12 by 1

Mathematical Techniques for Optimizing Data Gathering in Wireless Sensor Networks

by

Narayanan Sadagopan

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

August 2005

Dedication

To my family,

for their love and support.

Acknowledgments

I have thoroughly enjoyed my stay as a graduate student at the University of Southern California (USC). Being in the company of extremely bright and talented people has taught me the essential ingredients of success in professional/academic life: hard work, dedication, collaboration and above all humility and compassionate behavior.

I am lucky to have had the opportunity to work with Prof. Bhaskar Krishnamachari and Prof. Ahmed Helmy as my advisors. Bhaskar has been a tremendous motivating influence, with his un-ending enthusiasm for solving problems of practical import. He has taught me the importance of intuition and the need to constantly improve oneself (professionally as well as personally). My tutelage under him has led to improvement in several areas including analytical skills, presentation skills, awareness of ethical behavior and piano skills (courtesy Bhaskar's approval of my classes in the music school). My Ph.D. study at USC began under the guidance of Ahmed during the Summer of 2002. He has been a source of constant guidance throughout my Ph.D. at USC, giving me valuable feedback on my work. I am very glad to have collaborated with him and Fan in IMPORTANT and PATHS, studies that initiated me to graduate research, but are not part of this dissertation. These studies taught me the importance of realizing the immediate needs

of a research community and working towards making an impact by addressing those needs. Most importantly, both Bhaskar and Ahmed have been “friend(s), philosopher(s) and guide(s)” in the true sense. I am also thankful to Prof. Ramesh Govindan for his valuable feedback during the formative stages of this dissertation which helped me tie up a few loose ends.

As I started my Ph.D. study at USC, I also had the opportunity to work with Prof. Leonard Adleman as a teaching assistant for the undergraduate course on algorithms. Len has been a great inspiration, with his approach of solving complex problems from first principles. His simple yet elegant approach rekindled my interest in algorithms and mathematics. My interest in these areas were further stimulated by the advanced algorithms courses offered by Prof. Ashish Goel at USC, leading me to pursue analytical problems in networking for my Ph.D.

My stay at USC began in Fall 1999 under the guidance of Prof. Jeff Rickel and Prof. Maged Dessouky, as a student interested in AI research. Working on the VFTS was personally fulfilling as it led to a web based intelligent agent that was used by several schools for their undergraduate course in manufacturing. Both Jeff and Maged supported and encouraged my application to the Ph.D. program, for which I will always be indebted to them.

I am also deeply thankful to different funding sources such as the Computer Science department (Teaching Assistantship) and National Science Foundation (Research Assistantship) that have financially supported me throughout my graduate study.

Graduate study would not have been intellectually stimulating but for an excellent set of fellow students at USC. Group meetings, paper readings and collaborations helped me to constantly learn new concepts. Several conference and journal publications pertaining to the material in this dissertation resulted from collaborations with Bhaskar, Ahmed and Mitali. These collaborations have significantly helped me in improving the quality of the included material. Outside the scope of this dissertation, I have also had the opportunity to closely work with several students (and friends) such as Fan and Gang. Both Fan and I started at USC in Fall 1999. We started collaborating during the inception of IMPORTANT in Summer 2001. Fan has been a great friend, who was always been there be it for coffee breaks or staying long nights during IMPORTANT and PATHS. Working in both ANRG and AGROUP has given me some wonderful friends like Jabed, Marco, Sundeep, Kiran, Avinash, Karim, Shao-Cheng, Shyam, Yang, people with whom I have spent most of my time as a Ph.D. student. Outside these research groups, I have had the opportunity to make many friends at USC who have been around during several important milestones in my graduate study. It would be impossible to list all of them, but there are a few whose association I would always cherish: Abhimanyu (for our mutual sharing of thoughts on various subjects including research), Debo (for being among the first of our batch to graduate and showing the way) and his wife Vidhya, Arun and Girish (for making life exciting and entertaining with their pranks), Krishna (for being an eternal room-mate), Ifti (for making the dull afternoons interesting with discussions on maths and algorithms), Abhinav (for the morning tea and piano sessions), Utham (for

those occasional contemplative evenings), our neighbors Sameera, Pranjali, Deepa at Vista Magnolia (for the fun discussions and parties), the younger lot of graduate students Bhavin, Apoorva (for their un-ending enthusiasm) and many more.

Looking back, I realize that I have come a long way during my graduate study at USC, working on agent based systems for a couple of years, dabbling in mobile ad hoc networks for a year, before deciding to pursue my dissertation on designing optimal data gathering algorithms for wireless sensor networks. This journey has been made possible by a wonderful and loving family to whom I dedicate this dissertation. From a young age, my parents (*amma* and *appa*) have taught me the importance of good virtues and dedication towards achieving one's goals. My motivation to pursue a Ph.D. came from my uncle (*mama*) and brother (*anna*) who are among the first in our family to achieve this great honor. *Anna*'s family has given me the much needed breaks and a feeling of home away from home for several summers. The last one year has been truly memorable, courtesy my life-partner Anitha. With her around, I did not realize how quickly time passed by. Her gentle, calming influence has helped me to remain focused during these crucial final stages of the dissertation.

Narayanan Sadagopan

Los Angeles, California

June 9, 2005.

Contents

List of Figures

Abstract

Recent technological advances in miniaturization and chip design have led to the ubiquity of small, low cost devices capable of sensing, computation and wireless communication. This has resulted in the emergence of a novel networking paradigm where a network of several wireless sensors interacts with the physical world. This network senses interesting events and collects data about them, making it similar to a distributed database. Developing data gathering algorithms is hence one of the most widely studied topics in sensor network research. The limited energy of the sensor nodes, their autonomous mode of operation and highly dynamic environmental conditions makes the design of these mechanisms/algorithms extremely challenging.

Several algorithms proposed for gathering data from a sensor network are mainly based on intuition or heuristic approaches. In this thesis we argue that the energy constraints of the sensor nodes engender the need for designing these algorithms in a principled manner using mathematical modeling and optimization techniques. We study complementary aspects of data gathering which include one-shot data gathering, *en masse* data gathering and continuous data gathering. While these studies are independent and involve the use of varied mathematical tools including first order models, linear program

duality and game theory, they illustrate the advantage of using the analytical approach by outlining scenarios where traditional solutions to these data gathering problems are not optimal.

Chapter 1

Introduction

The availability of small devices that are capable of limited computation, wireless communication, and sensing has extended the applicability of networking technology to previously unchartered territory of monitoring and instrumenting the physical world. A network of several such sensor devices is intended to have a broad range of environmental sensing applications including weather data-collection, vehicle tracking, habitat monitoring [14] [56] [41] [7] [15]. Several real world deployments of these wireless sensor networks are described in [58]. One of the first deployments was at the Great Duck Island, about 50 miles off the coast of Maine [53]. The initial deployment consisted of 50 sensor nodes, which was later increased to 100 nodes [52]. This network was used for monitoring the breeding activities of *petrel*, which is perhaps the most common and elusive seabird in the western North Atlantic. Monitoring *petrel* or any wildlife activity should be done with utmost care as any disturbance could lead to a possible abandonment of the habitat. Scientists are often faced with the dilemma of wanting to collect as much data as possible about the wildlife without disrupting their activities. Wireless sensing

has offered a possible solution to this dilemma. The scientists have to enter the habitat at the beginning of the study to deploy the sensors at points of interest (such as burrows, in this case). These sensors are capable of measuring a wide range of physical quantities such as temperature, humidity, pressure, etc. Once the nodes are deployed and organized into a network, the sensor network can collect and store the desired measurements and transmit data to a central computer whenever necessary. This data can then be relayed by the central computer (sink) to other parts of the world through the Internet and satellite connectivity. A wireless sensor network can thus act an a great enabling technology for other similar applications as acknowledged by the Great Duck Island project [53]:

"In theory, much if not all monitoring work can then be done without any human presence on the island. This capability has enormous potential for conservation efforts in small, isolated locations where any human presence is likely to be disruptive, or with species that are particularly sensitive to disturbance. It could also permit re-assessment of current or prior studies in order to determine the effects of disturbance on results."

By virtue of its use in remote locations, it is desirable that the sensor network operates autonomously in an uncontrolled environment. This task becomes extremely challenging in the face of limited battery power (life) of the sensors. Moreover, future sensor network deployments are envisioned to have a few hundreds of these 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 [20] [3] [2] [57]. As in the Great Duck Island deployment, there may be a central data sink (or a collection of sinks) which issues requests for data and the network responds to them.

For example, a sample query for the Duck Island application might be "Find all locations where temperature is greater than 70 degrees and humidity less than 50%". Such a query can be responded in two steps: the first step consists of flooding the query in the network, while the second step consists of the relevant nodes forming a tree and sending their responses to the sink. These scenarios motivate the use of "data-centric" routing in sensor networks i.e. routing for named data (such as temperature, humidity, etc.) as opposed to routing for end to end addresses (address centric routing).

In the context of sensor network applications, queries can be classified in many ways:

- **Continuous versus one-shot queries**: depending on whether the queries are requesting a long duration flow or a single datum.

- **Simple versus complex queries**: complex queries are combinations that consist of multiple simple sub-queries (e.g. queries for a single attribute type); e.g. "what are the location and temperature readings in those nodes in the network where (a) the light intensity is at least w and the humidity level is between x and y OR (b) the light intensity is at least z." Complex queries may also be aggregate queries that require the aggregation of information from several sources; e.g. "report the average temperature reading from all nodes in region R1".

- **Queries for replicated versus queries for unique data**: depending on whether the queries can be satisfied at multiple nodes in the network or only at one such node.

- **Queries for historic versus current/future data**: depending on whether the data being queried for was obtained in the past and stored (either locally at the same node or elsewhere in the network), or whether the query is for current/future data. In the latter case data does not need to be retrieved from storage.

A querying or routing algorithm for a sensor network involves the design of how the sink should request for the required information and how the network should respond to it. This design is made extremely challenging because of the following defining characteristics of sensor networks:

1. Energy Constraints: The sensor nodes are small devices with limited battery. While computation and wireless communication are sources of energy drain, the latter consumes more energy than the former. Thus it is desirable to design simple localized algorithms that do not involve a huge communication overhead. The energy constraints also limit the data traffic and the lifetime of the network.

2. Autonomous Operation: These networks are envisioned to operate in highly dynamic environments. Thus decentralized online algorithms are preferred over centralized offline algorithms. These algorithms should adapt to dynamics in network connectivity (due to failed or energy depleted nodes) and energy levels of the nodes.

3. Spatial Correlations: Since these networks monitor physical phenomena, data from nearby sensors tend to be correlated (or similar). The number of transmissions can be potentially reduced by exploiting correlations, resulting in energy savings. Routing algorithms should thus leverage the existence of spatial correlations (whenever possible) while constructing routes.

The finite energy constraints and the autonomous mode of operation emphasize the need for conserving the energy of these networks while still achieving the desired functionality of gathering data. Hence in this study, we use mathematical modeling and optimization techniques to analyze a few practical problems that arise in the context of data gathering in a sensor network. Using these techniques, several useful design insights are gained. Specifically, the analytical approach enables us to outline scenarios in which traditional solutions to these problems do not yield optimal solutions. This observation motivates the proposed thesis:

Severe energy constraints in sensor networks engender the need for a principled approach to system design based on sound mathematical modeling and optimization.

In this dissertation, we support this thesis through the following case studies that explore complementary aspects of data gathering in sensor networks:

- One-shot data gathering.
- *En masse* data gathering.
- Continuous data gathering.

For one-shot data gathering where data is replicated, we propose the ACQUIRE (ACtive Querying In sensoR nEtworks) mechanism [47] [48]. In ACQUIRE, a query consisting of several sub-queries is processed by a sequence of local updates within d hops, followed by query forwarding. The look-ahead parameter d enables ACQUIRE to span the space from a Trajectory-Based Querying (TBQ) technique (when d=0) to a Flooding-Based Querying (FBQ) technique (when d = D, the network diameter). Using simple first order analysis, we show that the optimal value of the look-ahead parameter d for ACQUIRE (that minimizes the number of transmissions) depends only on the query rate and the data dynamics. For a given query rate, a large d (similar to FBQ) is suitable for low data dynamics, while a small d (similar to TBQ) is suitable for high data dynamics. The first order analysis yields a common framework for comparing ACQUIRE, FBQ, Expanding Ring Search (ERS) and random walk (random TBQ). It is seen that ACQUIRE (with optimal look-ahead) can give up to 60% improvement in energy (number of transmissions) over ERS and orders of magnitude energy savings over FBQ, at the cost of some increase in delay. The first order analysis is also validated by a detailed Markov chain analysis and simulations.

For applications involving environmental monitoring in remote areas such as planetary explorations and remote surveillance, it might not be possible or necessary for a user to query the network periodically. In such cases, information from the entire sensor network might have to be extracted "en masse" after a prolonged period of sensing. We formulate the problem of extracting the maximum possible data from the network as a multi-commodity flow problem [45] [46]. We present an iterative $(1 + \omega)$ approximation algorithm that is based on an algorithm presented previously by Garg and Konemann [19]. As a practical implementation, we use insights obtained from this centralized algorithm to develop a distributed fast greedy heuristic called E-MAX which uses the exponential metric (that is both "energy-aware" and "data-aware") based on the approximation algorithm. Through simulations, we show that the greedy heuristic incorporating the exponential metric performs near optimally (within 1 - 20% of the optimal) with significantly lower overhead compared to other energy aware shortest path routing approaches particularly when nodes are heterogeneous in their energy and data availability.

To construct balanced data gathering trees for continuous queries, we develop decentralized utility-based mechanisms (inspired by game theory) to optimize routing in sensor networks [44] [49]. Traditionally, game theory views the agents involved as being inherently self-interested and investigates the strategies they should adopt when maximizing their own payoff. At first glance, it would appear that such a theory would be ill-suited to sensor networks where it is assumed that each node functions towards a system-wide

goal rather than being selfish. Indeed, the Nash Equilibrium of a game is not always globally efficient. Our motivation for exploring the use of game theoretic mechanism design is to use this formalism to enable decentralized decision making. It would be of interest to see if networks can be designed to meet a specific global objective, given that each sensor is selfish. The design of the network would then involve choosing local utility functions for each sensor, such that when all sensors act selfishly using only local information exchanges, they end up maximizing the system-wide global objective. Specifically, we design local utility functions for constructing load balanced and energy balanced data gathering trees.

These case studies are carefully chosen to cover a wide range of problem formulations with noticeably different characteristics (i.e. different objective functions and parameters). This leads to the use of different mathematical tools for analyzing these problems. For example, for one-shot data gathering, it is of interest to minimize the average energy consumed to answer a given query. This in turn depends on the average number of update steps needed to completely resolve the query, which can be determined by modeling the query resolution as a probabilistic process (Markov chain). In our second case study on "en masse" data gathering, the objective is to maximize the amount of data extracted (under the constraints of limited energy and data levels at the nodes), which naturally lends itself to be modeled as a multi-commodity flow problem. For our third case study on constructing balanced trees for continuous data gathering, we present a novel approach to designing distributed algorithms for sensor networks by modeling

an inherently co-operative sensor network as a network of "selfish" sensors (where the utility functions of these sensors are designed with respect to a desired global objective). Choosing an appropriate analytical tool for a problem is an art in itself and can only be perfected by practice.

The rest of this dissertation is organized as follows: chapter 2 discusses the related work, chapter 3 describes one-shot data gathering, "en masse" data gathering is described in chapter 4, while chapter 5 describes continuous data gathering. The contributions of this dissertation are presented in chapter 6, while a brief overview of the analytical tools used in the dissertation is given in chapter 7.

Chapter 2

Related Work

The nature of sensor network applications and the severe energy constraints on these sensor nodes have given rise to several interesting system design principles for routing. In this section, we give a very broad classification of these principles.

2.1 Data-Centric Routing

As mentioned in Chapter 1, in several sensor network applications, queries and responses are for named data rather than end to end addresses. Bonnet, Gehrke and Seshadri [3], [2] as well as Yao and Gehrke [57] 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 *et al.* also argue that sensor networks ought to be viewed primarily as virtual databases, with query optimization performed via data-centric routing mechanisms within the network [20]. The efficient in-network

computation of aggregate responses to queries is the subject of the paper by Madden *et al.* [34].

Intanagonwiwat *et al.* propose and study Directed Diffusion, a data-centric protocol that is particularly useful for responding to long-standing (or continuous) queries [27] [26]. 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. When the queries are for truly long term continuous flows, the cost of the initial querying may be relatively insignificant, even if that takes place through naive flooding (as for instance, with the basic Directed Diffusion). However, when they are for one-shot data, the cost and overhead of flooding can be prohibitively expensive. Similarly, if the queries are for replicated data, a flooding may return multiple responses when only one is necessary. Thus other alternatives to flooding-based queries (FBQ) are clearly desirable. The interest-flooding cost for Directed Diffusion can be reduced by using Rumor-routing proposed by Braginsky and Estrin [4]. It adopts an interesting approach - sources with events launch mobile agents which execute random walks in the network resulting in event-paths. The queries issued by the querier/sink 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. Another routing scheme related to Rumor-routing is the Comb Needle technique proposed by Liu, Huang and Zhang [32]. In this case the queries build a horizontal comb-like routing structure,

while events follow a vertical needle-like trajectory to meet the "teeth" of the comb. A key tunable parameter in this construction is spacing between branches of the comb and correspondingly the length of the trajectory traversed by event notifications, which can be adjusted depending on the frequency of queries as well as the frequency of events. To minimize the average total cost per query, the comb inter-spacing as well as the length of the event trajectories should be smaller when the event to query ratio is higher; on the other hand when the event to query ratio is lower, the comb inter-spacing as well as the distance traversed by even notifications should be higher. When the frequency of queries is more than the frequency of events, and there are multiple querying nodes, then a *reverse-comb structure* can be employed, in which the events form the vertical comb while queries take horizontal needle trajectories to match with events.

Several techniques have been proposed for guiding a query in a sensor network. Nath and Niculescu propose Trajectory-Based Querying (TBQ) in which a query agent is routed along a trajectory that could be directed and deterministically [36]. Information Driven Sensor Querying (IDSQ) and Constrained Anisotropic Diffusion Routing (CADR) mechanisms proposed by Chu, Hausseker and Zhao utilize the spatial characteristics of the observed phenomena to optimize query routing [12]. 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.

Instead of guiding the query along a trajectory, another option is the use of an expanding ring search from the sink. An expanding ring search proceeds as a sequence of controlled floods, with the radius of the flood (i.e. the maximum hop count of the flooded packet) increasing at each step if the query is not resolved at the previous step. The choice of the number of hops to search at each step is a design parameter that can be optimized to minimize the expected search cost using a Dynamic Programming technique [10]. However, in the absence of replicated/cached information in the network, for arbitrarily located data, expanding ring searches do not provide useful gains over flooding. Cheng and Heinzelman show that less than 10% energy savings are obtained from expanding ring search compared to flooding when there is no caching/replication of data in the network, while the delay increases significantly [11]. Intuitively, when there is frequently cached information, expanding ring searches are more likely to offer significant improvements because the likelihood of resolving the query earlier would be higher.

Data-centric routing (resource discovery) has also been explored in the context of mobile ad hoc and peer-to-peer (P2P) overlay networks on the Internet. Helmy *et al.* propose CARD (Contact Based Architecture for Resource Discovery), a novel architecture for resource discovery in large-scale mobile ad hoc networks [24]. Lv *et al.* discuss the possibility of launching k-random walks through the unstructured P2P network for discovering required files/data [33]. However there are some subtle differences in these overlay networks and sensor networks. One of them is that the cost model is different –

in P2P networks one is primarily concerned with minimizing bandwidth usage and delay while we are primarily concerned with minimizing energy consumption; the second trajectories followed by active queries need not necessarily be random walks, they could be directed.

In chapter 3 of this thesis, we propose ACQUIRE, a novel data-centric routing scheme for answering one-shot queries for replicated data. In ACQUIRE, a query is processed by a series of local floods within d hops followed by query forwarding. As mentioned in chapter 1, we show that the optimal value of d (that minimizes the number of transmissions) is governed by the query rate and data dynamics. For a given query rate, a low value of d (similar to TBQ) is optimal for high data dynamics. On the other hand, a high value of d (similar to FBQ) is optimal for low data dynamics.

2.2 Data-Centric Storage

One alternative to eliminate the flooding phase in certain data-centric routing protocols described earlier is to store data within the sensor network in a structured manner. This is achieved by Data-Centric Storage (DCS), proposed by Ratnasamy *et al.*, in which sensed data are stored at a node determined by the name associated with the data [43]. This approach is particularly useful for storing information to deal with historic queries (i.e. queries for non-current data). Through simple first order analysis, it is shown that DCS is preferable in cases where the sensor network is large and there a many detected events and not all of them are queried for. They propose Geographic Hash Table (GHT), a

DCS system that hashes event types into geographic co-ordinates and stores the relevant data at the node closest to that location. While the initial incarnation of GHTs require the nodes to know their geographic location, the routing scheme proposed by Rao *et al.* that overlays a virtual co-ordinate system can be used when the perimeter nodes know their location information but the other nodes do not [42]. Gao *et al.* propose the fractional cascading information storage for answering range queries in sensor networks [18]. A sample class of range queries they consider is of the form (q, R, T), where q is the querying sensor, R is a rectangular region in the deployment area and T is the temperature range. To answer such queries efficiently, they propose to overlay a quad tree over the topology where each sensor stored summarized information about siblings of nodes that are on its path to the root of the quad tree. In a dense network of n nodes, each sensor thus stores $O(\log n)$ information. Moreover, this storage scheme makes close by sensors to have highly correlated world views, leading to smooth information gradients. Their localized algorithms for answering range queries are shown to be optimal with respect to the communication overhead.

2.3 Energy-Aware Routing

The limited energy of the sensors motivate the need for developing routing and querying algorithms that are energy efficient. Energy-aware routing (that dynamically adapts to changing energy levels of the nodes) significantly improves lifetime of wireless ad hoc networks [51] [54]. The definition of network lifetime is quite application specific. A

few examples of these definitions are: time till network disconnection, time till first (last) node is energy depleted, etc.

