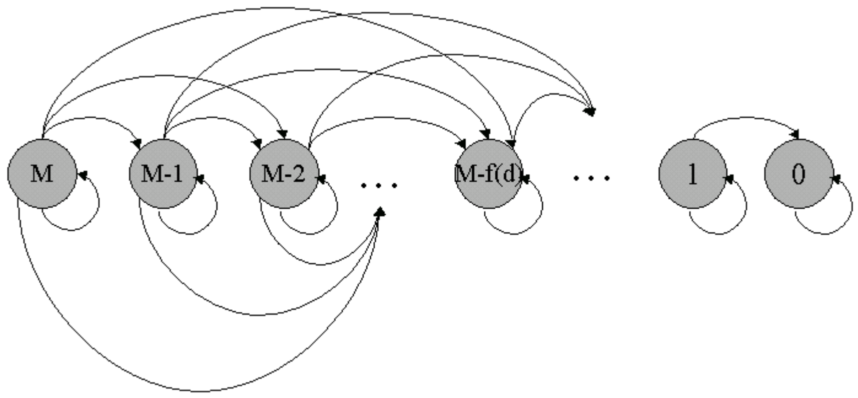


Figure 13: Markov Chain with states representing the number of unresolved sub-queries of an active query. Transitions are only to lower-valued states at most $\min(f(d), M)$ to the right, and the chain terminates at the absorbing state 0 when all sub-queries have been resolved.

forwarding process naturally lends itself to be modelled as a Markov Chain, as shown in figure 13.

The states of this Markov Chain are the number of unresolved sub-queries at any instant. Thus, given a query consisting of M sub queries, the states of the Markov Chain are $M, M-1, M-2, ...0$. State 0 is the absorption state. This Markov Chain has certain characteristics:

1. There is no transition from state M' to M'' where $M' < M''$
2. From a state M', there can be no transition to a state M'' where $M' - M'' > f(d)$

The Markov Chain formulation is useful in that once the transition probabilities between the various states are known, the mean time to absorption (i.e. S_M) can be easily calculated. Our next step is to find the transition probabilities of the Markov Chain.

Transition Probabilities:

Let K (with a slight abuse of notation) be the event of getting answers to K distinct sub-queries from $f(d)$ sensors. We now determine $P(K)$.

The answers obtained from these $f(d)$ sensors can be considered as strings of length $f(d)$, where each character is a variable $V_i, 1 \leq i \leq N$. Let $A(K)$ be the number of strings of length $f(d)$ which contain each of the variables $V_{l1}, V_{l2}, ...V_{lK}$ i.e. the number of strings consisting of the given set of K distinct variables. Now,

$$\begin{aligned} A(0) &= 0 \\ A(1) &= 1 \end{aligned}$$

$$A(j) \;=\; j^{f(d)} - \sum_{j'=1}^{j-1} \binom{j}{j'} A(j') \qquad (29)$$

i.e. the number of strings of length $f(d)$ that contain each of the j variables can be computed as follows:
First compute the number of all possible strings of length $f(d)$ which contain some or all of variables $V_{l1}, V_{l2}, ...V_{lj}$. Then subtract the number of strings containing less than j distinct variables. The number of strings containing $j' < j$ distinct variables is $\binom{j}{j'}A(j')$. Each such string of length $f(d)$ has a probability of $\frac{1}{N^{f(d)}}$. Thus, the probability that a string of length $f(d)$ consists of each of the variables $V_{l1}, V_{l2}, ...V_{lK}$ can be given by:

$$P(V_{l1}, V_{l2}, ...V_{lK}) = \frac{A(K)}{N^{f(d)}} \qquad (30)$$

There are $\binom{N}{K}$ ways of choosing K distinct variables. Thus,

$$P(K) \;=\; \binom{N}{K}\frac{A(K)}{N^{f(d)}} \qquad (31)$$

Using the recurrence for $A(j), 1 \leq j \leq N$ from Eqn. 29, $P(K)$ can be computed.
Next, we evaluate the probability that K' sub-queries are resolved given

1. answers to K distinct values are gained from the $f(d)$ sensors (let us call this the event K as before)
2. I sub-queries are currently unresolved (again, let us call this the event I)

We denote this probability by $P(K'|I, K)$. Let $K'|I = A$, and $K = B$. Now, $B = \cup_{j=1}^{\binom{N}{K}} B_j$ where B_j is the event of getting a certain set of K (out of N) distinct values from the $f(d)$ sensors such that:

1. B_js are mutually exclusive. i.e $P(B_i \cap B_{i'}) = 0$, for $i \neq i'$.
2. $P(K) = P(B) = \sum_{j=1}^{\binom{N}{K}} P(B_j)$.