Heinzelman *et al.* propose a family of adaptive protocols called SPIN for energy efficient dissemination of information throughout the sensor network [23]. These protocols actively monitor the current sensor resources and the nature of the data so as to efficiently utilize the scarce energy resources. Heinzelman *et al.* propose LEACH, a scalable adaptive clustering protocol in which nodes are organized into clusters and system lifetime is extended by randomly choosing the cluster-heads [22]. Each cluster-head in LEACH then sends the aggregated data directly to the base station, although this communication can be very expensive in large networks. Lindsey and Raghavendra propose an alternative data gathering scheme called PEGASIS [31]. In this scheme the nodes organize themselves in a chain to communicate data to neighbors and a sensor nodes is elected to send all the data to the base station. Lifetime is again improved by changing the elected node. Lindsey, Raghavendra and Sivalingam indicate that delay might also be an important metric in addition to lifetime and study data gathering schemes that explore the trade-off between energy consumed and delay incurred [30].

While the above studies propose heuristics to extend network lifetime, other studies adopt a more formal approach by using Linear Program (LP) formulations to maximize the network lifetime. Chang and Tassiulas formulate the global problem of maximizing the network lifetime with known origin destination flow requirements as a LP, and propose a flow augmentation heuristic technique based on iterated saturation of shortest-cost

paths to solve it [9]. The basic idea is that in each iteration, every origin node computes the shortest cost path to its destination, and augments the flow on this path by a small step. After each step the costs are recalculated, and the process repeated until any node runs out of its initial energy E_i. Chang and Tassiulas also propose an approximation algorithm that gives a $(1 + \omega)$ approximation ratio for maximizing the life time of a network [8]. This algorithm is based on the approximation algorithm developed by Garg and Konemann for multi commodity flows [19]. Bhardwaj and Chandrakasan use LP based on network flows to examine feasible role assignments (FRA) of nodes as a means of maximizing the lifetime of aggregating as well as non-aggregating sensor networks [1]. Kalpakis, Dasgupta and Namjoshi examine the MLDA (Maximum Lifetime Data Aggregation) problem and the MLDR (Maximum Lifetime Data Routing) problem, formulating it using multi-commodity network flows [28]. They use the solution obtained by solving the LP to construct a schedule of data gathering to optimize the network life time. They observe that as the network size increases, solving the LP takes considerable time. Hence they propose some clustering heuristics to achieve near-optimal performance. Gandham, Dawande and Prakash use LP flow based formulations to propose algorithms for the following problems that arise in collecting data from sensors at one or more base stations: a) minimizing the total energy and b) minimizing the total energy while minimizing the maximum energy spent by a node [17].

A novel definition of network life time is the "maximum node lifetime curve" proposed by Brown, Gabow and Zhang [5]. Informally, the maximum node lifetime curve

attempts to maximize the time until a set of nodes drain their energy (at a "drop point" say) while minimizing the number of nodes that drain their energy at each "drop point". They formulate the maximum node lifetime curve as an LP. They propose an algorithm that solves the LP to obtain the flow to be routed on each path to maximize the node lifetime curve. The running time of this algorithm is further improved upon by Hou, Shi and Sherali [25].

In chapter 4 of this thesis, we examine the maximum data gathering problem where data is gathered "en masse" from a sensor network. This problem is analogous to the problem of maximizing the life time of a network where data is gathered continuously at some desired frequency. We show that the "en masse" data gathering problem introduces "data-awareness" in addition to "energy-awareness" as an important design consideration for routing.

2.4 Utility-Based Routing

Due to the autonomous and unattended mode of operation of a sensor network, it is desirable to design routing algorithms that are decentralized. Quite a few studies have used the concepts of utility functions for this purpose. This approach was first used by Byers and Nasser for constructing a data aggregation tree in a sensor network [6]. For energy efficiency considerations, they allow for some sensors to be in a sleep state and not perform any sensing. This notion is captured by a global objective function that is concave in the number of nodes involved in the sensing operation at each round. They

then propose a distributed algorithm in which each sensor node decides whether it should be involved in the sensing operation at each round. This algorithm maximizes the total utility of the sensing operation over time. Kannan *et al.* propose a paradigm shift in routing for inherently cooperative sensor networks by treating each sensor as a selfish entity [29]. They suggest each sensor should make an "optimal" decision for itself rather than trying to attain global optimality. They use this approach to design algorithms for reliable data gathering. For each sensor, they propose a utility function that depends on the importance of its data and the reliability of its path to the sink. They define the desired optimal tree as the Nash Equilibrium of the interaction of these selfish sensors. It is shown that computing the Nash Equilibrium is NP-Hard in general, but outline conditions in which the Nash Equilibrium corresponds to commonly desired trees such as the most reliable tree or a shortest path tree. Similar game theoretic studies have been widely used to model the impact of social behavior on the Internet [39]. Attaining the desired global objective in the presence of selfish agents (whose utility functions are known beforehand) is dealt by Algorithmic Mechanism Design (AMD) [37]. This is usually done by offering incentives to the agents in the form of payments.

In chapter 5 of this thesis, unlike earlier studies that analyze the Nash Equilibrium resulting from the interaction of selfish agents (whose utility functions are known *apriori*), we examine the problem of designing utility functions for individual sensors

(agents) such that the resulting Nash Equilibrium coincides with a desired global objective. Specifically, we design these utility functions for two specific global objectives of load balancing and energy balancing.

Chapter 3

One-Shot Data Gathering

Depending on the applications, there are likely to be different kinds of queries in sensor networks. Consider the following examples:

- **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 *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 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?" into a single batched query. This 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).

Flooding-based query mechanisms such as the Directed Diffusion data-centric routing scheme [27] 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 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. [1]

Hence, we present a new data-centric querying mechanism, ACtive QUery forwarding In sensoR nEtworks (ACQUIRE) [47]. Fig. 3.1 shows the different categories of queries and the kinds of queries in sensor networks that ACQUIRE is well-suited for: one-shot queries for replicated data.

[1] In the above examples, there is an implicit assumption that the sub-queries can all be resolved. Without this assumption, it is not be possible to do anything more intelligent than querying all nodes in the network.

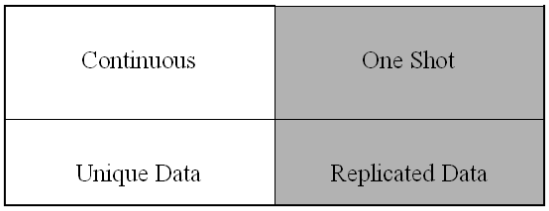


Figure 3.1: A categorization of queries in sensor networks: the shaded boxes represent the query categories for which the ACQUIRE mechanism is well-suited.

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 modeling approach that allows us 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 d and the ratio of update rate to query rate. Our mathematical analysis is validated through simulations.

The rest of this chapter is organized as follows: we provide a basic description of the ACQUIRE mechanism in section 3.1. In section 3.2 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 3.3. We first examine the impact of critical parameters on the energy cost of ACQUIRE and then compare it to the alternative approaches in section 3.4. Our analytical models are validated by simulations in section 3.5. The average response latency incurred by ACQUIRE, ERS and FBQ is analytically modeled in section 3.6, while the caching storage requirements are discussed in section 3.7. A discussion of our results is presented in section 3.8. Finally, section 3.9 summarizes this chapter.

3.1 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 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 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 3.2 and 3.3 respectively.

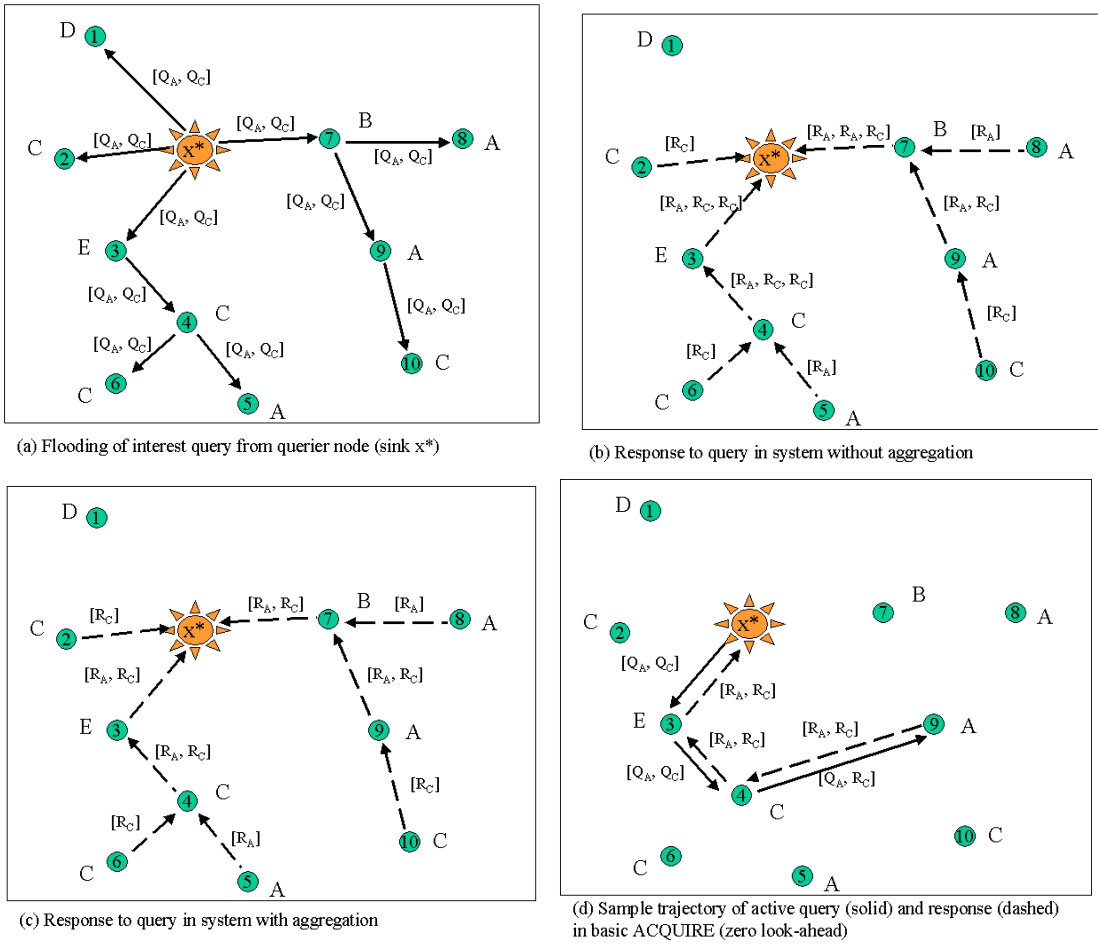


Figure 3.2: Illustration of traditional flooding-based queries (a), (b), (c), and ACQUIRE (d) in a sample sensor network.

3.2 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.

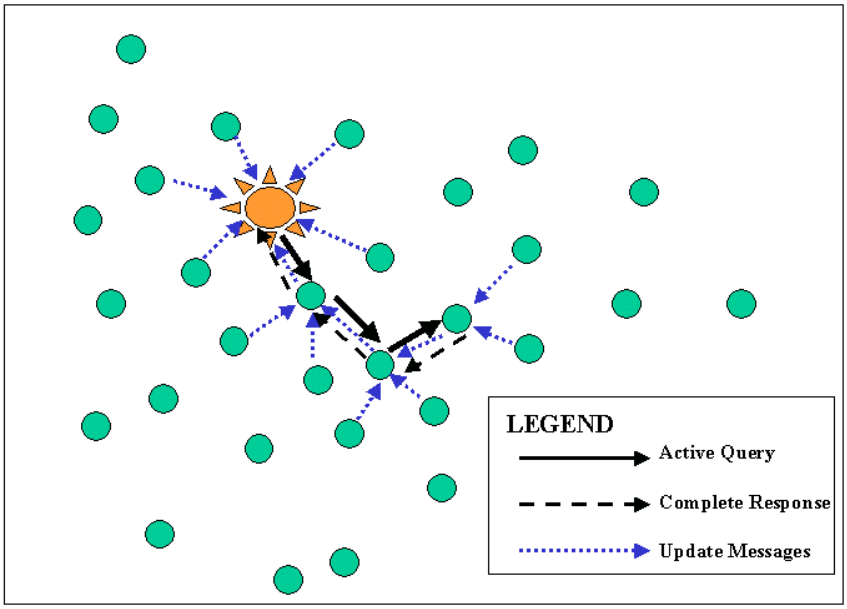


Figure 3.3: Illustration of ACQUIRE with a one-hop lookahead ($d = 1$). At each step of the active query propagation, the node carrying the active query employs knowledge gained due to the triggered updates from all nodes within d hops in order to partially resolve the query. As d becomes larger, the 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.

3.2.1 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.

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[2]. 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[3].

[2]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

[3]One way to think of the size of the 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 subqueries 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 sublinearly with M (as shown in section 3.2). 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).

Let 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 3.2.4.

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, $c = 0.01$[4].

[4]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.

After the query is completely resolved, the last node which has the query returns the completed response[5] to the querier x^* along the reverse path [6]. 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:

$$E_{avg} = (cE_{update} + d)S_M + \alpha \tag{3.1}$$

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]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.

[6]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.

3.2.2 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 $f(d)$.

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 sub-queries. Thus, if there are currently M sub-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 sub-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 fewer sub-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. $\sigma_M =$ 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,

$$\sigma_M = \sum_{i=1}^{M} X_i \tag{3.2}$$

By linearity of expectation,

$$E(\sigma_M) = \sum_{i=1}^{M} E(X_i) \tag{3.3}$$

$$E(\sigma_M) = N \sum_{i=1}^{M} \frac{1}{M-i+1} \tag{3.4}$$

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,

$$E(\sigma_M) \approx N(\ln M + \gamma) \tag{3.5}$$

Now, since we consider fetching information from $f(d)$ sensors as 1 trial (step) rather than $f(d)$ trials (steps)[7]:

$$S_M = \frac{E(\sigma_M)}{f(d)} \approx \frac{N(\ln M + \gamma)}{f(d)} \tag{3.6}$$

Eqn. 3.6 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 ($f(d)$).

[7]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. We will briefly discuss this issue again in section 3.8

To answer more complex questions like "what is the probability that at least one sub-query can be resolved in a single step?", we formulate the query forwarding process as a Markov chain in section 3.2.3.

3.2.3 Markov Chain Analysis

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 sub-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 forwarding process naturally lends itself to be modeled as a Markov chain, as shown in figure 3.4.

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 $M' < M''$

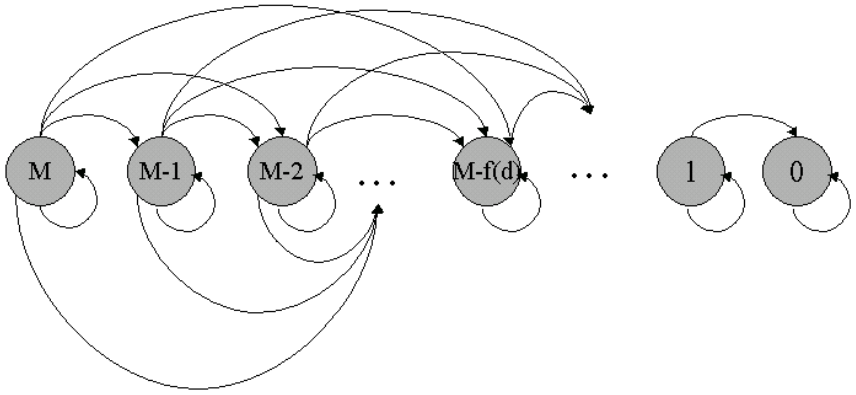


Figure 3.4: Markov chain with states representing the number of unresolved sub-queries of an active query. Transitions are only to lower-valued states at most $\min(f(d), M)$ to the right, and the chain terminates at the absorbing state 0 when all sub-queries have been resolved.

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,

$$\begin{aligned} A(0) &= 0 \\ A(1) &= 1 \\ A(j) &= j^{f(d)} - \sum_{j'=1}^{j-1} \binom{j}{j'} A(j') \end{aligned} \tag{3.7}$$

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:

$$P(V_{l1}, V_{l2}, ...V_{lK}) = \frac{A(K)}{N^{f(d)}} \tag{3.8}$$

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

$$P(K) = \binom{N}{K} \frac{A(K)}{N^{f(d)}} \tag{3.9}$$

Using the recurrence for $A(j), 1 \leq j \leq N$ from Eqn. 3.7, $P(K)$ 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. 3.8)

$$\begin{aligned} P(A|B) &= \frac{P(A \cap B)}{P(B)} \\ &= \frac{P(A \cap (\cup_{j=1}^{\binom{N}{K}} B_j))}{P(K)} \\ &= \frac{P(\cup_{j=1}^{\binom{N}{K}} A \cap B_j)}{P(K)} \\ &= \frac{\sum_{j=1}^{\binom{N}{K}} P(A \cap B_j)}{P(K)} \\ &= \frac{\sum_{j=1}^{\binom{N}{K}} P(A|B_j)P(B_j)}{P(K)} \end{aligned} \tag{3.10}$$

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.

$$\forall j, 1 \leq j \leq \binom{N}{K} : P(A|B_j) = P(K'|I, B_j) = \frac{\binom{K}{K'}\binom{N-K}{I-K'}}{\binom{N}{I}} \tag{3.11}$$

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,

$$\begin{aligned} P(A|B) &= \frac{\binom{K}{K'}\binom{N-K}{I-K'}}{P(K)\binom{N}{I}} \sum_{j=1}^{\binom{N}{K}} P(B_j) \\ &= \frac{\binom{K}{K'}\binom{N-K}{I-K'}}{P(K)\binom{N}{I}} P(K) \\ &= \frac{\binom{K}{K'}\binom{N-K}{I-K'}}{\binom{N}{I}} \end{aligned} \tag{3.12}$$

Thus,

$$P(K'|I, K) = \begin{cases} \frac{\binom{K}{K'}\binom{N-K}{I-K'}}{\binom{N}{I}} & \text{if } K' \leq K, 0 \leq I - K' \leq N - K \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

Thus,

$$
\begin{aligned}
P(K'|I) &= \sum_{l=K'}^{f(d)} P(K'|I,l) \times P(l) \\
&= \sum_{l=K'}^{f(d)} \frac{\binom{l}{K'}\binom{N-l}{I-K'}}{\binom{N}{I}} \times \frac{A(l)}{N^{f(d)}} \quad \text{(from Eqn.3.9)}
\end{aligned} \tag{3.14}
$$

$P(K'|I)$ gives the transition probability from state I to state $I - K'$. Using the above expression, the state transition matrix R 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:

$$(I - R)S = E \tag{3.15}$$

where I is an $M \times M$ Identity matrix, S is a $M \times 1$ column matrix and 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.

3.2.4 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,

$$E_{update} = (f(d-1) + \sum_{i=1}^{d} iN(i)) \tag{3.16}$$

where $N(i)$ is the number of nodes at hop i. $N(i)$ will be determined later in section 3.2.5.

3.2.5 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:

$$E_{avg} = (cE_{update} + d)S_M + \alpha \tag{3.17}$$

where α is the expected number of hops from the node where the query is completely resolved to the querier x^* [8]. This is the cost of returning the completed response back to

[8]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.

the querier node. This response can be returned along the reverse path in which case α can be atmost dS_M. Thus,

$$E_{avg} = (cE_{update} + 2d)S_M \tag{3.18}$$

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,

$$E_{avg} = 2E(\sigma_M) \tag{3.19}$$

3.2.6 Optimal Look-ahead

As mentioned in section 3.2.5,

$$\begin{aligned} E_{avg} &= \{(f(d-1) + \sum_{i=1}^{d} iN(i))c + 2d\}S_M \quad \text{(from Eqn.3.16 and 3.18)} \\ &\approx \{(f(d-1) + \sum_{i=1}^{d} iN(i))c + 2d\}\frac{N(\ln M + \gamma)}{f(d)} \quad \text{(from Eqn.3.6)} \end{aligned} \tag{3.20}$$

If we ignore boundary effects, it can be shown that $f(d) = (2d(d+1)) + 1$ for a grid[9]

Also,

$$\begin{aligned} N(i) &= f(i) - f(i-1) \\ &= 2i(i+1) - 2(i-1)i \\ &= 4i \end{aligned} \tag{3.21}$$

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

$$\begin{aligned} E_{avg} &\approx \{(2(d-1)(d) + 1 + \sum_{i=1}^{d} 4i^2)c + 2d\} \frac{N(\ln M + \gamma)}{(2(d)(d+1)) + 1} \\ &\approx \{(2(d-1)(d) + 1 + \frac{4}{6}(d)(d+1)(2d+1))c + 2d\} \frac{N(\ln M + \gamma)}{(2(d)(d+1)) + 1} \\ &\approx \{\frac{cN(\ln M + \gamma)}{3} \frac{4d^3 + 12d^2 - 4d + 3}{2d^2 + 2d + 1} + N(\ln M + \gamma) \frac{2d}{2d^2 + 2d + 1}\} \end{aligned} \tag{3.22}$$

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:

$$\begin{aligned} \frac{2}{3} \frac{(N \ln M + \gamma)(4cd^4 + 8cd^3 + 22cd^2 + 6cd - 5c - 6d^2 + 3)}{(2d^2 + 2d + 1)^2} &= 0 \\ 4cd^4 + 8cd^3 + 22cd^2 + 6cd - 5c - 6d^2 + 3 &= 0 \end{aligned} \tag{3.23}$$

[9]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. Ignoring this overlap does not affect our study as we will point out in section 3.8

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

If $d = 0$:

$$E_{avg} \approx 2N(\ln M + \gamma) \qquad \text{(from Eqn. 3.5 and 3.19)} \tag{3.24}$$

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

3.2.7 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. 3.22 derived in section 3.2.6. 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. 3.24 in section 3.2.6.

Figure 3.5 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 3.6 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

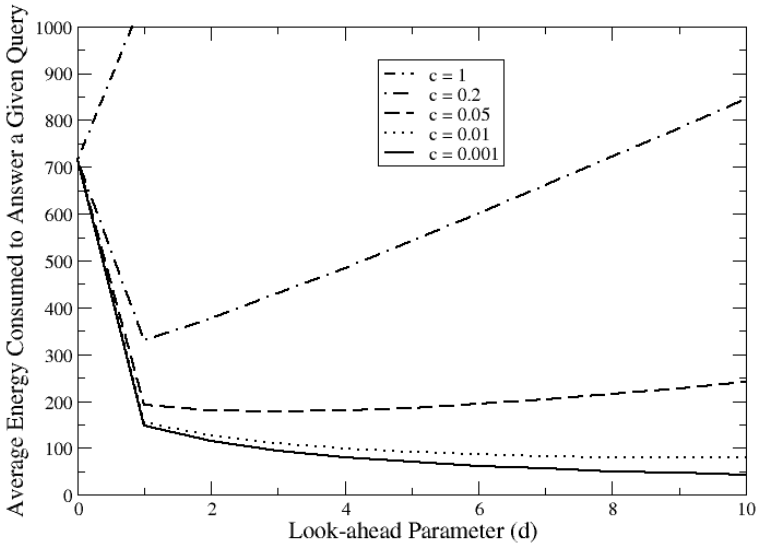


Figure 3.5: Effect of c and d on the Average Energy Consumption of the ACQUIRE scheme. Here, $N = 100$ and $M = 20$

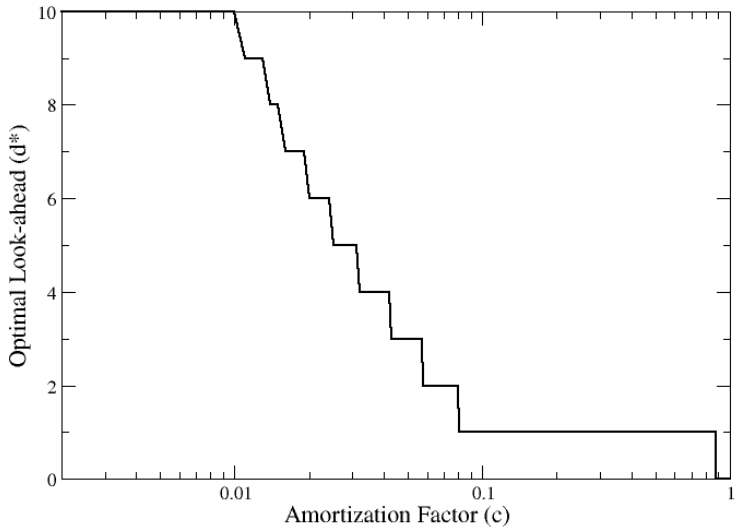


Figure 3.6: Effect of c on d^* for $N = 100$ and $M = 20$. The x-axis is plotted on a log scale.

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. $d = 0$).

3.3 Analysis of Alternative Approaches

In this section, we present the energy cost of expanding ring search and flooding-based query mechanisms.

3.3.1 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 3.2.2:

$$\begin{aligned}
\sum_{i=1}^{t_{min}} N(i) &= N(\ln M + \gamma) \qquad \text{(from Eqn.3.5)} \\
4\sum_{i=1}^{t_{min}} i &= N(\ln M + \gamma) \qquad \text{(from Eqn.3.21)} \\
2t_{min}(t_{min}+1) &= N(\ln M + \gamma) \\
2(t_{min})^2 + 2t_{min} - N(\ln M + \gamma) &= 0 \qquad (3.25)
\end{aligned}$$

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:

$$\begin{aligned}
E_{update} &= \sum_{i=1}^{t_{min}} (f(i-1) + iN_{avg}(i)) \\
&= \sum_{i=1}^{t_{min}} f(i-1) + \sum_{i=1}^{t_{min}} iN_{avg}(i) \qquad (3.26)
\end{aligned}$$

$N_{avg}(i)$ 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 sub-queries resolved $M_r(i-1)$ before the i^{th} step can be given as follows:

$$\begin{aligned} f(i-1) &= N(\ln(M_r(i-1)) + \gamma) \qquad \text{(from Eqn.3.5)} \\ M_r(i-1) &= e^{\frac{f(i-1)}{N} - \gamma} \end{aligned} \tag{3.27}$$

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

$$p_i = \frac{M - M_r(i-1)}{N} \tag{3.28}$$

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

$$\begin{aligned} N_{avg}(i) &= N(i)p_i \qquad \text{(substituting } p_i \text{ for } p \text{ in Eqn.3.31)} \\ &= N(i)(\frac{M - e^{\frac{f(i-1)}{N} - \gamma}}{N}) \end{aligned} \tag{3.29}$$

Since the query is not forwarded to any other node,

$$
\begin{aligned}
E_{avg} &= E_{update}c \qquad (\text{substituting } d=0,\ \alpha=0,\ S_M=1 \text{ in Eqn.3.17}) \\
&= (\sum_{i=1}^{t_{min}}(2i(i-1)+1) + \sum_{i=1}^{t_{min}} iN(i)(\frac{M - e^{\frac{f(i-1)}{N}-\gamma}}{N}))c \\
&= (\frac{1}{3}(t_{min})(t_{min}+1)(2t_{min}+1) - (t_{min})(t_{min}+1) + t_{min} + \\
&\quad \sum_{i=1}^{t_{min}} iN(i)(\frac{M - e^{\frac{f(i-1)}{N}-\gamma}}{N}))c \\
&= (\frac{2}{3}(t_{min})(t_{min}+1)(t_{min}-1) + t_{min} + \sum_{i=1}^{t_{min}} iN(i)(\frac{M - e^{\frac{f(i-1)}{N}-\gamma}}{N}))c
\end{aligned}
\tag{3.30}
$$

3.3.2 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. 3.17 from section 3.2.5,

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

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 3.2.2:
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

$$N_{avg}(i) \quad = \quad N(i)p \tag{3.31}$$

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:

$$\begin{aligned} E_{avg} &= (f(R) + \sum_{i=1}^{R} iN_{avg}(i))c \\ &= (f(R) + \sum_{i=1}^{R} iN(i)\frac{M}{N})c \\ &= (f(R) + \frac{M}{N}\sum_{i=1}^{R} iN(i))c \\ &= (2R(R+1) + 1 + \frac{2}{3}\frac{M}{N}R(R+1)(2R+1))c \end{aligned} \tag{3.32}$$

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

3.4 Comparison of ACQUIRE, ERS and FBQ

3.4.1 Effect of c

These schemes were analytically compared across different values of c chosen in the range of $[0.001, 1]$ as in section 3.2.7. 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. 3.30 from section 3.3.1, Eqn. 3.32 from section 3.3.2 and Eqn. 3.22 from section 3.2.6 (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. (3.25) from section 3.3.1, we obtain $t_{min} = 13$. This value of t_{min} is then used in Eqn. 3.30.

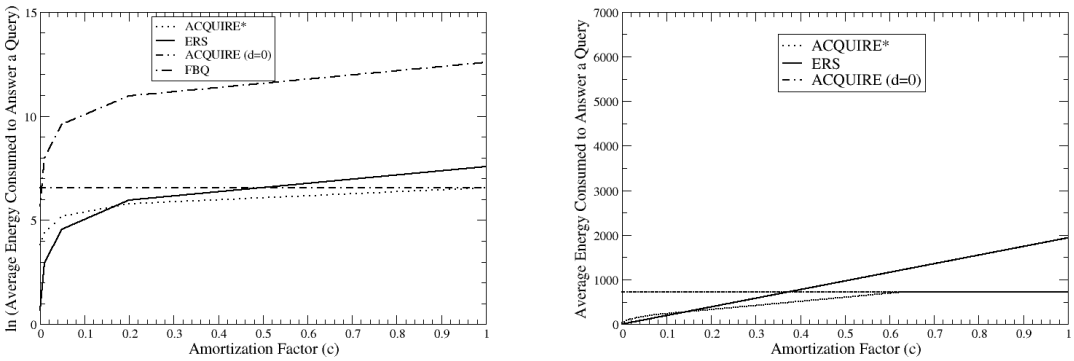


Figure 3.7: Comparison of $ACQUIRE^*$, ERS, ACQUIRE with $d = 0$ and FBQ with energy on a log-scale (left) and linear scale (right). ($N = 100$ and $M = 20$)

As figure 3.7 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 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 3.6.

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. 3.25). With these values of M and N as shown by the figure 3.8, $ACQUIRE^*$ outperforms ERS if $c \geq 0.06$ (approx.). In this case, $d^* = 2$ as shown by figure 3.6. Moreover in this case as figure 3.8 shows, for $c > 0.2$, even ACQUIRE with $d = 0$ would outperform ERS Again for $c = 1$, $ACQUIRE^*$ achieves a 75% energy savings over ERS.

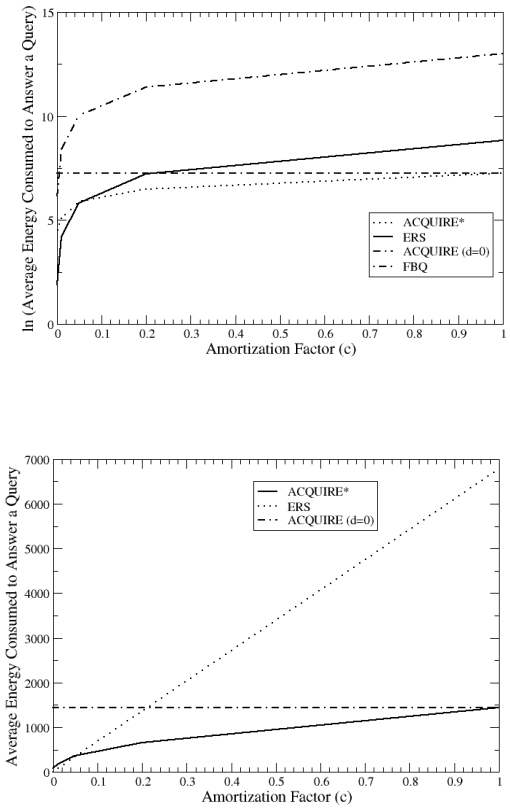


Figure 3.8: Comparison of $ACQUIRE^*$, ERS, ACQUIRE with $d = 0$ and FBQ with energy on a log scale (left) and linear scale (right). ($N = 160$ and $M = 50$)

As figure 3.7 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 3.4.2. 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.

3.4.2 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.

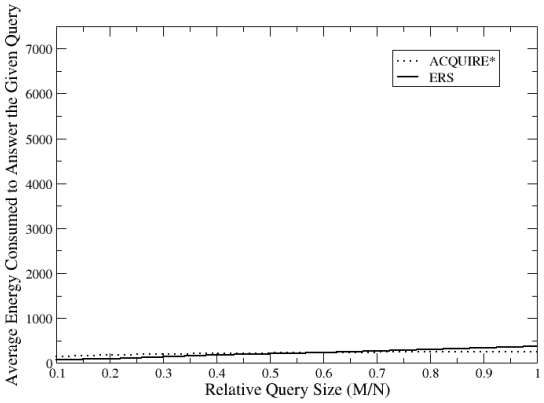
As the figures 3.9 shows, the average energy consumed to resolve a query increases with increasing M for a given N. For $c = 0.05$, ERS gives an almost 50% energy savings over $ACQUIRE^*$ when $\frac{M}{N} \leq 0.5$, while in the other cases $ACQUIRE^*$ outperforms ERS across all the amortization factors and relative query sizes in our study. In these cases, the energy savings of $ACQUIRE^*$ over ERS range as high as 85% ($c = 1$, $\frac{M}{N} = 1$).

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).

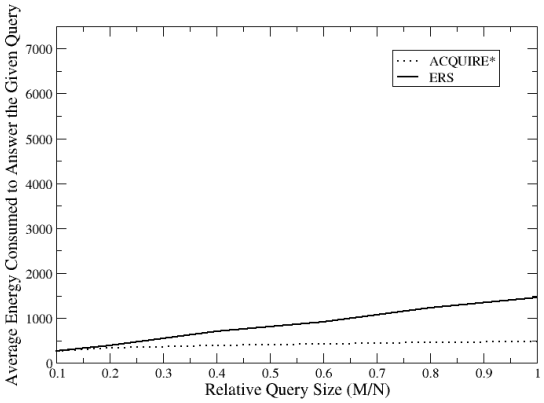
3.5 Validation of the Analytical Models

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

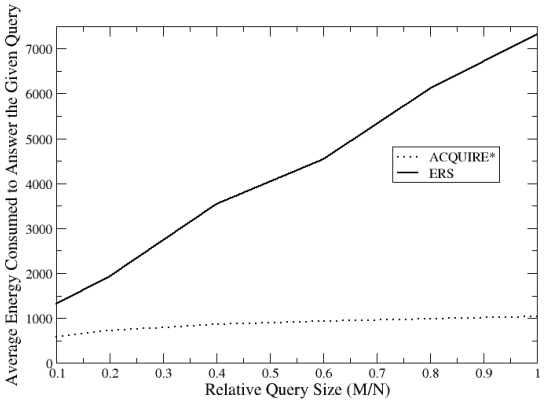
1. Validate the effect of c on ACQUIRE.



(a) $c = 0.05$



(b) $c = 0.2$



(c) $c = 1$

Figure 3.9: Comparison of $ACQUIRE^*$, and ERS ($N = 100$) with respect to $\frac{M}{N}$ for different amortization factors.

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

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 like overlap in the query trajectory, and boundary effects that are not modeled in our analysis.

3.5.1 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. These loops lead to a $45 - 50\%$ degradation in the performance[10] as we will show in section 3.5.3. For ERS and FBQ, there is no trajectory as the query is never forwarded to other nodes.

3.5.2 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. 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.

[10]The analysis of ACQUIRE with d=0 case assumed that there is no looping. The significant degradation observed in simulations suggests that the use of straightening algorithms such as those described in [4] or geographically directed trajectories such as routing on curves [36] is necessary when ACQUIRE is used with d=0 in practice.

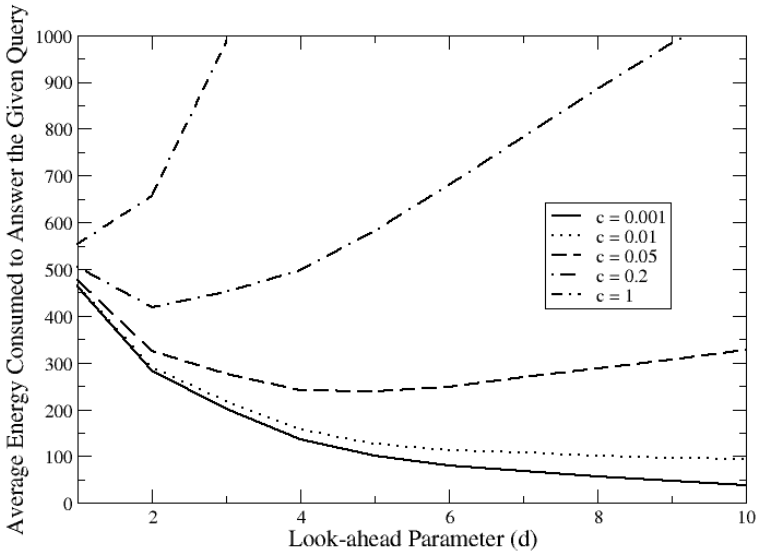


Figure 3.10: Effect of c on d^* by simulations ($N = 100$, $M = 20$). Compare with theoretical curves in figure 3.5

As figure 3.10 shows, with increasing c, the optimal look-ahead d^* decreases. This concurs with our analysis in section 3.2.6. 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 3.5. 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 3.5.1, ACQUIRE with $d = 0$ has loops in its trajectory, which degrades its performance by around $45 - 50\%$.

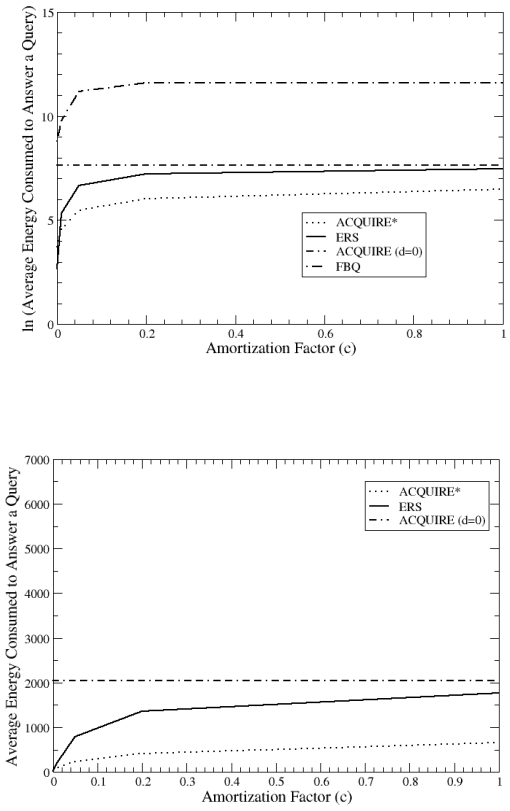


Figure 3.11: Comparison of $ACQUIRE^*$, ERS, ACQUIRE with $d = 0$ and FBQ by simulations ($N = 100$, $M = 20$) with energy costs on a log scale (left) and linear scale (right). Compare with theoretical curves in figure 3.7.

3.5.3 Comparison of ACQUIRE, ERS and FBQ

3.5.3.1 Effect of c

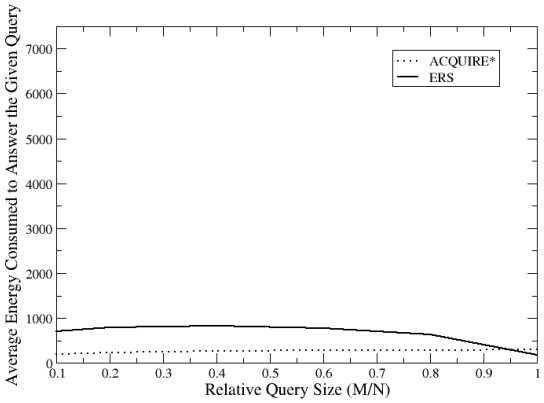
For both ERS and FBQ, we use the same simulation setup as ACQUIRE. For both these mechanisms, we simulate 100 different runs. Each run consists of 1000 queries each containing 20 sub-queries. In each run, we use the query-files, grid-files and same values of c described in section 3.5.2.

As figure 3.11 shows, FBQ has the worst energy consumption in comparison with the other approaches. This concurs with the analytical curves in figure 3.7. 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 discrepancy is because of the loops in the query trajectory in the case of random walk as was mentioned in section 3.5.1. However, the energy savings of $ACQUIRE^*$ over ERS seem to be similar to that shown by the analytical curves in figure 3.7.

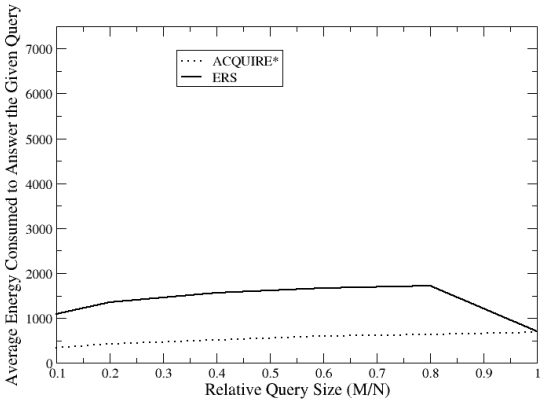
3.5.3.2 Effect of $\frac{M}{N}$

We use a similar set of simulation runs as mentioned in section 3.5.3.1 for values of c and $\frac{M}{N}$ as mentioned in section 3.4.2

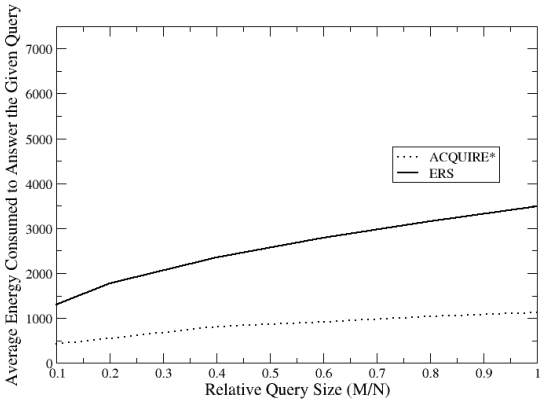
As figure 3.12 shows, for all values of c considered in this study, the average energy consumption of $ACQUIRE^*$ increases with increasing $\frac{M}{N}$. This behavior is also seen for ERS when $c = 1$. These observations are consistent with the analytical behavior in section 3.4.2. However, for $c = 0.05$ and $c = 0.2$, ERS seems to have a lower energy consumption at $\frac{M}{N} = 1$. This has to do with the way, we modeled ERS (and FBQ) in section 3.3.1 (and 3.3.2). In these mechanisms, we model the update frequency as $\frac{1}{c}$. Moreover these mechanisms are modeled to cache only those variables which are part of the query to give a better energy performance. In such situations, an update might occur even if a single variable of the complete query cannot be resolved. This inflates



(a) $c = 0.05$



(b) $c = 0.2$



(c) $c = 1$

Figure 3.12: 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 3.9

the energy consumption in cases where $M < N$. However, when $M = N$, all variables can be cached at the center. Moreover, in this case, there is only 1 possible query of $M = N$ variables. So, for values of $c < 1$, subsequent queries can be answered from the cache. If $c = 1$, an update will be done for every query and hence in this case, the energy consumption increases with increasing $\frac{M}{N}$. In the case of ACQUIRE, we cache all the variables within d hops at the cost of increased energy consumption. Moreover, since ACQUIRE has a Local Update phase and a Forward phase, we can exactly control the update frequency. If a *active* node cannot resolve a query completely, it will not seek an update, but will forward the query to another node.

From simulations, at $\frac{M}{N} = 1$, $c = 0.05$, ERS gives 50% energy savings over $ACQUIRE^*$. 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 $\frac{M}{N} \neq 1$) 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 this energy savings come at a cost? We attempt to answer this question in the next section by modeling the average response delay incurred by these mechanisms.

3.6 Latency Analysis

In this section, we attempt to analytically compare the average latency in answering a query (i.e. the response latency) 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.

3.6.1 ACQUIRE

In this section, we analyze the delay incurred by $ACQUIRE^*$ in answering a query. Our metric for delay is the number of sequential hop by hop transmissions.