Also, $\forall j, 1 \leq j \leq \binom{N}{K} : P(B_j) = \frac{A(K)}{N^{f(d)}}$ (from Eqn. 30)

$$\begin{aligned} P(A|B) &= \frac{P(A \cap B)}{P(B)} \\ &= \frac{P(A \cap (\cup_{j=1}^{\binom{N}{K}} B_j))}{P(K)} \\ &= \frac{P(\cup_{j=1}^{\binom{N}{K}} A \cap B_j)}{P(K)} \end{aligned}$$

$$
\begin{aligned}
&= \frac{\sum_{j=1}^{\binom{N}{K}} P(A \cap B_j)}{P(K)} \\
&= \frac{\sum_{j=1}^{\binom{N}{K}} P(A|B_j)P(B_j)}{P(K)} \qquad (32)
\end{aligned}
$$

Now, $P(A|B_j)$ i.e. $P(K'|I, B_j)$ is the probability that K' sub-queries (out of the I) are resolved given a particular set B_j of K distinct values obtained from the $f(d)$ sensors.

$$
\forall j, 1 \leq j \leq \binom{N}{K} : P(A|B_j) = P(K'|I, B_j) = \frac{\binom{K}{K'}\binom{N-K}{I-K'}}{\binom{N}{I}} \qquad (33)
$$

i.e. given K distinct answers from the $f(d)$ sensors, K' (out of the I) sub-queries can be resolved iff the current query consists of some K' variables chosen from the K variables and $I - K'$ variables chosen from the remaining $N - K$ variables. The former can be chosen in $\binom{K}{K'}$ ways and the latter in $\binom{N-K}{I-K'}$ ways. However, the total number of ways of choosing I variables from N is $\binom{N}{I}$, thus giving the required probability. Thus,

$$
\begin{aligned}
P(A|B) &= \frac{\binom{K}{K'}\binom{N-K}{I-K'}}{P(K)\binom{N}{I}} \sum_{j=1}^{\binom{N}{K}} P(B_j) \\
&= \frac{\binom{K}{K'}\binom{N-K}{I-K'}}{P(K)\binom{N}{I}} P(K) \\
&= \frac{\binom{K}{K'}\binom{N-K}{I-K'}}{\binom{N}{I}}
\end{aligned}
\qquad (34)
$$

Thus,

$$
P(K'|I, K) = \begin{cases} \frac{\binom{K}{K'}\binom{N-K}{I-K'}}{\binom{N}{I}} & \text{if } K' \leq K, 0 \leq I - K' \leq N - K \\ 0 & \text{otherwise} \end{cases} \qquad (35)
$$

Thus,

$$
\begin{aligned}
P(K'|I) &= \sum_{l=K'}^{f(d)} P(K'|I, l) \times P(l) \\
&= \sum_{l=K'}^{f(d)} \frac{\binom{l}{K'}\binom{N-l}{I-K'}}{\binom{N}{I}} \times \frac{A(l)}{N^{f(d)}} \quad \text{(from Eqn.31)} \qquad (36)
\end{aligned}
$$

$P(K'|I)$ gives the transition probability from state I to state $I - K'$. Using the above expression, the state transition matrix $\mathbb{Q}$ for the Markov Chain can be calculated. Let

$S_i, 1 \leq i \leq M$ be the mean number of steps to absorption from state i. Then the S_is can be calculated as follows:

$$(\mathbb{I} - \mathbb{Q})S = \mathbb{E} \tag{37}$$

where $\mathbb{I}$ is an $M \times M$ Identity matrix, S is a $M \times 1$ column matrix and $\mathbb{E}$ is a $M \times 1$ column matrix of ones. S_M will give the mean number of steps to absorption from state M i.e. the mean number of steps to answer a query consisting of M sub-queries.