$ACQUIRE^*$ takes S_M steps to answer a query of size M, where S_M is given by equation 3.6 in section 3.2.2. 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 (modeled 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:

$$\begin{aligned} T_{avg} &= c\{S_M(2d^*) + 2S_M(d^*)\} + (1-c)\{2S_M(d^*)\} \\ &= 2S_M d^*(c+1) \\ T_{avg} &= 2d^*\{\frac{N(ln(M)+\gamma)}{f(d^*)}\}\{c+1\} \end{aligned} \tag{3.33}$$

For the random walk,

$$\begin{aligned} T_{avg} &= E_{avg} \\ &= 2N(ln(M)+\gamma) \end{aligned} \tag{3.34}$$

3.6.2 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. 3.25. The delay 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 delay of 0). Thus, the average latency in ERS is given as follows:

$$\begin{aligned} T_{avg} &= c\{\sum_{i=1}^{t_{min}} 2i\} + (1-c)\{0\} \\ &= c(t_{min})(t_{min}+1) \end{aligned} \tag{3.35}$$

3.6.3 FBQ

Similar to the delay 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 delay 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 delay of 0). Thus, the average latency incurred by FBQ is

$$T_{avg} = 2ct_{min} \tag{3.36}$$

Figure 3.13 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 delay for both FBQ and ERS increases linearly with c as is evident from Eqn. 3.36 and Eqn. 3.35 respectively. Interestingly, the delay in $ACQUIRE^*$ seems to have a piece-wise linear behavior with respect to c. This is apparent in figure 3.13, where the x-axis is plotted on a log-scale. This is because the changing c alters d^* which in turn alters $\frac{d^*}{f(d^*)}$ i.e. the slope of the line T_{avg} as shown by Eqn. 3.33. At the point where the delay is constant w.r.t.c, $ACQUIRE^*$ resembles a random walk ($d = 0$). The difference in the delay is significant (around 500 transmissions), when d^* for ACQUIRE goes from 1 to 0.

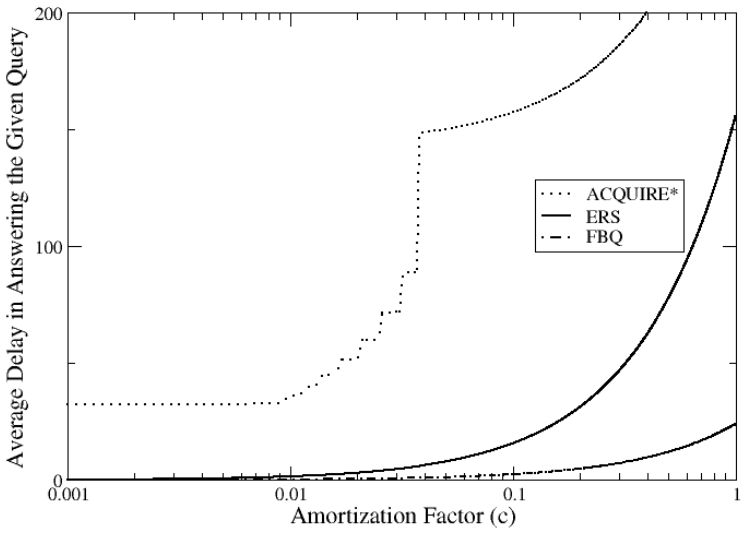


Figure 3.13: Comparison of the delay incurred by $ACQUIRE^*$, ERS and FBQ. Here, $N = 100$, $M = 20$, $X = 10^4$, $t_{min} = 13$. The x-axis is plotted on log-scale.

3.7 Storage Requirements

Our analysis in this chapter assumes that all the mechanisms 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 $f(d)$ 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.

3.8 Discussion

Partly for ease of analysis, we have described and modeled a very basic version of the ACQUIRE mechanism in this chapter. 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. 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 modeling 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 delay 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 study. Our assumption has been that each sub-query is independent. This would be another direction for future work.

3.9 Summary

In this chapter, we presented 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: one-shot 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 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 can be higher particularly when $NlnM$ is high and when c 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 present ACQUIRE as a highly scalable technique, energy-efficient at solving 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.

Chapter 4

En Masse Data Gathering

In many sensor network applications involving environmental monitoring in remote locations, planetary exploration and military surveillance, it is neither necessary nor even possible for a user to obtain data from the network continuously, in real time. In such applications, the information from the entire network can be extracted *en masse* after a prolonged period of sensing and local storage. This process of extraction can be done by a stationary sink or a mobile sink (robot). However, since communication is often the most expensive operation for a sensor node, the limited batteries may make it impossible to collect all the data stored in the network. We examine the problem of maximizing the data extracted (by a stationary sink) from such an energy-limited sensor network consisting of heterogeneous nodes. The maximum data extraction problem is an analog of the maximum lifetime problem of interest in continuous data-gathering sensor networks that has been studied previously [8] [28][1][50]. However, this problem introduces an additional element of "data-awareness" that must be considered in addition to "energy-awareness".

We first show how the maximum data extraction problem can be formulated as a linear program. We then adapt and extend techniques for multi-commodity flow problems first developed by Garg and Konemann [19] to develop an iterative algorithm for our problem with a provable $(1 + \omega)$ approximation. This algorithm suggests a new link metric (involving the remaining energies of both sender and receiver nodes, the distance between them, and the data level at the sender) that we then used to develop a fast, practically implementable, distributed heuristic that we refer to E-MAX. The heuristic employs a selfish strategy in that each sensor gives priority to transmitting its data before relaying that of other nodes. Our simulations show that this sophisticated heuristic (obtained from the analysis) offers near-optimal performance under all conditions, and significantly better compared to other energy-aware solutions (mainly developed by intuition) particularly when nodes are heterogeneous in their energy and data availability.

The rest of the chapter is organized as follows: we define the maximum data extraction problem and present the LP formulation and its dual in section 4.1. An interpretation of the LP dual suggests the $1 + \omega$ iterative approximation algorithm that we present and analyze in section 4.2. We discuss the implementation of this algorithm in section 4.3. We present fast implementable heuristics in section 4.4. Simulation results comparing these implementations are presented in section 4.5. Section 4.6 summarizes the chapter.

4.1 Problem Definition

Consider a scenario where several sensors that are deployed in a remote region have completed their sensing task and have some locally stored data. We are interested in collecting the maximum amount of data possible from all these sensors at a sink node T, given some remaining energy constraints in each of these sensors.

Figure 4.1 shows a sample scenario. Each node i is labeled with its (x,y) coordinates, its available data and remaining energy. The goal is to extract all this data to the sink node. The arrows and the indicated flows on each indicate the optimal solution for this particular example obtained by using the LP we describe in the next section.

We now present the formal model for the problem.

4.1.1 Model

Let N be the total number of sensors. Let T be the target (sink) to which the data is to be sent. Let D^j_{max} be the amount of data (bytes) collected by sensor j, where $D^j_{max} > 0$. e^j_{max} is the residual energy of sensor j. These sensors are arbitrarily deployed in a region. d_{ij} is the Euclidean distance between sensors i and j. A sensor i can communicate with any sensor j which is within the communication range R from it. Thus, this communicating range overlays a connectivity graph $G = (V, E)$, where $|V| = N$. An edge $(i, j) \in E$ iff $d_{ij} \leq R$.

The energy consumed in transmitting a unit byte from one sensor to the other depends on the distance between them.Tx_{ij} is the energy consumed in transmitting a single byte

from sensor i to j. Tx_{ij} is assumed to be proportional to the d_{ij}^2 i.e. $\text{Tx}_{ij} = \beta_t d_{ij}^2$, where $\beta_t > 0$. R_{ij} is the energy consumed at sensor j for receiving a single byte of data from sensor i. R_{ij} is assumed to be independent of the distance d_{ij}. Let $R_{ij} = \beta_r$, where $\beta_r > 0$.

For the ease of modeling, we add a fictitious source S, such that there exists an edge from S to every other node in V, except T. Also, add an edge from T to S. Let this new graph be G'. Thus, $G' = (V', E')$, where $V' = V \cup S$ and $E' = E \cup \{(S, i)\} \cup \{(T, S)\}$, where $i \in V$, $i \neq T$. As will be shown later in section 4.1.2, the location of the fictitious source S can be arbitrary.

The problem of collecting the maximum possible data from these energy-constrained sensor nodes can be formulated as a multi-commodity flow problem. As shown later in sections 4.1.2 and 4.1.3, the addition of S and its associated links simplifies the LP formulation for the multi-commodity flow problem and our analysis.

4.1.2 Linear Program (LP) Formulation

In this section, we formulate the maximum data collection problem as an LP. The constraints of the LP are

1. *Flow Conservation:* The amount of data transmitted by a node is equal to sum of the amount of data received by the node and the amount of data generated by the node itself.

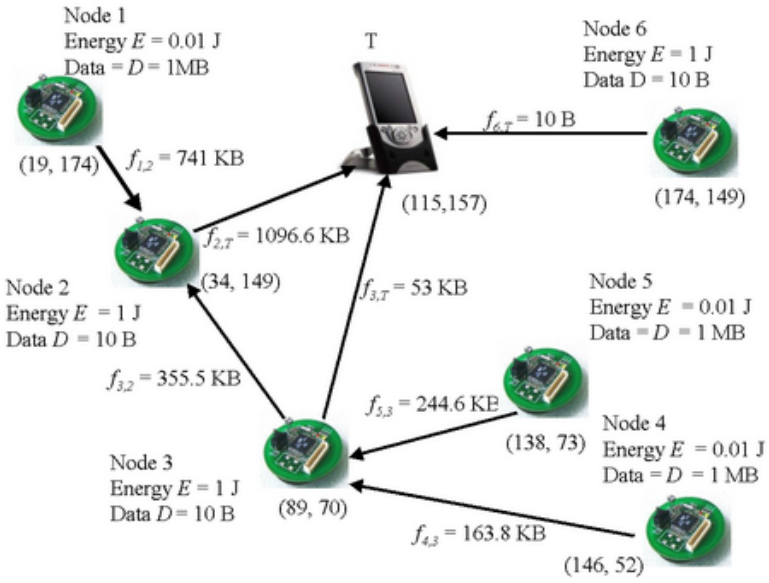


Figure 4.1: Illustration of a sample scenario with optimal solution to the maximum data extraction problem. Solution assumes $\beta_t = 32.4\mu J/byte/km^2$ and $\beta_r = 400nJ/byte$

2. *Energy Constraint:* The amount of data received and transmitted by a node is limited by the energy of the node. However, there are no energy constraints for the fictitious source S and the sink T.

Let $f_{i,j}$ be the amount of data transmitted from node i to node j. The LP can now be formulated as follows:

$$\begin{aligned} &\text{Maximize } f_{T,S} \text{ such that} \\ &\sum_{j=1}^{N} f_{j,i} - \sum_{j=1}^{N} f_{i,j} = 0 \\ i \neq S,T : \beta \sum_{(i,j)\in E} f_{i,j} d_{i,j}^2 + \sum_{(j,i)\in E} f_{j,i} &\leq e^i \\ i \neq S,T : f_{S,i} &\leq D_{max}^i \\ (i,j) \in E' : f_{i,j} &\geq 0 \end{aligned} \tag{4.1}$$

where

$$\beta = \frac{\beta_t}{\beta_r} \tag{4.2}$$

$$i \neq S,T : e_i = \frac{e_{max}^i}{\beta_r} \tag{4.3}$$

Note that we have normalized the energy in terms of receptions i.e. each reception costs a unit of energy, while each transmission from (i,j) costs βd_{ij}^2 units [1], where $\beta \geq 1$.

By adding the fictitious source S and its associated links, we have made the following transformations to the multi-commodity flow problem:

[1] In general, cost of transmitting a single byte over link (i,j) can be modeled as t_{ij}. This cost could account for metrics like link reliability, energy expenditure (as in this case), etc.

1. The amount of data transmitted by S to any node i is equal to the amount of data D^i_{max} generated at node i. The reception of this data from S does not incur any energy consumption. Hence, as we mentioned in section 4.1.1, the placement of S does not affect the solution.

2. Note that since the flow conservation is satisfied by both the fictitious source S and the target T, all the data received by the target will be transmitted to the source S, which in turn will be equal to the amount of data transmitted by the source S.

3. The problem now becomes one of maximizing the circulation of the commodities from S to T and back. The advantage of this will become apparent in section 4.1.3.

The above LP can be solved to compute the maximum amount of data that can be collected from the N sensors. It would also give the amount of data (flow) that should be sent from sensor i to sensor j.

However, in this study we are interested in proposing a constructive algorithm that maximizes the amount of data collected. For this purpose, we attempt to understand the structure of the primal LP solution by examining the dual LP in section 4.1.3.

4.1.3 Dual LP Formulation

The dual of the LP is as follows:

$$\begin{aligned}
\text{Minimize} \sum_{i \neq S,T} b^i e^i + c^i D^i_{max} & \\
i \neq S,T,(i,j) \in E : a^j - a^i + \beta b^i d^2_{ij} + b^j &\geq 0 \\
a^S - a^T &\geq 1 \\
i \neq T : a^i - a^S + c^i &\geq 0 \\
i \neq S,T : b^i &\geq 0 \\
i \neq S,T : c^i &\geq 0
\end{aligned} \tag{4.4}$$

Let a, b and c be vectors such that their i'th element is denoted by a^i, b^i and c^i respectively. Let

$$A(b,c) = \sum_{i \neq S,T} b^i e^i + c^i D^i_{max} \tag{4.5}$$

i.e. $A(b,c)$ is the objective function of the dual LP. The above dual has the following interesting interpretation:

Let $l(b,c)$ be a length metric and let $l_{i,j}(b,c)$ be the length of edge (i,j) in this metric. Then, if

$$l_{i,j}(b,c) \;=\; \begin{cases} \beta b^i d_{ij}^2 + b^j & \text{if} \;\; i \neq S \\ c^i & \text{if} \;\; i = S, j \neq T \end{cases} \tag{4.6}$$

the dual LP can be re-written as follows:

$$\begin{aligned} &\text{Minimize} \quad A(b,c) \quad \text{such that} \\ i \neq T, S : l_{i,j}(b,c) \;&\geq\; a^i - a^j \\ a^S - a^T \;&\geq\; 1 \\ l_{S,i}(b,c) \;&\geq\; a^S - a^i \end{aligned} \tag{4.7}$$

Consider an arbitrary S-T path P. Let P be $S, i_1, i_2, ...i_k, T$. Now, the length of path P can be written as follows:

$$\begin{aligned} l(P) \;&=\; l_{S,i_i}(b,c) + l_{i_k,T}(b,c) + \sum_{z=1}^{k-1} l_{i_z,i_{z+1}}(b,c) \\ &\geq\; (a_S - a_{i_1}) + (a_{i_k} - a_T) + a_{i_1} - a_{i_k} \;\; (\text{from Eqn.}4.7) \\ &\geq\; a_S - a_T \\ &\geq\; 1 \end{aligned} \tag{4.8}$$

Thus, the length of any arbitrary S-T path in the $l(b,c)$ metric is greater than or equal to 1, which implies that the length of the shortest path in the $l(b,c)$ metric should also be greater than equal to 1. Thus, by transforming the primal LP in section 4.1.2 into a circulation problem, we get a very simple value i.e. 1 for the length of the shortest path in the $l(b,c)$ metric.

Let $P(b,c)$ be the shortest path in the metric $l(b,c)$ and $\alpha(b,c)$ be the length of this shortest path. Thus, the objective of the dual LP is to minimize $A(b,c)$ such that $\alpha(b,c) \geq 1$. This is equivalent to minimizing $\frac{A(b,c)}{\alpha(b,c)}$.

Let

$$\zeta \quad = \quad \text{Min}\{\frac{A(b,c)}{\alpha(b,c)}\} \tag{4.9}$$

The above interpretation of the dual LP leads to a simple algorithm which approximates the optimal value of the primal. We specify the algorithm and analyze it in section 4.2.

4.2 Approximation Algorithm

We propose an iterative algorithm which is a modification of the Garg-Konemann algorithm [19]. Before proposing the algorithm, we introduce the following notations:

- b_j^i be the value of b^i in the j'th iteration.Initially, $b_0 = c_0 = \delta$ i.e. $\forall i : i \in V' : b_0^i = c_0^i = \delta$. The choice of δ is discussed in section 4.2.2.

- f_j be the total T-S (or S-T) flow till the j'th iteration. Initially, this flow is zero i.e. $f_0 = 0$.

- $A(j) = A(b_j, c_j)$ be the value of the dual objective function after iteration j.

- $P_j = S, i_1, i_2, ...i_k, T$ be the shortest S-T path in the $l(b_j, c_j)$ metric (at iteration j)[2]. On P_j, let $i_0 = S$ and $i_{k+1} = T$.

- $\alpha(j) = \alpha(b_j, c_j)$ be the length of the shortest S-T path after iteration j.

Moreover, $\forall i \in V'$, node i has a capacity associated with it. The capacity of a node depends on where it appears in an S-T path. Hence, we define the capacity of a node with respect to some path $P = S, i_1, i_2, ...i_k, T$ in which it appears. The capacity $\kappa(i)$ of each node in this path is modeled as follows:

$$\begin{aligned} \kappa(S) &= D_{max}^{i_1} \\ \kappa(i_1) &= \frac{e_{max}^{i_1}}{\beta d_{i_1,i_2}^2} \\ \kappa(i_z) &= \{\frac{e_{max}^{i_z}}{1 + \beta d_{i_z,i_{z+1}}^2}\} \text{ for} 1 < z \leq k \end{aligned} \tag{4.10}$$

The algorithm is as follows:

1. j = 0

[2]The number of nodes along the shortest S-T path changes from iteration to iteration i.e. it should be k_j. However for notational convenience, we drop the subscript j.

2. while($\alpha(j) < 1$) do

   (a) Select the shortest S-T path P_j.

   (b) Route c units of data along this path, where c is given as follows:

$$c = \text{Min}_{0 \leq z \leq k}\{\kappa(i_z)\}$$

   i.e. "saturate" the shortest path.

   (c) Update the vectors b and c as follows:

For $1 < z \leq k$,

$$b_j^{i_z} = b_{j-1}^{i_z}\{1 + \frac{\epsilon c \beta d_{i_z,i_{z+1}}^2}{e^{i_z}} + \frac{\epsilon c}{e^{i_z}}\}$$

For $z = 1$,

$$c_j^{i_1} = c_{j-1}^{i_1}\{1 + \frac{\epsilon c}{D_{max}^{i_1}}\}$$
$$b_j^{i_1} = b_{j-1}^{i_1}\{1 + \frac{\epsilon c \beta d_{i_1,i_2}^2}{e^{i_1}}\}$$

   (d) $f_j = f_{j-1} + c$

   (e) $j = j + 1$

On termination, f_j gives the value of the total data collected. While choosing the shortest S-T path, the algorithm accounts for the battery utilization (fraction of the battery power utilized) as well as the data utilization (fraction of the data sent) at each sensor. Thus, the

algorithm, as mentioned earlier in the introduction to this chapter, is both "energy-aware" and "data-aware". We analyze the approximation ratio of the algorithm in section 4.2.1.

4.2.1 Analysis

Initially, the value of the objective function of the dual LP is given as

$$\begin{aligned} A(0) &= \sum_{i \neq S,T} b_0^i e^i + c_0^i D_{max}^i \\ &= \sum_{i \neq S,T} \delta e^i + \delta D_{max}^i \end{aligned} \tag{4.11}$$

On a subsequent iteration j, the objective function is given as follows:

$$\begin{aligned} A(j) &= \sum_{i \neq S,T} b_j^i e^i + c_j^i D_{max}^i \\ &= A(j-1) + \epsilon c \{ c_{j-1}^{i_1} + b_{j-1}^{i_1} \beta d_{i_1,i_2}^2 + \sum_{(i_k,i_{k+1}) \in P_{j-1}}^{k \neq 1} b_{j-1}^i (1 + \beta d_{i_k,i_{k+1}}^2) \} \\ A(j) &= A(j-1) + \epsilon (f_j - f_{j-1}) \alpha(j-1) \end{aligned} \tag{4.12}$$

Solving the above recurrence, we get

$$A(j) = A(0) + \epsilon \sum_{l=1}^{j} (f_l - f_{l-1}) \alpha(l-1) \tag{4.13}$$

If each element of b_j and c_j is decreased by $b_0 = c_0 = \delta$, then the objective function of the dual LP is given as follows:

$$\begin{aligned} A(b_j - \delta, c_j - \delta) &= \sum_{i \neq S,T} (b_j^i - \delta)e^i + (c_j^i - \delta)D_{max}^i \\ &= \sum_{i \neq S,T} b_j^i e^i + c_j^i D_{max}^i - \sum_{i \neq S,T} \delta e^i + \delta D_{max}^i \\ A(b_j - \delta, c_j - \delta) &= A(j) - A(0) \end{aligned} \tag{4.14}$$

Let the shortest path P_j of length α_j be $S, i_1, i_2, ...i_k, T$. i.e. $\alpha_j = l_{P_j}(b_j, c_j)$. Now, the length of P_j in the metric $l(b_j - \delta, c_j - \delta)$ is given as follows:

$$\begin{aligned} l_{P_j}(b_j - \delta, c_j - \delta) &= l_{S,i_1}(b_j - \delta, c_j - \delta) + l_{i_1,i_2}(b_j - \delta, c_j - \delta) + \\ &\quad \sum_{(i_k,i_{k+1}) \in P_j}^{k \neq 1} l_{i_k,i_{k+1}}(b_j - \delta, c_j - \delta) \\ &= (c_j^{i_1} - \delta) + (b_j^{i_1} - \delta)\beta d_{i_1,i_2}^2 + \sum_{(i_k,i_{k+1}) \in P_j}^{k \neq 1} (b_j^{i_k} - \delta)(\beta d_{i_k,i_{k+1}}^2 + 1) \\ &= \alpha(j) - \delta\{1 + \beta d_{i_1,i_2}^2 + \sum_{(i_k,i_{k+1}) \in P_j}^{k \neq 1} (\beta d_{i_k,i_{k+1}}^2 + 1)\} \end{aligned}$$

Thus,

$$\alpha(b_j - \delta, c_j - \delta) \geq \alpha(j) - \delta L \tag{4.15}$$

where L is the largest value of

$$1 + \beta d^2_{i_1,i_2} + \sum_{(i_k,i_{k+1})\in P_j}^{k\neq 1} (\beta d^2_{i_k,i_{k+1}} + 1)$$

Now, from Eqn. 4.9 in section 4.1.3

$$\begin{aligned} \zeta &\leq \frac{A(b_j - \delta, c_j - \delta)}{\alpha(b_j - \delta, c_j - \delta)} \\ &\leq \frac{A(j) - A(0)}{\alpha(j) - \delta L} \\ \alpha(j) &\leq \delta L + \frac{\epsilon}{\zeta} \sum_{i=1}^{j} (f_i - f_{i-1})\alpha(i-1) \end{aligned} \tag{4.16}$$

In order to get an upper bound on $\alpha(j)$, we let all the $\alpha(i)$'s, where $1 \leq i \leq j-1$ to be as large as possible i.e. $\forall i : 1 \leq i \leq j-1$,

$$\alpha(i) = \delta L + \frac{\epsilon}{\zeta} \sum_{l=1}^{i} (f_l - f_{l-1})\alpha(l-1) \tag{4.17}$$

Thus,

$$\begin{aligned} \alpha(j) &\leq \alpha(j-1) + \frac{\epsilon}{\zeta}(f_j - f_{j-1})\alpha(j-1) \\ &\leq \alpha(j-1)(1 + \frac{\epsilon}{\zeta}(f_j - f_{j-1})) \\ &\leq \alpha(j-1)e^{\frac{\epsilon}{\zeta}(f_j - f_{j-1})} \\ \alpha(j) &\leq \alpha(0)e^{\frac{\epsilon}{\zeta} f_j} \\ \alpha(j) &\leq \delta L e^{\frac{\epsilon}{\zeta} f_j} \end{aligned} \tag{4.18}$$

Hence at iteration t, when the shortest S-T path has length greater than or equal to 1, the following inequality holds

$$\begin{aligned} 1 \leq \alpha(t) &\leq \delta L e^{\frac{\epsilon}{\zeta} f_t} \\ \frac{\zeta}{f_t} &\leq \frac{\epsilon}{ln(\frac{1}{\delta L})} \end{aligned} \tag{4.19}$$

Thus, despite the modifications to the Garg-Konemann algorithm to adapt it to our problem, we see that the bound obtained in the above inequality is exactly the same as that obtained in [19].

Although the algorithm chooses the shortest path based on data and energy utilization of nodes along the path, neither the remaining energy nor the remaining data at each sensor is actively monitored. Hence it would be interesting to see if the flow f_t is feasible. We analyze the feasibility of the flow in section 4.2.2.

4.2.2 Scaling

Whenever, b^i of a node i has increased, the node has been on a path whose length is strictly less than 1. If this had not been the case, the shortest S-T path had a length greater than 1 and the algorithm should have terminated. Moreover, the b^i is increased by a factor of at most $(1+\epsilon)$. Thus $b_t^i < (1+\epsilon)$. Now, if b^i is increased x times during the execution of the algorithm, in the worst case all these x increase operations completely utilize the energy e^i of node i. Thus, in the worst case, $b_t^i = \delta(1+\epsilon)^x < 1+\epsilon$, which implies that $x < log_{1+\epsilon}(\frac{1+\epsilon}{\delta})$. The factor by which the node i is over-utilized is i is

x. Thus by scaling f_t down by a factor of $log_{1+\epsilon}(\frac{1+\epsilon}{\delta})$, we can ensure that the energy constraint is not violated.

If b^i is increased x times, c^i is also increased at most x times. Thus, it can be seen that even the data constraint is violated by at most a factor of $log_{1+\epsilon}(\frac{1+\epsilon}{\delta})$. Thus, a single scaling operation ensures that both the energy and the data constraints are satisfied, giving a feasible flow of value $\frac{f_t}{log_{1+\epsilon}(\frac{1+\epsilon}{\delta})}$. Similar to [19], the ratio of the values of the dual to the primal solutions is

$$\gamma = \frac{\zeta}{f_t} log_{1+\epsilon} \frac{1+\epsilon}{\delta}$$

As shown in [19], if

$$\delta = (1+\epsilon)((1+\epsilon)L)^{-\frac{1}{\epsilon}} \tag{4.20}$$

then

$$\gamma \leq (1-\epsilon)^{-2} \tag{4.21}$$

Thus to get within a factor of $(1+\omega)$ of the optimal, ϵ is given as follows:

$$\epsilon \leq \sqrt{\frac{1}{1+\omega}} \tag{4.22}$$

Thus, the algorithm mentioned in section 4.2 can solve the multi-commodity flow problem and produce results arbitrarily close to the optimum.

Number of Iterations: From the analysis in this section, we observe that the factor by which a node i's energy is over-utilized is at most $log_{1+\epsilon}\frac{1+\epsilon}{\delta}$. Thus, the number of iterations in which i is the bottleneck on the shortest path is at most $log_{1+\epsilon}\frac{1+\epsilon}{\delta}$. There are a total of N nodes, and thus, the total number of iterations is $O(Nlog_{1+\epsilon}\frac{1+\epsilon}{\delta})$.

In the next section, we discuss a few issues which might be important for the implementation of the approximation algorithm in a sensor network.

4.3 Implementation of Approximation Algorithm

We now describe an iterative implementation of the algorithm mentioned in section 4.2. We refer to this implementation as A-MAX. In this implementation, the edge length metric is the one used by the approximation algorithm. The energy of each node i is initialized to e^i, while the data of each node is initialized to D^i_{max}. b^i_0 and c^i_0 are initialized to δ.

In each iteration the sink chooses a node with the shortest path to it as a *candidate* nodes. A sensor node is called *active* if it has been chosen by the sink as a *candidate* node at any iteration. Sensor nodes that are 1 hop away from the *active* nodes are called *threshold* nodes. Initially only the sink is *active*, while all the nodes within a distance R from the sink are *threshold* nodes. Each iteration k consists of the following steps:

1. The sink sends a message containing the iteration number k to its neighbors. It also advertises its shortest path to the sink having length 0.

2. Each neighbor i initializes its shortest path to the sink i.e. p_k^i to $\beta b_k^i d_{i,sink}^2$.

3. The *active* and the *threshold* nodes execute a distance vector algorithm using $\beta b_k^i d_{ij}^2 + b_k^j$ as the length of edge (i, j). Updates from downstream nodes are ignored to avoid loops. After the algorithm terminates, each of these nodes has the length of the shortest path and the next hop to the sink.

4. Each node i that is either a *active* or a *threshold* node sends a response towards the sink. This response contains the value of $p_k^i + c_k^i$. The response is also used to find the capacity of the path along which it is forwarded. The capacity is found as per Eqn. 4.10 in section 4.2. The forwarding process sets up a reverse path state in all the nodes along the path.

5. The sink selects the node z with the minimum p_i^k as a *candidate* node. If p_z^k is greater than or equal to 1, the sink sends a message to all active nodes to scale down their flows by the scaling factor mentioned in section 4.2.2 and then send their data towards the sink. In this case the algorithm terminates.

6. If p_z^k is less than 1, the sink sends a message addressed to the node z containing c i.e. the minimum capacity along the path. The message is forwarded based on the state created in an earlier step. As the message is being forwarded back along the reverse path, each node sets z to be downstream of itself. This is used to avoid

loops. For each node i_1 along the path, the value of $b_{k+1}^{i_1}$ is updated using the value of $b_k^{i_1}$ according to the algorithm.

7. The node z updates its routing table to augment the flow to its current next hop to the sink by c. It also updates c_{k+1}^z using c_k^z according to the algorithm.

8. Go to step 1.

Message Complexity: In each iteration, in the worst case all the N nodes will participate in the distance vector algorithm. Thus, from standard analysis of distributed Bellman Ford algorithm, the number of messages exchanged due to the distance vector algorithm in each iteration is $O(N|E|)$, where $|E|$ is the number of edges in the graph G [13]. For step 4, the response of each *active* or *threshold* node would be forwarded $O(N)$ times. Since, there are a total of N nodes, the total control overhead in step 4 is $O(N^2)$. Steps $5-6$ in the implementation require $O(E)$ messages in each iteration. Since $|E| = O(N^2)$, for each iteration, the total message complexity of A-MAX is $O(N^3)$.

As shown in section 4.2.2, the number of iterations is $O(N \log_{1+\epsilon} \frac{1+\epsilon}{\delta})$, and hence the total message complexity of A-MAX is $O(N^4 log_{1+\epsilon} \frac{1+\epsilon}{\delta})$.

4.3.1 Parameter Settings and Implementation Concerns

In A-MAX, each node i initializes $b_0^i = c_0^i = \delta$. As shown in Eqn. 4.21 in section 4.2.2, δ is dependent on ϵ and L. ϵ depends on the $(1+\omega)$ approximation needed as shown in Eqn. 4.22 in section 4.2.2. As shown in section 4.2.1, L is the largest value

of $\{1 + \beta d_{i_1,i_2}^2 + \sum_{(i_k,i_{k+1})\in P_j}^{k\neq 1}(\beta d_{i_k,i_{k+1}}^2 + 1)\}$. Thus, L depends on the topology of the network. However, it is possible to get a loose bound on L given as follows:

$$L \leq (N-1)\{\beta R^2 + 1\} \tag{4.23}$$

i.e. L is maximum when all the nodes are arranged in a linear fashion and just within range of each other.

In simulating this implementation however, we find that this loose bound on L results in extremely small values of δ (in fact so close to 0 that it has to be manually set to an arbitrary small positive value to make it work). As a result, A-MAX takes a large number of iterations to converge for small values of ω. Hence, as we shall see in the simulations section 4.5, A-MAX may not be suitable for practical implementations. This motivated us to look for the fast, practical heuristics described in the next section.

4.4 Fast Greedy Heuristics

4.4.1 Motivation

We can observe that the implementation of the approximation algorithm A-MAX consists of 2 phases. First, there is a *negotiation phase*, in which nodes do not actually transmit data but try to make decisions about how much data to send and how much data they will relay for other nodes. Since neither the remaining energy nor the remaining

data at a sensor is actively monitored at the negotiations phase, some nodes may be over-committed (i.e. may have negotiated to send or receive more data than allowed by their energy and data constraints). In the second, *data transmission phase*, each node scales down the data that it has to send or receive from each of its neighbors to accommodate the constraints and only then starts the data transmission towards the sink (T). This cumbersome two-phase process is one reason A-MAX is inefficient in implementations.

Another observation that can be made about the maximum data extraction problem is that the reception costs make it expensive for nodes to relay other nodes' data. As a result, one of the characteristics of the optimal solution is that each node should only commit to relay another nodes' data if it has energy remaining after transmitting all its own data - in other words, it behaves selfishly.

Motivated by these observations, we seek to develop efficient heuristics that avoid the 2-phase overshoot and scale down iterations of A-MAX and incorporate selfish, greedy behavior. It turns out that these heuristics are much faster in implementation and with the right metric can perform very well in practice.

4.4.2 Description of Heuristics

The key features of the heuristics we develop are:

1. Each sensor is greedy. i.e if a sensor has the shortest path to the sink, it accords priority to sending its data first. If after sending its data, the sensor exhausts its battery, it disconnects itself from the network.

2. There is no scaling operation.

3. Each sensor actively monitors its remaining battery power and remaining data.

We explore three variants of the greedy strategy. Each variant differs from the other in the link metric it chooses for distance vector routing.

Exponential Metric: This metric is based on the algorithm mentioned in section 4.2. At iteration k, if a sensor i receives or sends data, it updates its b^i as follows:

$$b_k^i = b_{k-1}^i\{e^{\epsilon\lambda_1^i}\} \tag{4.24}$$

where λ_1^i is the current battery utilization of sensor i i.e. the ratio of the battery power spent (till and including iteration k) to the total battery power. This update is approximately the same as that mentioned in section 4.2, when $\epsilon < 1$ (as $1 + x \approx e^x$, for small x). If the data sent is its own data, the sensor i also updates its c^i in the same manner:

$$c_k^i = c_{k-1}^i\{e^{\epsilon\lambda_2^i}\} \tag{4.25}$$

where λ_2^i is the current data utilization of sensor i i.e. the ratio of quantity of its data sent (till and including iteration k) to its total data quantity. However, the initial values of b^i's and c^i's are still to be chosen. Intuitively, if all sensors have the same value for the ratio of $\frac{D_{max}^i}{e^i}$, all these sensors can be treated the same. However, if a few sensors have a low value for this ratio, which implies that they can carry data from other sensors, it would

be advantageous to choose these sensors as the next hop towards the sink. Hence, we set the initial values of b^i's and c^i's are follows:

$$b_0^i = \frac{D_{max}^i}{e^i} \tag{4.26}$$

$$c_0^i = \frac{D_{max}^i}{e^i} \tag{4.27}$$

We will refer to the *Exponential Metric* based implementation as E-MAX. For this implementation we assume $\epsilon < 1$.

To illustrate the importance of this metric, we choose two different sets of variants of the distance vector implementation. The first set consists of variants that are neither energy-aware nor data-aware. These are

1. *DIST-MAX*: The length of an edge (i, j) (such that $d_{ij} \leq R$) is d_{ij}. This metric is constant across iterations. DIST-MAX finds the shortest path based on Euclidean distances.

2. *H-MAX*: The length of an edge (i, j) is 1 iff $d_{ij} \leq R$. This metric is also constant across iterations. H-MAX finds the shortest path based on hop count.

   As DIST-MAX and H-MAX do not use b^i 's, c^i 's or the data levels at the nodes, they are neither energy-aware nor data-aware. We also consider a set of energy-aware variants of the distance vector implementation proposed in [51]:

3. *LINEAR-COST*: The length of an edge (i, j) (such that $d_{ij} \leq R$) at iteration k is $b_k^i + b_k^j$ such that:

$$b_k^i = c\{\lambda_1^i\} \tag{4.28}$$

$$b_0^i = \frac{1}{e^i} \tag{4.29}$$

4. *QUAD-COST*: The length of an edge (i, j) (such that $d_{ij} \leq R$) at iteration k is $b_k^i + b_k^j$ such that:

$$b_k^i = c\{\lambda_1^i\}^2 \tag{4.30}$$

$$b_0^i = \frac{1}{e^i} \tag{4.31}$$

Both LINEAR-COST and QUAD-COST assign cost (b^i) to a node i that is a function (linear and quadratic respectively) of the current battery utilization λ_1^i of node i. In our study, we set $c = 0.5$. Thus, both LINEAR-COST and QUAD-COST find the shortest path in terms of an appropriate function (linear and quadratic respectively) of the battery utilization of the nodes.

5. *MIN-ENERGY*: The length of an edge (i, j) (such that $d_{ij} \leq R$) is $\beta d_{ij}^2 + 1$, which is the total energy spent in transmission and reception over link (i, j). MIN-ENERGY finds the shortest path in terms of energy expenditure.

It is interesting to note that the edge length metric $b^i \beta d_{ij}^2 + b^j$ used by E-MAX is a hybrid of the edge length metrics used by LINEAR-COST (or QUAD-COST) and MIN-ENERGY. However, E-MAX uses b^i as an exponential function of the battery utilization of i. E-MAX also introduces an additional data-awareness by using c^i (in the path length from node i) and initializing b^i 's depending on the data levels at the nodes.

Number of Iterations: In all the above greedy heuristics, at each iteration, a node i's energy or data is completely exhausted. Since, there are a total of N nodes, the number of iterations is $O(N)$.

4.4.3 Implementation of Heuristics

The implementation mentioned in section 4.3 can be easily extended to incorporate the greedy heuristics with the appropriate edge length metrics. However, there are some differences:

1. Each node i uses the appropriate initializations for b_0^i and c_0^i. In E-MAX, Eqn. 4.26 and Eqn. 4.27 are respectively used. In LINEAR-COST (or QUAD-COST), Eqn. 4.29 (or Eqn. 4.31) is used. At each iteration k, E-MAX updates the value of b_k^i and c_k^i as per Eqn. 4.24 and Eqn. 4.25 respectively, while LINEAR-COST (or QUAD-COST) updates b_k^i using Eqn. 4.28 (or Eqn. 4.30). H-MAX, DIST-MAX and MIN-ENERGY do not use the values of b^i 's and c^i 's.

2. Each node i actively monitors its remaining battery and remaining data levels. While calculating the capacity of the path, the remaining battery power and remaining data are used as opposed to e^i and D^i_{max} used in A-MAX. If the battery level of a sensor hits 0, it ceases to be a part of the network and hence does not execute the distance vector algorithm.

3. While sending a response to the sink, each *active* and *threshold* node sends its remaining data in addition to the length of its path to the sink. The sink chooses a *candidate* from those nodes that still have remaining data to send.

4. At each iteration, the *candidate* node immediately sends data to the sink.

5. There is no scaling operation involved. The algorithm terminates if the sink detects that all the nodes[3] in the network do not have data to send or if the sink is disconnected from the network due to its first hop neighbors dying out.

Message Complexity: For these greedy heuristics, the message complexity at each iteration is the same as that for A-MAX i.e. $O(N^3)$. Since, as shown in section 4.4.2, the number of iterations is $O(N)$, the total message complexity is $O(N^4)$.

In section 4.5, we describe our simulation scenarios used to compare the different approaches and the results.

[3]This might need the sink to have an approximate estimate on the number of nodes in the network.

4.5 Simulations and Results

We simulated A-MAX and the heuristics (E-MAX, DIST-MAX, H-MAX, LINEAR-COST, QUAD-COST and MIN-ENERGY) using a high level simulator. This simulator did not model MAC effects.

The simulation set up consists of 50 nodes randomly deployed in an area of 0.5 km x 0.5 km. 10 different random seeds are used in the simulations and the results are averaged across these runs.

Each node has a radio range of 0.2. We use the first order radio model used in [28]. In this model, a sensor consumes $\epsilon_{elec} = 400nJ/byte$ for running the transmitter or receiver circuitry and $\epsilon_{amp} = 800pJ/byte/m^2$ for running the amplifier circuitry. Thus $\beta_r = \epsilon_{elec}$. In order to send a single byte, the sensor has to run its transmitter and amplifier circuitry. Now, a sensor can receive data only from sensors within its range i.e. within a distance of R. Hence,

$$\beta_t \approx \epsilon_{elec} + \epsilon_{amp} R^2$$

Thus, using the definition of β from Eqn. 4.2 in section 4.1.2,

$$\beta \approx 1 + \frac{\epsilon_{amp}}{\epsilon_{elec}} R^2$$

Converting distances into km units, for $R = 0.2$ km, we get $\beta = 81$. We use this value of β in our simulations. We assume that the maximum energy is 1J as in [28]. As

we mentioned earlier in section 4.1.1, our model assumes that a reception of a single unit(byte) consumes one unit of energy, we normalize the maximum energy in terms of bytes that can be received. From the value of ϵ_{elec}, it can be shown that 1J of energy allows $1250K$ receptions. Thus, in our simulations we set the maximum energy to $1250K$. The maximum data is set to $1000K$ bytes. We also count the number of transmissions and receptions due to exchange of control messages in the various implementations (that are appropriate modifications of A-MAX). These control messages include the distance vector updates (step 3), messages forwarded during the response of an *threshold* node or *active* node to the sink (step 4) and the messages forwarded during the response of the sink to the candidate node (step 5 and 6). These control messages are assumed to be of 1 byte. This is a reasonable assumption as the control messages usually contain a single real valued quantity. Moreover, if i transmits a control message to j, we compute the energy spent in the control message as βd_{ij}^2 for node i's transmission and 1 for node j's reception. This helps us to quantify the overhead of A-MAX and the heuristics in terms of the percentage of energy at each node.

The amount of data collected in our problem scenario is mainly dependent of the energy of the first hop sensors and the amount of data they have. Thus, in our simulations we define a few *good* nodes. A node i is a *good* node if it has a very low value of $\frac{D_{max}^i}{e^i}$. These *good* nodes do not have any other special capabilities. For a *good* node i, $e^i = 1250000$ (1J of energy) and $D_{max}^i = 10$ (bytes) of For a node j that is not *good*, $e^i = 1250$ (0.01J of energy) and $D_{max}^i = 1000K = 1M$ (bytes). As we show in our

results, when all nodes have the similar values of energy and data, the three heuristic implementations perform very similar and give close to optimal performance. Varying the energy and data levels of the *good* nodes, their number and their placement helps us to outline scenarios where E-MAX significantly outperforms the other heuristics.

We were interested in studying the performance of A-MAX and the heuristics in terms of optimality of solution produced and the overhead. We compared the solutions produced by these implementations with that produced by solving the LP specified in section 4.1.2 using *lp_solve*. The objectives of our simulations were to study the following:

1. Effect of δ and ω on the performance of A-MAX.
2. Effect of *good* nodes on the performance of the heuristics.
3. Effect of density on the performance on the heuristics.

4.5.1 Effect of δ and ω on A-MAX

We discuss the scenario when there are 20 *good* nodes in the network and 4 of them are in range of the sink. However, the conclusions are applicable to most of the scenarios we examined.

For a given ω, ϵ can be easily determined using Eqn. 4.22 in section 4.2.2. We set ϵ to the maximum value permitted by Eqn. 4.22 for a given ω. In our case, if ϵ is very small, due to a very loose bound on L, $\delta \approx 0$. Hence, for the implementation to work correctly,

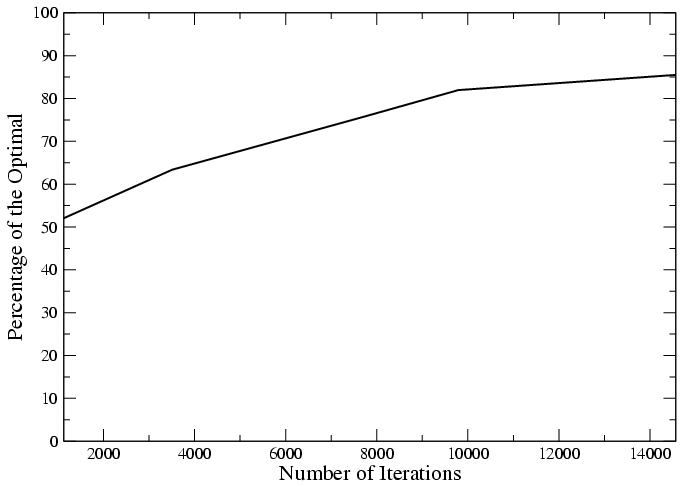


Figure 4.2: Effect of number of iterations on the solution produced by A-MAX due to varying δ. The area of deployment is 0.5 km x 0.5 km. $R = 0.20$ km. Number is nodes is 50. $\omega = 0.1$.

we set δ to small non-zero values. As mentioned in section 4.3.1, the value of δ affects the number of iterations of A-MAX and the optimality of the solution produced.

For the aforementioned scenario, we use the following values of δ: 0.0000001, 0.000001, 0.0001, 0.01 and 0.1. We set $\omega = 0.1$. We count the number of iterations taken by A-MAX across these values. We notice that as δ decreases, the number of iterations increases.

Figure 4.2 shows the effect of the number of iterations (by varying δ) on A-MAX. As the number of iterations increases (δ decreases), A-MAX produces solutions that are closer to the optimal.

We also varied ω by fixing $\delta = 0.0000001$. Figure 4.3 shows the effect of the number of iterations (by varying ω) on the optimality of the solution produced by A-MAX. We used the following values of ω: $0.01, 0.1, 1, 3, 5, 7$ and 10. As ω is increased, the number

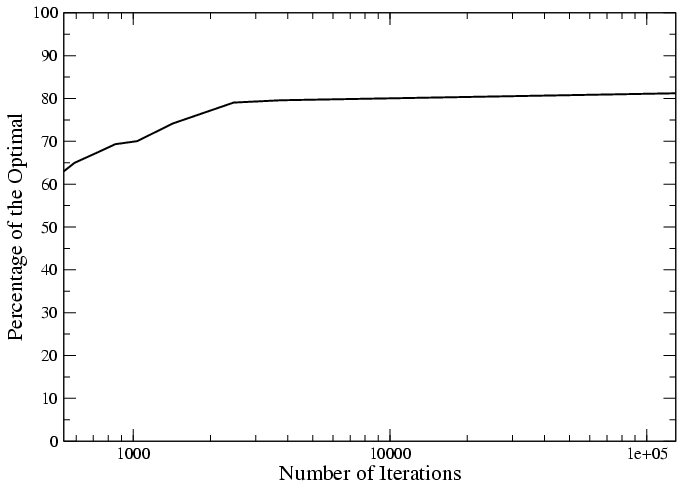


Figure 4.3: Effect of number of iterations on the solution produced by A-MAX due to varying ω. The area of deployment is 0.5 km x 0.5 km. $R = 0.20$ km. Number is nodes is 50. $\delta = 0.0000001$. The x-axis is plotted on a log-scale.

of iterations decrease. We also observe that at $\omega = 10$, the solution produced by A-MAX is around 65% of the optimal, while at $\omega = 7$, the solution is around 80% of the optimal.

Figures 4.2 and 4.3 demonstrate the trade-off between the number of iterations and the quality of solution obtained by A-MAX. In general, for solutions that are arbitrarily close to the optimum, a greater number of iterations are needed. This translates to a higher number of control messages and hence a greater overhead. In all these scenarios, the overhead of A-MAX measured from simulations precludes its implementation in an energy-constrained sensor network.

These curves justify the need for fast heuristics that we developed. Next, we evaluate the performance of the heuristics mentioned at the beginning of section 4.5.

4.5.2 Effect of *good* nodes on the heuristics

Our initial aim was to understand the effect of ϵ on E-MAX. In a sample scenario with 10 good nodes placed at random, we varied ϵ from 0.1 to 1 in steps of 0.2. Across all these scenarios, the solution produced by E-MAX remains unchanged. So, in the rest of our simulations with E-MAX, we arbitrarily set $\epsilon = 0.1$.

From our preliminary study, we observe that if all the nodes in the network have similar amount of energy and data, then all these greedy implementations perform similarly. However, the importance of the exponential metric becomes apparent as we vary the energy, the data levels, the number and the placement of *good* nodes in the network.

Initially, we choose these good nodes at random. Figure 4.4 shows the effect of the energy of the *good* nodes. In this scenario, we fix the number of good nodes to be 20. We vary their energies (and appropriate reception units) from 1 times (i.e. 0.01 J) to 100 times (i.e. 1 J) of the energy of the other nodes (i.e. 0.01 J). We observe that the energy of the *good* nodes has a significant impact on the relative performance of the heuristics. At lower values (around 0.01 J) of energy for *good* nodes, all the heuristics perform similarly and close to optimal (within 5%). The performance gap between E-MAX and the other heuristics increases (up to 250%) as the energy of the *good* nodes increases, all other factors remaining the same. DIST-MAX (and MIN-ENERGY) are seen to perform marginally better than H-MAX (and LINEAR-COST and QUAD-COST). It would be interesting to see if the significant difference between E-MAX and the other heuristics at high energy values of the *good* nodes is independent of other factors like the data levels

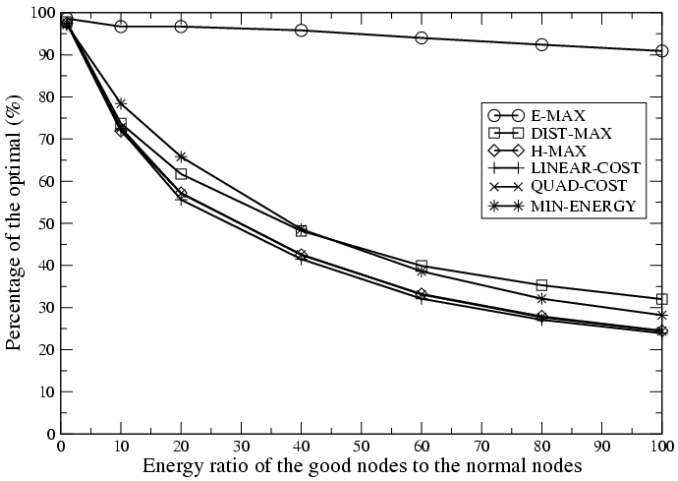


Figure 4.4: Effect of the energy of *good* nodes on the heuristics. The area of deployment if 0.5 km x 0.5 km. $R = 0.20$ km. Number is nodes is 50.

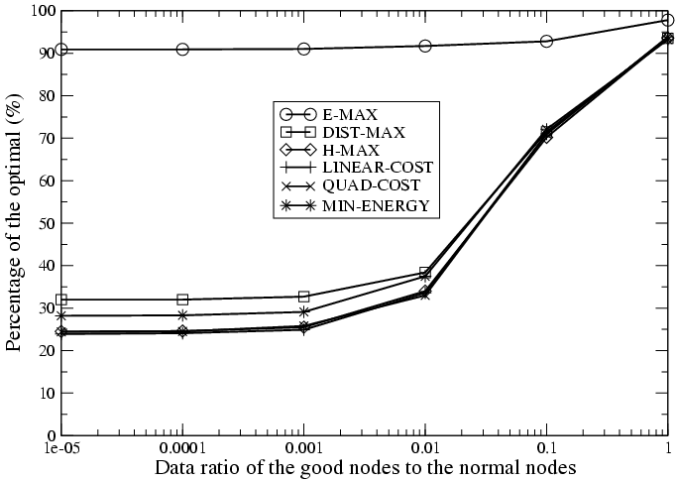


Figure 4.5: Effect of the data ratio of the *good* nodes on the heuristics. The area of deployment is 0.5 km x 0.5 km. $R = 0.20$ km. Number is nodes is 50. The x-axis is plotted on a log-scale.

of the *good* nodes, fraction of *good* nodes and their placement. Hence in the subsequent simulations, we set the energy of the *good* nodes at 1J.

The effect of the data ratio of the *good* nodes is shown in figure 4.5. In these scenarios, 20 *good* nodes are chosen at random as before. While the energy level of these *good*

nodes is fixed at 1 J, the data levels of are varied from 10^{-5} times (i.e. 10 bytes) to 1 time (i.e. 1 MB) of the data levels of the normal nodes. As observed earlier, all heuristics perform similarly and quite close to the optimal (within $5 - 10\%$) when all nodes have the same amount of data (or when the data ratio is 1). However, E-MAX outperforms the other heuristics significantly (up to 300 %) when the data ratio of the good nodes decreases.

Figure 4.6 shows the effect of increasing the ratio of the *good* nodes in the network. In these scenarios, E-MAX produces solutions that are within 10% of the optimal. It also gives up to 450% improvement (more data collected) over the other heuristics. This happens when the fraction of *good* nodes is 0.8. At the extremes of the x-axis, when almost all nodes are similar in their energy and data levels, all the heuristics perform equally good and give very close to optimal solutions. As the fraction of *good* nodes increases beyond 0.8, the amount of data to be collected is far lower than the amount of energy in the network. Hence all the schemes yield close to optimal solutions. On the other hand, when the fraction of *good* nodes is close to 0, the amount of data to be collected by far exceeds the energy in the network. In such cases, nodes that are closer to the sink monopolize the flow of data and hence all heuristics perform the same. From the topology, we were able to deduce that the increase in the number of *good* nodes leads to an increase in the number of *good* nodes close to the sink. While E-MAX is able to take advantage of these nodes, the other heuristics do not.

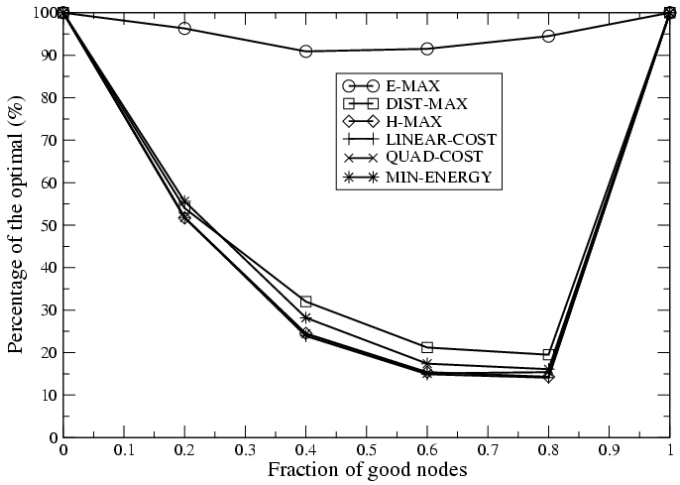


Figure 4.6: Effect of fraction of *good* nodes on the heuristics. The area of deployment is 0.5 km x 0.5 km. $R = 0.20$ km. Number is nodes is 50.

These observations suggest that not only the fraction but also the placement of these *good* nodes relative to the sink affects the relative performance of these heuristics.

To understand the effect of the placement of the *good* nodes relative to the sink, we fix the number of *good* nodes to 15 and increase the number of *good* nodes within the range of the sink (in our random scenarios, the minimum number of neighbors of the sink was 8). These *good* nodes are chosen at random. Intuitively, as the number of *good* nodes within the range of the sink increases, the rest of the heuristics should perform as good as E-MAX. But figure 4.7 shows otherwise. This illustrates the importance of the placement of the *good* nodes. If there is a connected "back-bone" of *good* nodes i.e. a series of *good* nodes in range of each other and in range of the sink, E-MAX is able to take advantage of them as opposed to the other heuristics. This "back-bone" helps in gathering more data. E-MAX is able to gather around 300% more data than the other heuristics (when the number of first hop *good* nodes is 3, 6 and 8). The metric

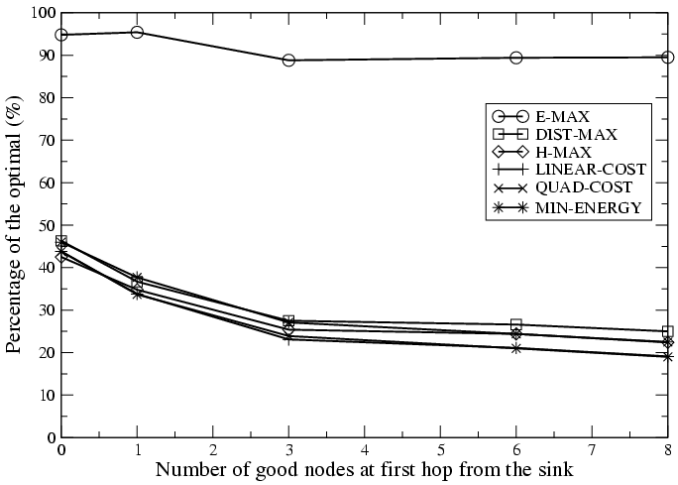


Figure 4.7: Effect of placement of *good* nodes on the heuristics. The area of deployment is 0.5 km x 0.5 km. $R = 0.20$ km. Number is nodes is 50.

used by E-MAX helps to route the data along such "back-bones". Even if there are only up to 8 *good* nodes at the first hop, the high density of the scenario considered leads to the existence of such "back-bones". It would be interesting to examine the relative performance of these heuristics in lower density deployment. Hence, we next vary the size of the deployment area and study the performance of the heuristics.

4.5.3 Effect of density on the heuristics

We randomly (using the same 10 random seeds as before) deploy 200 nodes using a communication range of 0.25 km in a bigger area of 2km x 2km.

The performance trends shown in Figure 4.8 are very similar to those in Figure 4.6. Again E-MAX gives solutions that are within 10% of the optimal solution. E-MAX can

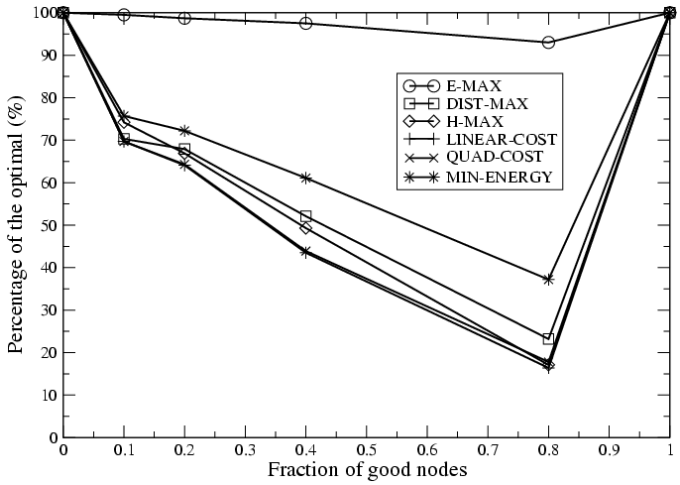


Figure 4.8: Effect of fraction of *good* nodes on the heuristics. The area of deployment is 2 km x 2 km. $R = 0.25$ km. Number is nodes is 200.

also give more than 200% improvement over the other heuristics. This occurs when the fraction of good nodes is 0.8.

Across all of the scenarios mentioned in sections 4.5.2 and 4.5.3, the heuristics took significantly lesser number of iterations as compared to A-MAX. For example, in the scenario used for the performance of A-MAX in section 4.5.1, E-MAX took only 36 iterations and gives a solution that is 99.9% of the optimum. Such significant reduction in iterations were observed across all scenarios used in our study. This translated to an overhead in the range of 5-10%. This overhead shows that these greedy heuristics are suitable for implementation in an energy-constrained sensor network.

When all nodes have similar data and energy levels, all these greedy implementations perform similarly. However, if there are nodes in the network with very high energy and low data, E-MAX (due to the metric based on the Garg-Konemann algorithm) significantly outperforms the other approaches. In some scenarios it results in collecting up to

450% more data than the other energy-aware routing approaches. In most of the scenarios we used, E-MAX gives flows that are within 10% of the optimum.

4.6 Summary

In remote monitoring sensor network applications where data does not need to be gathered continuously, the key problem of interest is that of extracting the maximum information from the local storage of network nodes, post-sensing. We have formulated this problem as a linear program in this chapter, and presented and analyzed a $1 + \omega$ iterative approximation algorithm for it. For ease of distributed implementation, we then developed a greedy heuristic that incorporates the link metric suggested by the approximation algorithm. Our simulation results enable us to conclude that this heuristic, the E-MAX algorithm, performs near-optimally (within 1 to 10% in most scenarios) and significantly better than other energy-aware routing approaches that are mainly developed through intuition.

While the problem formulation we presented does not explicitly incorporate fairness, it could be extended with some modifications to incorporate different priorities or even equal priorities on the data from sensor nodes by varying the data constraints on individual nodes. A thorough examination of fairness issues is one of our areas for future work. While we have assumed that a node transmits at the maximum rate, it would be interesting to study the tradeoff between the data collected and the delay in cases where a node can tradeoff the transmission rate for a lower energy expenditure.

Other future work could involve the incorporation of data aggregation. Existing results pertaining to maximum lifetime problems with aggregation, such as [28] and [1], suggest ways in which our work may be extended in this direction.

Chapter 5

Continuous Data Gathering

In several application scenarios, a central sink may be interested in obtaining a periodic update about the status of the physical quantities such as temperature, pressure, humidity, etc from the network. Consider the following queries:

1. "Report the maximum temperate in the network for the next 30 minutes at the rate of 1 reading every 30 seconds". In this case, each node just needs to send a single unit of data, which is the maximum temperature in its sub tree. We call such an aggregation as "perfect" aggregation.

2. "Report the temperature from all the nodes for the next 30 minutes at the rate of 1 reading every 30 seconds". In this case, each node needs to send the temperature of all the nodes in its sub tree and there is no aggregation.

The network should respond to these queries by streaming data to the sink at the required periodic rate. Hence such queries are called continuous queries. One mechanism to organize a continuous data gathering from a large multi-hop sensor network, is to construct

a tree rooted at the base-station/sink node. This can be done by a two phase process whereby the sink issues a flood, and nodes organize themselves by level (depending on how many hops they are from the sink) and select a neighboring node at the previous level to be their parent [27]. Zhou and Krishnamachari show that the parent selection criteria greatly impacts the nature of the final tree created [59]. While their study is primarily simulation based, in this chapter, we adopt a more formal approach to designing local parent selection criteria using techniques from mechanism design and game theory.

Informally, we investigate the following problem: *Given a global objective function, can suitable local utility functions be designed such that each sensor node while "selfishly" optimizing its own local utility function leads to optimizing the desired global objective*.

Specifically, we consider the global objective of constructing a tree that results in a fair utilization of energy resources in the network such that no sensor node suffers an early energy depletion.

We assume that the energy dissipation at a sensor node is proportional to the number of data units received and transmitted by it. For a data gathering tree used to execute a query of the first type, each sensor node (other than the root) transmits one unit of data. The number of data units received by a sensor node is proportional to the number of its children. For such a query, it is useful to construct a load balanced tree, where the load (number of children attached to a parent) is balanced at each level of the tree. Alternatively, consider a sensor node executing a query of the second type. The number

of units received and transmitted by it is proportional to the size of its subtree. In such a scenario, it is of interest to construct an energy balanced tree in the network that ensures that nodes with lower energy have smaller subtrees.

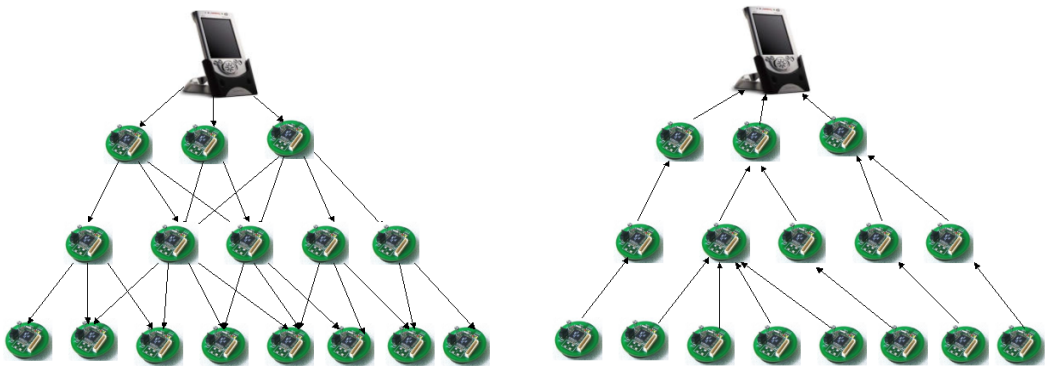
For the purpose of demonstrating our approach, we focus on the construction of continuous data gathering trees that are optimized for the global objectives described above: load balance and energy balance. For each of these objectives, we design a local utility function for each sensor node, such that the desired objective is attained while the sensor nodes selfishly optimize their local utility functions. We present an efficient, distributed heuristic *DistributedParentBid* for constructing a load balanced tree in the network. The performance of *DistributedParentBid* is evaluated by comparing against an optimal centralized algorithm over several scenarios involving random deployment of sensor nodes in the network. Our simulation results show that the heuristic produces the desired load balanced tree for around 90% of the simulation scenarios. We also show that an energy balanced tree can be built over the network, provided each sensor node routes its data along the shortest path (using an edge length metric that is weighed inversely to the current energy of the sensor node) to the sink. This can be achieved in a simple manner by using a distributed distance vector algorithm with the appropriate edge length metric. As mentioned in section 2.4 in chapter 2, quite a few studies have used the utility-based approach for designing routing algorithms for sensor networks. Byers and Nasser use this approach to study applications that assume "perfect" aggregation and do not require

the collection of data from all the sensor nodes at each round [6]. In this work, we investigate application scenarios that require data from all the sensors at each round of data gathering. Within this class of applications, we consider both "perfect" and no data aggregation. Kannan *et al.* assume a utility function for each sensor *apriori* and analyze the Nash Equilibrium resulting from the interaction of selfish sensors in the context of reliable data gathering [29]. Here, our aim is to design appropriate utility functions such that the resulting Nash Equilibrium coincides with the desired global objective.

The rest of the chapter is organized as follows: the formal description of the load balanced data gathering tree construction problem is given in section 5.1. We describe and analyze our decentralized algorithm for the construction of a load balanced tree in sections 5.1.1, 5.1.2 and 5.1.3. Section 5.2 describes our second case study, which is the construction of an energy balanced data gathering tree in the network. Finally, section 5.3 summarizes the chapter.

5.1 Load Balanced Data Gathering Tree Construction

Spanning trees have been widely used for gathering data from a large multi hop network to the base station/sink node. The following two phase process can be used for constructing a spanning tree in a sensor network as shown in figure 5.1. The first phase involves flooding of a message from the sink node to all the sensor nodes in the network. This is followed by the second phase, whereby the sensor nodes organize themselves into levels based on their distance to the sink node. The sensor nodes at lower levels correspond to



(a) Sink issues a flood, nodes organize into levels

(b) Each node selects a parent at the higher level

Figure 5.1: An illustration of load balancing. (b) depicts an arbitrary parent selection strategy. Notice that level 1 is load balanced, while level 2 is not, given the graph connectivity restrictions in (a)

those closer to the sink node. Each sensor node other than the sink node selects one node from the previous level (with which it can communicate) as its parent [59].

In this section we consider a data gathering application that assumes "perfect" aggregation. The leaf nodes of the tree transmit one unit of data to their parent. The parent nodes (except the sink node) receive all the data from their children, aggregate the gathered information, and transmit one unit of data to their respective parent. One unit of energy is dissipated at the leaf nodes, while the parent nodes dissipate energy proportional to the number of children. Our goal is to construct a load balanced spanning tree that ensures fair utilization of the energy resources at all the sensor nodes.

Due to "perfect" aggregation, a key feature of the balanced data gathering construction problem is that the decisions taken by sensor nodes at one level of the tree are independent of those taken at another level. Thus, at each level, ensuring that each sensor node selects its parent such that none of the parents are overloaded will lead to a global

tree in which none of the sensor nodes are overloaded. Hence, we are interested in constructing a load balanced tree in the network that has the following property: Given a set of candidate parent nodes, each sensor node must select as parent, the sensor node at the previous level with the smallest number of children. In other words, if a sensor node discovers a candidate parent node that has fewer children than the current parent node, the sensor node must select this node as its new parent. The load balancing problem is essentially finding an optimal semi-matching in a bipartite graph. Harvey *et al.* show that this problem can be reduced to a minimum cost maximum flow problem [21]. They present efficient centralized offline algorithms for finding the optimal semi-matching. They also show that the resulting optimal semi-matching minimizes all norms including the variance of the load (number of children) i.e. the $||L_2||$ and the maximum load ($||L_\infty||$). In this study, we present a distributed algorithm for load balancing. As discussed earlier, the choices made by a sensor node at one level in the tree do not affect the choices made by the sensor nodes at the other levels. Hence, in the remaining part of this section, we will focus on a single level of the hierarchical structure created by the initial flood i.e. a set of children nodes with their corresponding sets of possible parents.

Before formally describing the problem, we introduce some preliminary notation:

1. M is the set of all parents at a level.

2. N is the set of children at a level.

3. M^j is the set of potential parents of sensor node j, such that $\forall i \in N : \bigcup_{i=1}^{|N|} M^i = M$.

4. $G = (M \bigcup N, E)$ is a bipartite graph such that if an edge $(i, j) \in E$, then $i \in M$ and $j \in N$ or vice-versa.

5. $|E| \leq M \times N$.

It is useful to abstract load balancing as a bandwidth allocation problem. Assume that a parent allocates its bandwidth (of 1 unit) equally to all its children. For example, if a parent has 3 children in the optimal tree then each child gets a bandwidth of $\frac{1}{3}$. In the optimal (load balanced tree), let x_i^* be the bandwidth allocated to a sensor node i. Consider the optimal allocation vector $x^* = (x_1^*, x_2^*, ...x_N^*)$, such that $\sum_{i=1}^{N} x_i^* = |M|$. Using simple arguments, it can be shown that the optimal (load balanced) allocation vector x^* is max-min fair.

Thus, the global objective is to attain a max-min fair allocation for all the children (at each level). The mechanism design for the load balanced tree is described over the next three sections, section 5.1.1 specifies the strategy space and the local utility function for each sensor node. The algorithm in described in section 5.1.2 and analyzed in section 5.1.3.

5.1.1 Game Description

We consider an iterative game. We define an iteration as a round in which parents announce their bandwidth guarantees and the children decide on the parent they want to attach. This game is played on the edges $e \in E$ of the bipartite graph mentioned in section 5.1. The players in this game are the children. For each player i such that $i \in N$,

let $S_k^i \subseteq M^i$ be the strategy space at iteration k. $u_k^i(p)$ denotes the utility of sensor node i for choosing sensor node p as its parent in iteration k. We define $u_k^i(p) = C_k^p$ where C_k^p is the bandwidth guaranteed by parent p at iteration k. i.e. the parent p is committed to provide a bandwidth of *at least* C_k^p to child i for all iterations after k as long as i is a child of p. Let

$$\begin{aligned} u_k^i &= \text{Max}_{p \in S_k^i}\{u_k^i(p)\} \\ &= \text{Max}_{p \in S_k^i}\{C_k^p\} \end{aligned} \tag{5.1}$$

This utility function in Eqn. 5.1 dictates that at each iteration, a sensor node prefers to connect to a parent that gives it the maximum bandwidth guarantee. i.e. each sensor node is "selfish" about the bandwidth guarantee it receives. At every iteration, each parent p gives equal bandwidth guarantees to its children.

In section 5.1.2, we describe the algorithm that constructs a load balanced tree in the presence of "selfish" sensor nodes.

5.1.2 Algorithm for Load Balanced Tree Construction

In this section, we describe *DistributedParentBid*, an iterative distributed algorithm for constructing a load balanced spanning tree in the network. The algorithm assumes that each sensor node's decision making is governed by the utility function described by Eqn. 5.1 in section 5.1.1. Before describing the algorithms of the parent and child, we

introduce some notations and definitions. As described in section 5.1.1, we define one iteration to be one round of communication between the parents and the children. This round consists of the following steps:

1. A candidate parent announces its bandwidth guarantee.

2. Each child responds to all candidate parents that offer the best bandwidth guarantee.

3. A candidate parent chooses a subset of the responding children informing them to attach to it.

4. Each child then chooses one of the candidate parents to attach and informs the rest that it does not want to attach to them.

Let

1. deg(j) be the degree of a node j in the bipartite graph G described in section 5.1.

2. P_k^i be the parent of node i at iteration k.

3. y_k^p be the number of additional children that parent p can take during iteration k, giving a bandwidth guarantee of C_k^p.

4. n_k^p be the number of **new** children that request to attach to parent p during iteration k.

5. a_k^p be the number of **new** children that actually attach to parent p during iteration k[1].

6. z_p^k be the number of children that leave parent p during iteration k.

7. N_k^p be the set of children currently attached to parent p at the end of iteration k.

8. B_k^p be the total bandwidth auctioned by a parent p at the end of iteration k. i.e.

$$B_k^p = |N_k^p| C_k^p \text{ if } |N_k^p| > 0 \tag{5.2}$$

$$= C_k^p \text{ otherwise} \tag{5.3}$$

Definition 5.1.1 *A parent p is considered to be saturated at iteration k iff $B_k^p = 1$.*

i.e. a parent p is saturated at iteration k iff its total auctioned bandwidth equals the maximum bandwidth of 1. On saturation at iteration k, parent p will offer a bandwidth guarantee of $C_k^p = \frac{1}{|N_k^p|}$ to each of the children attached to it.

If $|N_k^p| > 0$, saturation implies that p cannot take any more children at the guaranteed bandwidth of C_k^p. If $|N_k^p| = 0$, $B_k^p = C_k^p = 1$ implies that each of p's children have a bandwidth guarantee of 1 from some other parent p'. $\forall p$: $p \in M$, $y_k^p = 0$ iff p is saturated at iteration k.

DistributedParentBid consists of algorithms for **Parent** and **Child**, which are described below:

[1]A child might request to be attached to several parents during iteration k, but will ultimately choose exactly one parent that is willing to take it at the end of iteration k.

Parent: A parent p successively increases its bandwidth guarantees from $\frac{1}{deg(p)}$, $\frac{1}{deg(p)-1}$, $\frac{1}{deg(p)-2}$, so on until it gets saturated. In a system implementation, this can be achieved by the parent p broadcasting a message containing the bandwidth guarantee so that all children within its radio range can receive the bandwidth announcement. Each child that is not already attached to p, will respond to p if p guarantees the highest possible bandwidth among all its potential parents. If p cannot guarantee the announced bandwidth to all the children that respond, it will allow a subset of the responding children to select it as a parent and reject the rest. For example, consider a parent p that has 5 children currently. If p announces a bandwidth guarantee of $\frac{1}{7}$ and 3 additional children respond to it, p can only choose 2 out of the 3 children. This is because on choosing 2 children, p saturates with 7 children, each getting a bandwidth of $\frac{1}{7}$. An unsaturated p does not increase its bandwidth guarantee from iteration k to $k+1$ iff any of the following conditions are satisfied:

1. At least one of its current children switches to some other parent during iteration k.

2. At least one child that is not currently attached to it during iteration k requests to switch to it.

These conditions ensure that a parent p will increase its bandwidth guarantee iff no additional children want to switch to it at its current guarantee. While this is apparent for the second condition, the first condition is for letting a child that was rejected earlier by p to switch to it at the same bandwidth guarantee if possible (which may happen due to

some already attached child leaving p for some other parent). Parent p stops issuing the bandwidth announcement messages once it is saturated.

Initially, a parent p sets $C_k^p = \frac{1}{deg(p)}$. While p is not saturated (may also happen if a child leaves a saturated parent making $B_k^p < 1$), it executes the following iterative algorithm (here, k is a global variable that keeps track of iterations previously executed by this parent):

Parent(p) {

while $(B_k^p < 1)$ {

$C_{k+1}^p \leftarrow C_k^p$

announce C_{k+1}^p to all its children

/* *Find the upper bound on the number of children that can be taken* */

$y_{k+1}^p \leftarrow \frac{1}{C_{k+1}^p} - |N_k^p|$.

if $(n_{k+1}^p > y_{k+1}^p)$ reject the excess children

$|N_{k+1}^p| \leftarrow |N_k^p| + a_{k+1}^p - z_{k+1}^p$.

if $(B_{k+1}^p = 1)$ exit

/* *If a child left or responded to the announcement,*

don't increase the bandwidth guarantee */

if $((z_{k+1}^p > 0)$ or $(n_{k+1}^p > 0))$ { $C_{k+1}^p \leftarrow C_k^p$ }

else { $C_{k+1}^p \leftarrow \frac{1}{\frac{1}{C_k^p} - 1}$ }

$k \leftarrow k + 1$

}

}

Child: At an iteration k, the strategy space of child i, $S_k^i = \{p | p \in M^i, B_k^p < 1\}$. i.e. at each iteration k, each child i will try to connect to an unsaturated parent p that provides the best bandwidth guarantee. In a system implementation, this can be achieved by each child sending a message in response to the bandwidth announcement from a parent p. If there are multiple such parents, the child will request to connect to all of them[2]. If multiple parents are willing to accept it, the node will choose one of the parents and will inform the others that it does not wish to connect to them.

Termination: DistributedParentBid terminates at the smallest iteration k at which all parents are saturated i.e. $\sum_{p=1}^{M} B_k^p = M$.

5.1.3 Analysis

In this section, we show that *DistributedParentBid* algorithm terminates and analyze its running time. We show that a Nash Equilibrium exists on termination of the algorithm. We also outline a candidate scenario where this algorithm might not produce a load balanced tree. However, by comparing it against a centralized algorithm for load balancing through simulations, we show that the proposed algorithm produces a load balanced tree for around 90% of the scenarios studied.

[2]The child has to respond to all such parents because a previously unsaturated parent might have to reject some children if it gets saturated during the current iteration.

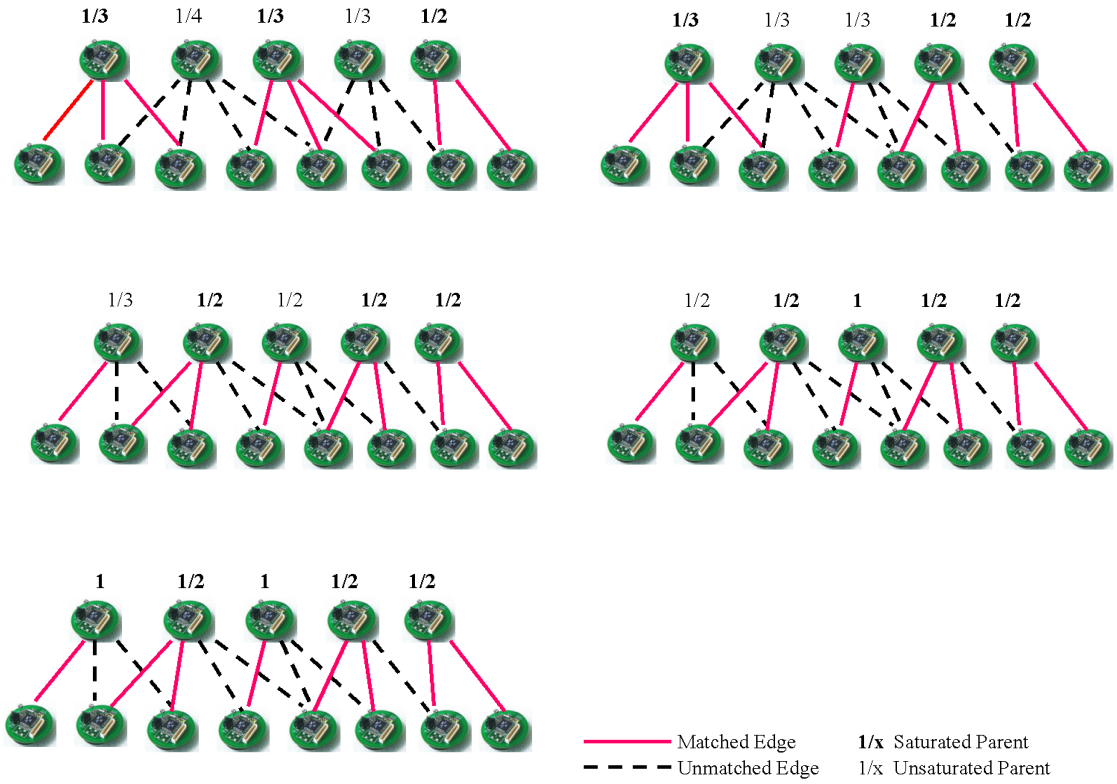


Figure 5.2: Load balancing level 2 of the tree shown in figure 5.1 (b) using *Distributed-ParentBid*.

Let $S_k = \sum_{i=1}^{M} B_k^i$ be the total auctioned bandwidth across all the parents at the end of iteration k. Let k^* be the smallest iteration at which all parents p are saturated i.e. $\sum_{p=1}^{M} B_{k^*}^p = M$.

Lemma 5.1.1 *The following quantities are non-decreasing functions of k:*

1. *For each parent p, C_k^p.*

2. *For each child i, u_k^i.*

Proof:

1. The algorithm *Parent(p)* never decreases C_k^p from one iteration to the next. Thus, $C_{k+1}^p \geq C_k^p$

2. By definition, $u_k^i = \text{Max}_{p \in S_k^i}\{C_k^p\}$. If i is attached to the same parent at iteration k and $k+1$, from the first part of this Lemma, $u_{k+1}^i \geq u_k^i$. i switches from p to p' during iteration $k+1$ iff $C_{k+1}^{p'} > C_{k+1}^p \geq C_k^p$. Hence, $u_{k+1}^i > u_k^i$. ■

Lemma 5.1.2 *If at least one child switches from one parent to the other during iteration* k, $S_k > S_{k-1}$.

Proof:

Let $N' \subseteq N$ be the set of children that switch and $N - N'$ be the set of children that do not switch parents. Then

$$\begin{aligned} S_k - S_{k-1} &\geq \sum_{i \in N} \{u_k^i - u_{k-1}^i\} \\ &\geq \sum_{i \in N} \{u_k^i - u_{k-1}^i\} + \sum_{j \in N - N'} \{u_k^j - u_{k-1}^j\} \end{aligned}$$

From Lemma 5.1.1, we observe that $u_k^i \geq u_{k-1}^i$ for all i. Specifically for $i \in N'$, $u_k^i > u_{k-1}^i$. Hence, if at least one child switches parents during iteration k ($N' \neq \phi$), then $S_k > S_{k-1}$. ■

Lemma 5.1.3 *If* $S_k > S_{k-1}$, $\delta = S_k - S_{k-1} \geq \frac{1}{(\gamma)(\gamma-1)}$, *where* $\gamma = Max_{p \in M}\{deg(p)\}$ *is the maximum degree of a parent in the graph* G .

Proof: If $S_k > S_{k-1}$, two cases arise:

1. A child i switches from parent p to q during iteration k.

2. No child switches, but an unsaturated parent p increases its bandwidth guarantee from C^p_{k-1} to C^p_k.

In the first case, a child i switches from parent p to q during iteration k iff $C^q_k > C^p_k$. Now, $C^p_k \geq C^p_{k-1}$ implies $u^i_k > u^i_{k-1}$. From the proof of lemma 5.1.2, we obtain that $S_{k-1} - S_k \geq \{u^i_k - u^i_{k-1}\}$. This difference is the smallest when exactly one child i switches from a parent p that offers it a bandwidth of $\frac{1}{\gamma}$ to a parent q that offers it a bandwidth of $\frac{1}{\gamma-1}$. Hence, the smallest difference $\delta = \frac{1}{(\gamma)(\gamma-1)} \leq S_k - S_{k-1}$. Hence, in general, if at least one node switches at iteration k, $S_k > S_{k-1}$ and difference $\delta = S_k - S_{k-1} \geq \frac{1}{(\gamma)(\gamma-1)}$. For the second case, we have the following:

$$\begin{aligned} S_k - S_{k-1} &\geq \sum_{q \in M} \{C^q_k - C^q_{k-1}\} \\ &\geq \{C^p_k - C^p_{k-1}\} \\ &\geq \frac{1}{\gamma - 1} - \frac{1}{\gamma} \\ &\geq \delta \end{aligned}$$

where p is an unsaturated parent. ■

Lemma 5.1.4 *The total auctioned bandwidth across all parents (S_k) is a non-decreasing function of k. Moreover, either $S_{k+1} > S_k$ or $S_{k+2} > S_{k+1} = S_k$.*

Proof:

Consider an iteration $k < k^*$. There exists at least one unsaturated parent p at the end of iteration k. Two possibilities arise:

1. None of the unsaturated parents increase their bandwidth guarantee during iteration $k+1$. Two sub-cases arise:

   (a) At least one child switches during this iteration then from Lemma 5.1.2, $S_{k+1} > S_k$.

   (b) No child switches during this iteration, then $S_{k+1} = S_k$. Now, during iteration $k+2$, all unsaturated parents increase their bandwidth guarantees. Again if a child switches parents, then $S_{k+2} > S_{k+1}$. If no child switches during iteration $k+2$ then,

   $$S_{k+2} - S_{k+1} \geq \sum_{j \in M} \{C^j_{k+2} - C^j_{k-1}\}$$

   Since $C^j_{k+2} > C^j_{k+1}$ for each unsaturated parent j, $S_{k+2} > S_{k+1} = S_k$.

2. At least one unsaturated parent increases its bandwidth guarantee during iteration $k+1$. Then, as seen from the proof of 1 (b), $S_{k+1} > S_k$. ■

We now prove our main result about the running time of *DistributedParentBid*.

Theorem 5.1.1 *The total number of iterations taken by DistributedParentBid is* $O(M\gamma^2)$. *At termination, a Nash Equilibrium exists.*

Proof:

Initially, each parent p announces a bandwidth guarantee of $C_0^p = \frac{1}{deg(p)} \geq \frac{1}{\gamma}$. Thus, $S_0 \geq \frac{M}{\gamma}$. If k^* is the smallest iteration at which all parents are saturated, $S_{k^*} = M$. From Lemma 5.1.4, we see that there are at most 2 iterations when S_k remains constant (i.e. k and $k+1$). Moreover, whenever S_k increases it increases by at least δ as seen from Lemma 5.1.3. Thus, the total number of iterations (T) for the termination of *DistributedParentBid* is given by:

$$\begin{aligned} T &\leq 2\{\frac{S_{k^*} - S_0}{\delta}\} \\ &\leq 2M\gamma^2 \end{aligned}$$

Next, we prove the existence of a Nash Equilibrium on the termination of *DistributedParentBid*.

On termination at iteration k^*, all parents are saturated i.e. $\forall p$, $B_{k^*}^p = 1$. Every saturated parent p offers a bandwidth of $\frac{1}{|N_{k^*}^p|}$ where $N_{k^*}^p$ is the number of children currently attached to it. Consider any child i. The current parent of i at iteration k^* is $q = P_{k^*}^i$. The bandwidth allocated by q to all its children (including i) is $\frac{1}{|N_{k^*}^q|}$. Then at iteration k^*, every other potential parent q' of i must have at least $|N_{k^*}^q| - 1$ children. This can be proved by contradiction. Let us assume that a parent q' is saturated with at most $|N_{k^*}^q| - 2$ children each getting a bandwidth of at least $\frac{1}{|N_{k^*}^q|-2}$. Since the bandwidth guaranteed by a parent is non-decreasing in the number of iterations as shown by Lemma 5.1.1, there would have been an iteration $k' < k^*$ such that q' would have offered a bandwidth of at

least $\frac{1}{|N_{k*}^q|-1}$ while q would have offered a bandwidth of at most $\frac{1}{|N_{k*}^q|}$. At this iteration i would have switched and attached to q'. But this contradicts the fact that i is attached to parent q and not q'. Thus, every potential parent q' of i must have at least $|N_{k*}^q| - 1$ children and can offer a bandwidth of at most $\frac{1}{|N_{k*}^q|}$ to i (if i switches from q to q'), which is the bandwidth offered by q. Hence, at termination each child is attached to the parent that allocates the highest possible bandwidth, implying the existence of a Nash Equilibrium.■

As mentioned in section 5.1, a load balanced tree corresponds to an optimal semi-matching. Our approach to proving the optimality of *DistributedParentBid* was based on the work of Harvey *et al.* which characterizes an optimal semi-matching using the notion of cost reducing paths (defined below) [21].

Definition 5.1.2 *Given a semi-matching $S \subseteq E$ in $G = (V, E)$, a cost reducing path R is a sequence of alternating matched and unmatched edges given by $R = \{(v_1, u_1), (u_1, v_2), \ldots (u_{z-1}, v_z)\}$ where $v_i \in M$, $u_i \in N$ and $(v_i, u_i) \in S$ for all i, such that $deg_S(v_1) - deg_S(v_z) > 1$. $deg_S(v_i)$ is the number of children matched to parent v_i in the semi-matching S.*

Moreover, the authors of [21] also prove the following:

Theorem 5.1.2 *A semi-matching is optimal iff no cost-reducing path exists.*

We now show that *DistributedParentBid* does not always produce a load balanced tree as shown in figure 5.3.

Notice that in figure 5.3 (b), the maximum load is 4, while in (c), it is 3. The above sub optimality occurs due to the existence of the cost reducing path A-B-C-D-E in the

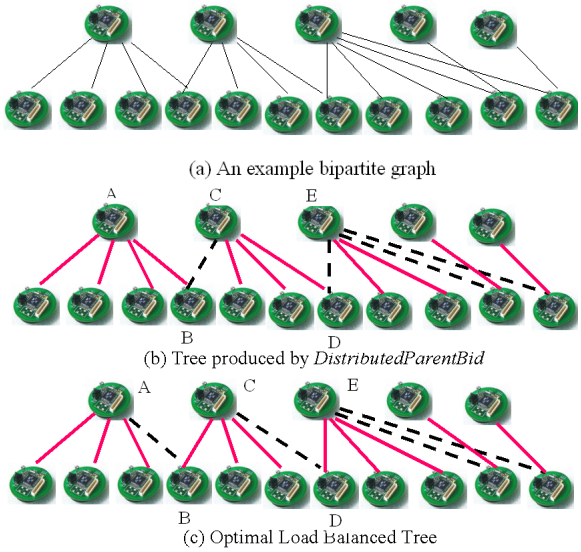


Figure 5.3: Sub optimality of *DistributedParentBid*.

bipartite graph i.e. the path consisting of parent A - child B - parent C - child D - parent E as shown in figure 5.3 (b). The matched and the unmatched edges along this path can be switched to obtain a better load balanced tree as shown in figure 5.3 (c). In general, the sub-optimality depends on the number of such paths existing in the bipartite graph after *DistributedParentBid* terminates. In the worst case scenario, the maximum load produced by *DistributedParentBid* can be a worse off by a factor of M from the optimum, where M is the number of parents i.e. the maximum load can be equally distributed across all the parents.

We attempted to quantify the sub-optimality (in random scenarios) by simulations where we compared the tree resulting from an implementation of *DistributedParentBid* and the centralized algorithm for producing an optimally load balanced tree described in [21]. 7 parent and 30 child nodes were randomly deployed in an area of 500m x 500m.

For each value of the transmission range used (250m and 200m), 5000 simulations (using random seeds) were run. In 90% of the simulation runs, *DistributedParentBid* produced trees that were isomorphic to the centralized algorithm, while in the remaining 10% of the simulations, the maximum (minimum) load produced by *DistributedParentBid* differed by a maximum of 1 from the centralized algorithm. These results give us reasonable confidence that *DistributedParentBid* is a good heuristic for load balancing.

5.2 Energy Balanced Data Gathering Tree Construction

In this section, we discuss the construction of a spanning tree in the network for a continuous data gathering application that assumes no data aggregation. An example of such a query is described in the introduction of this chapter. We consider an arbitrary deployment of n sensor nodes in the monitored terrain. Each sensor node samples one unit of data from the environment, which must be transmitted to the sink node in the network. Note that one unit of data can be one or several bytes. In order to route all the data, the sensor nodes form a spanning tree rooted at the sink.

Each leaf sensor node in the tree transmits one unit of data. The non-leaf sensor nodes in the tree route their own data as well as the data collected by all the sensor nodes in their subtree. Thus, each leaf sensor node dissipates one unit of energy, and the non-leaf sensor nodes dissipate energy proportional to the size of their subtree. To ensure the longevity of the network, our goal is to construct a spanning tree such that nodes with lower energy have smaller subtrees.

Let T denote a spanning tree constructed over the network, rooted at the sink node. Consider any node i in the spanning tree. Let e^i denote its current energy and $S(T,i)$ represent the size of its subtree. As discussed earlier, energy dissipation at a sensor node is proportional to the size of its subtree. After one iteration of data routing, the remaining energy of the sensor node is given by $e^i - S(T,i)$ [3]. To ensure longevity of the network, our goal is to construct an *energy balanced* spanning tree in the network, which maximizes a weighted sum of the remaining energy of all the sensor nodes such that nodes with lower energy get a higher weight [4]. Hence, our desired global objective function is to construct a spanning tree T' such that:

$$T' = \arg\max_{T}\{\sum_{i=1}^{n}\frac{(e^i - S(T,i))}{e^i}\}$$

Here $< e^1, ..., e^n >$ represent the current energy of the sensor nodes. Energy balancing, as defined here is a weaker notion of fairness than max-min. Figure 5.4 illustrates an energy balanced tree. The number on the sensor node shows the remaining energy, while the number on the edge shows the amount of data bytes transmitted. Each sensor node is assumed to generate one unit of data consisting of k bytes. Figure 5.4(a) depicts the tree resulting from an arbitrary parent selection strategy, while figure 5.4(b) shows the energy balanced tree. In Figure 5.4(b), notice the distribution of load commensurate to

[3] We assume that the remaining energy is always positive. i.e. all sensor nodes have sufficient energy to accomplish the data gathering.

[4] This objective is reasonable if we view the entire sensor network as a distributed energy source.

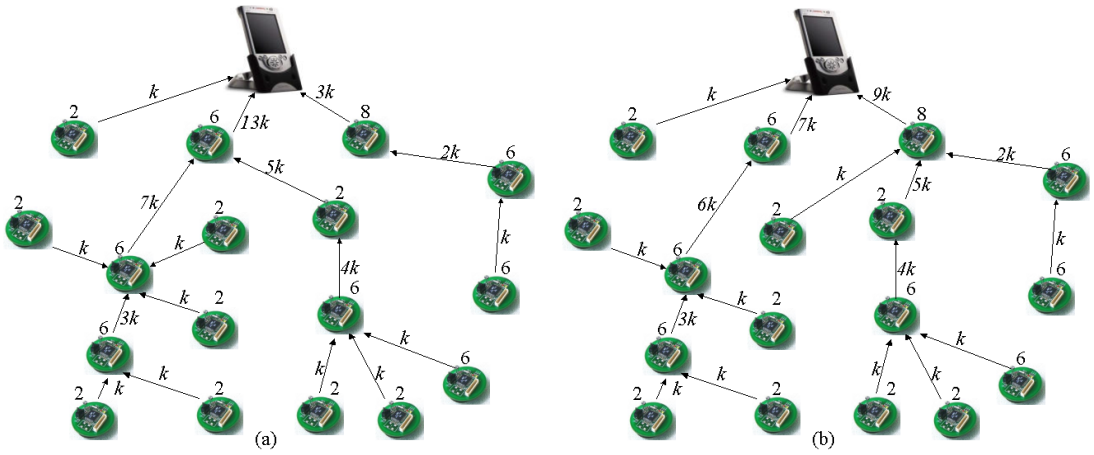


Figure 5.4: An illustration of energy balancing.

the energy on the 2 first hop nodes having energy of 6 and 8 units as opposed to figure 5.4(a).

We represent the network by the graph $G = (V, E, B)$, where the set of vertices represent the sensor nodes. An edge $(i, j) \in E$ exists between two vertices $v_i, v_j \in V$, provided that the sensor node represented by vertex v_j lies within the transmission range of the sensor node represented by vertex v_i. $B \in V$ denotes the sink node in the network. Theorem 5.2.1 states the key result that enables the design of a distributed mechanism for constructing an energy balanced tree.

Theorem 5.2.1 *Given the network graph $G = (V, E, B)$, construct a weighted directed graph $G' = (V, E_d, B, w)$ such that*

1. *If $(i, j) \in E$, then $(i, j) \in E_d$.*

2. $\forall (i, j) \in E_d, i \neq B : w(i, j) = \frac{1}{e^i}$.

The energy balanced tree is the sink-rooted Shortest Path Tree (SPT) in the weighted graph G'.

Proof:

Let $T' = (V, E', B)$ represent the SPT, and $T'' = (V, E'', B)$ be any other spanning tree for the graph $G' = (V, E_d, B, w)$, rooted at B. Let $P'_{i,B}$ and $P''_{i,B}$ denote the paths from any vertex i to B in T' and T'' respectively. The weights (length) associated with these paths is defined as follows.

- $w(P'_{i,B}) = \sum_{(i_1,i_2)\in P'_{i,B}} w(i_1, i_2)$.
- $w(P''_{i,B}) = \sum_{(i_3,i_4)\in P''_{i,B}} w(i_3, i_4)$.

Since, T' is the SPT for G', we know that

$$\forall i \in V, w(P'_{i,B}) \quad \leq \quad w(P''_{i,B}) \tag{5.4}$$

Summing up over all the nodes in G', we get

$$\sum_{i\in V} w(P'_{i,B}) \quad \leq \quad \sum_{i\in V} w(P''_{i,B}) \tag{5.5}$$

$$\sum_{i\in V} \sum_{(i_1,i_2)\in P'_{i,B}} w(i_1, i_2) \quad \leq \quad \sum_{i\in V} \sum_{(i_3,i_4)\in P''_{i,B}} w(i_3, i_4) \tag{5.6}$$

Interchanging the summations,

$$\sum_{(i_1,i_2)\in E'} N(T', (i_1, i_2)) * w(i_1, i_2) \quad \leq \quad \sum_{(i_3,i_4)\in E''} N(T'', (i_3, i_4)) * w(i_3, i_4) \quad (5.7)$$

where for each edge (y, z) in a spanning tree T, $N(T, (y, z))$ is the number of nodes in T for which (y, z) occurs on the path to B. In the SPT T', this path is the shortest path, while in an arbitrary spanning tree T'', this is not necessarily the shortest path to B. Since T is a tree, there is no alternative path in T for these nodes to reach B. Thus, these nodes will be in the sub tree rooted at a node y. Hence,

$$N(T, (y, z)) \quad = \quad S(T, y)$$

On replacing $N(T, (y, z))$ by $S(T, y)$ and $w(y, z)$ by $\frac{1}{e^y}$ in Eqn 5.7, we get

$$\sum_{i\in V} \frac{S(T', i)}{e^i} \quad \leq \quad \sum_{j\in V} \frac{S(T'', j)}{e^j} \quad (5.8)$$

Thus,

$$T' \quad = \quad \arg\max_{T} \{ \sum_{i=1}^{X} \frac{(e^i - S(T, i))}{e^i} \}$$

■

From the above analysis, we conclude that the desired energy balanced tree is the shortest path tree in the weighted graph G'. The shortest path tree for any given graph can be computed in polynomial time.

Theorem 5.2.1 gives us the key insight for designing local utility functions for the sensor nodes for achieving our global objective. If each sensor node i "selfishly" routes its data to the sink over the shortest path using $\frac{1}{e^i}$ as the edge length metric, it results in the desired energy balanced data aggregation tree. The cost function $c(i,j) = \frac{1}{e^i}$ has the nice property that sensor nodes with lower energy will have a higher cost for their outgoing link and hence are less likely to have a large sub tree rooted at them. This ensures longevity of the data gathering tree.

We describe a simple mechanism for constructing an energy balanced tree in the following section.

5.2.1 Mechanism Design

In this section, we describe a mechanism for constructing the energy balanced data gathering tree. Consider an iterative game on the edges of the graph $G = (V, E, B)$. Since we are interested in balancing the utilization of all the nodes in the network, this graph corresponds to the entire deployment topology (unlike the bipartite graph used in the load balanced tree construction in section 5.1.1).

The strategy space S_κ^i of each sensor node i at iteration κ is given as follows:

$$S_\kappa^i = \{j|(i,j) \in E\} \tag{5.9}$$

The utility function of each sensor node i is given as follows:

$$\begin{aligned} u_\kappa^i &= \text{Min}_{j \in S_\kappa^i}\{\frac{1}{e^i} + Q_\kappa^j\} \\ &= \text{Min}_{j \in S_\kappa^i}\{Q_\kappa^j\} \end{aligned} \tag{5.10}$$

where Q_κ^j is the length of the shortest path from sensor node j to the sink B at iteration κ. $\forall \kappa : Q_\kappa^{sink} = 0$. Thus, at each iteration, a sensor node i chooses a node j that offers it the shortest path (using $\frac{1}{e^z}$ as the metric of edge length for any edge $(z,w) \in E$) to B.

The above mechanism can be easily implemented by a distributed distance vector algorithm. An iteration κ corresponds to an iteration of the distance vector algorithm. The energy balanced tree construction terminates when the distance vector algorithm terminates.

5.3 Summary

In this chapter, we advocate a utility function based approach for designing sensor networks. Unlike previous related studies, we illustrate that treating the sensor nodes as

"selfish" (using appropriate utility functions) enables the design of distributed algorithms for optimizing the network performance as a "whole". This approach is illustrated by two case studies of constructing a load balanced data gathering tree and an energy balanced data gathering tree in a sensor network. The second case study uses a weaker notion of fairness than the first, but it enables the optimization of a global objective using local optimization of an appropriate utility function at each sensor.

As part of future work, it would be interesting to extend our mechanism of load balancing and energy balancing for more general scenarios that account for heterogeneity in quality of the links, more sophisticated models of data aggregation, etc. We also plan to investigate other desired global objectives of data collection trees for which decentralized mechanisms using local utility functions can be designed.

Chapter 6

Contributions

In this dissertation, we examined a few practical problems concerning data gathering in wireless sensor networks. These problems were chosen such that they explore complementary aspects of data gathering: one-shot (chapter 3), *en masse* (chapter 4) and continuous (chapter 5). The severe energy constraints of the sensor nodes motivated us to adopt a principled approach to these problems based on mathematical modeling and optimization. Using various mathematical tools that included first order models, Markov chains, linear programs and game theory, we gained several useful insights in the design of algorithms for these data gathering problems. The analytical approach not only led to the design of optimal algorithms but also outlined scenarios where traditional solutions to these problems may not be optimal.

Our key contributions are the following:

One-shot data gathering

- Development of ACQUIRE (that combines d hop FBQ with TBQ), a mechanism for answering one-shot queries for replicated data in sensor networks.
- Analytical proof that the optimal choice of d that minimized the expected number of transmissions is governed by the data dynamics. Higher the data dynamics, lower the optimal value for d.
- A unified analytical framework for analyzing existing approaches like FBQ and ERS, validated by simulations.
- Demonstrated scenarios where ACQUIRE with optimal setting for d results in significant energy savings over ERS (up to $60-75\%$) and orders of magnitude savings over FBQ (especially in highly dynamic environments and high query rates).

En Masse data gathering

- Formulation of the maximum data extraction as a multi commodity flow problem and the design of a $(1+\omega)$ approximation algorithm for it.
- A fast, near optimal, distributed heuristic E-MAX for *en masse* data gathering. E-MAX used a sophisticated edge length metric (obtained from an analytical model based on multi commodity flows and LP duality) that is both energy-aware and data-aware.

- Comparison of E-MAX with other energy-aware approaches through simulations which shows that E-MAX significantly outperforms these other approaches (up to 450 % improvement in the amount of data collected across scenarios studied), especially when nodes have varying data and energy levels.

Continuous data gathering

- Proposed a novel approach to designing distributed algorithms for gathering data from a sensor network using concepts of utility functions and selfishness.

- Design of utility functions for the global objectives of load balancing and energy balancing respectively such that while each sensor selfishly optimizes its utility function, the network as a "whole" converges to the desired global objective.

This dissertation takes a small step in the direction of designing data gathering algorithms for energy-constrained wireless sensor networks using a principled approach based on mathematical modeling and optimization. The design insights learnt from our case studies indicate that this formal approach is quite promising and can lead to algorithms with significantly better performance over those that are purely based on heuristics. As the field of sensor networking matures there will be many more avenues for improvement in both the analytical modeling as well as the design of these systems. The formal, mathematical approach not only gives optimal baselines against which different approaches can be evaluated, but also reveals the science underlying the optimal solutions, leading to better system design.

Chapter 7

Mathematical Background

In this chapter, we give a brief introduction to the various mathematical techniques used in this thesis including probability, Markov chains, linear program duality and game theory.

7.1 Probability

This section gives an overview of the basic concepts in probability and random variables. Section 7.1.1 introduces the notion of probability, while section 7.1.2 discusses the concept of random variables.

7.1.1 Introduction

In this section, we define some basic concepts in probability theory using a set theoretic approach based on the material in [40]. We first start with a notion of an experiment and its associated outcomes. For example, in an experiment of tossing a die, the set

of possible outcomes is given by the set $\{1, 2, 3, 4, 5, 6\}$ i.e. the numbered face that shows up on a single throw of a die. Some basic definitions is probability theory are the following:

Definition 7.1.1 *Sample Space: The set Λ of all possible outcomes of an experiment is the sample space for that experiment.*

For the single throw of a die, the sample space $\Lambda = \{1, 2, 3, 4, 5, 6\}$.

Definition 7.1.2 *Trial: A single performance of an experiment is called a trial. At each trial we observe a single outcome $\lambda_i \in \Lambda$.*

Definition 7.1.3 *Event: A subset of a sample space is an event.*

For example, getting an even number on a single throw of a die corresponds to the event $Even = \{2, 4, 6\}$.

Definition 7.1.4 *Mutually Exclusive Events (MEE): Events A and B are mutually exclusive iff $A \cap B = \emptyset$ i.e. these events do not have any outcome in common.*

For example, the events of getting an event number i.e. $Even = \{2, 4, 6\}$ and getting an odd number $Odd = \{1, 3, 5\}$ on a single throw of a die are mutually exclusive.
Every event $A \subseteq \Lambda$ is assigned a number $P(A)$, the probability of event A. The probability of an event has the following properties:

1. $P(A) \geq 0$.

2. $P(\Lambda) = 1$.

3. $P(\emptyset) = 0$.

4. For mutually exclusive events A and B, $P(A \cup B) = P(A) + P(B)$.

Assuming a fair die in which each numbered face shows up with equal probability on a single throw, $P(\{1\}) = P(\{2\}) = P(\{3\}) = P(\{4\}) = P(\{5\}) = P(\{6\}) = \frac{1}{6}$.

Definition 7.1.5 *Independent Events: Events A and B are independent iff*

$$P(A \cap B) \quad = \quad P(A)P(B)$$

Intuitively, two events are independent, if the fact that A occurs does not change the probability of B occurring. For example, getting a head on a single coin toss is independent of getting a 6 on a single throw of a die.

Definition 7.1.6 *Conditional Probability: The conditional probability of event A assuming that event B has occurred is given by:*

$$P(A|B) \quad = \quad \frac{P(A \cap B)}{P(B)}$$

For example, the conditional probability of getting a two given that an even numbered face showed up is $\frac{P(\{2\} \cap \{2,4,6\})}{P(\{2,4,6\})} = \frac{1}{3}$.

Definition 7.1.7 *Total Probability: If* $A_1, A_2, ..., A_n$ *are such that* $\cup_{i=1}^{n} A_i = \Lambda$ *and* $\cap_{i=1}^{n} A_i = \emptyset$ *and* B *is an arbitrary event over the sample space* Λ*, then*

$$P(B) = \sum_{i=1}^{n} P(B|A_i)$$

7.1.2 Random Variables

In this section, we will mainly focus on discrete random variables. Similar definitions for continuous random variables can be found in [40].

Definition 7.1.8 *Discrete random variable* X *is a function that maps a finite or a countably infinite sample space* Λ *to a set of real numbers. i.e.* $X : \Lambda \rightarrow \Re$.

For a random variable X and a real number x, we define an event $X = x$ as $\{\lambda \in \Lambda : X(\lambda) = x\}$. Hence, the probability that X takes on the value x is given by

$$P(X = x) = \sum_{\lambda \in \Lambda : X(\lambda) = x} P(\{\lambda\}) \tag{7.1}$$

Definition 7.1.9 *Probability Density Function (PDF) of* X *is the function*

$$f(x) = P(X = x)$$

Definition 7.1.10 *Cumulative Distribution Function (CDF) of X is the function*

$$\begin{aligned} F(x) &= P(X \leq x) \\ &= \sum_{y \leq x} P(X = y) \end{aligned}$$

Definition 7.1.11 *The joint PDF of two variables X and Y is given by*

$$f(x, y) = P(X = x, Y = y)$$

Definition 7.1.12 *Independence: Random variables X and Y are independent iff*

$$P(X = x, Y = y) = P(X = x)P(Y = y)$$

Definition 7.1.13 *Conditional Probability: The probability of X given Y is*

$$P(X = x|Y = y) = \frac{P(X = x, Y = y)}{P(Y = y)}$$

One of the most useful summaries of the distribution of a random variable X is the expectation (mean or average) which is given as follows

$$E(X) = \sum_{x} xP(X = x) \tag{7.2}$$

Expectation is also defined for combinations of random variables. Particularly, for a sum of random variables, the following theorem shows the relationship between the expectation of the sum and the expectation of the individual random variables.

Theorem 7.1.1 *Linearity of Expectation: The expected value of a sum of random variables X and Y is given by*

$$E(X+Y) \quad = \quad E(X) + E(Y)$$

The above relationship holds even if X and Y are not independent. It also extends to summation of an arbitrary number of random variables.

An example of a discrete random variable is the geometric random variable. Consider an experiment in which each trial has only two outcomes, e.g. a coin flip. Such a trial is called a **Bernoulli trial**. In each trial, let one of the outcomes (say **success**) occur with probability p and the **failure** occur with probability $q = 1 - p$. Define a random variable X to be the number of trials needed to obtain the first success. X has values in the range $1, 2, ...$ The PDF of X is given by

$$P(X = k) \quad = \quad q^{k-1}p \tag{7.3}$$

since we have $k - 1$ failures before the first success.

Definition 7.1.14 *Geometric Distribution: A PDF satisfying equation 7.3 is said to be a geometric distribution. X is said to be a geometric random variable.*

Theorem 7.1.2 *The expected value $E(X)$ of a geometric random variable X is $\frac{1}{p}$.*

7.2 Markov Chains

We first give a brief overview of Markov chains in section 7.2.1, followed by a classification of the states of a Markov chain in section 7.2.2. Section 7.2.3 describes an important result in the analysis of Markov chains with absorbing states.

7.2.1 Introduction

We restrict our discussion to discrete time Markov chains based on the simple and detailed description in [35]. We start with an intuitive definition of a stochastic process. Suppose we observe some characteristic of a system at discrete points in time (say $0, 1, 2, ...$). Let X_t be the value (or state) of the system characteristic at time t. Quite often, X_t is not known with certainty and may be viewed as a random variable. A discrete-time stochastic process is a description of the relation between the random variables $X_0, X_1, X_2, ...$ A Markov chain is a discrete-time stochastic process which is defined as follows:

Definition 7.2.1 *A discrete-time stochastic process is a Markov chain if, for $t = 0, 1, 2, ...$ and all states,*

$$P(X_{t+1} = i_{t+1}|X_t = i_t, X_{t-1} = i_{t-1}, ..., X_1 = i_1, X_0 = i_0)$$
$$= P(X_{t+1} = i_{t+1}|X_t = i_t)$$

i.e. the probability distribution of X_{t+1} is dependent only on the state at time t and is independent of all states visited by the chain before time t.

Example: Consider a sequence of flips of a fair coin. The number of heads after n flips i.e. H_n can be modeled as a Markov chain with the following equation:

$$P(H_{n+1} = h + 1 | H_n = h) = \frac{1}{2}$$
$$P(H_{n+1} = h | H_n = h) = \frac{1}{2}$$

For simplicity, assuming that $P(X_{t+1} = j | X_t = i)$ is independent of time t (also called the stationarity assumption), we define the probability of transition from state i to state j as follows:

$$r_{ij} = P(X_{t+1} = j | X_t = i) \tag{7.4}$$

A complete description of a Markov chain of s states is given by the probability distribution over the states at time $t = 0$ i.e. $P(X_0 = s)$ for all states s and a $s \times s$ state transition matrix R where $R_{ij} = r_{ij}$, the state transition probability from state i to state j. Each entry in this matrix is non-negative and the entries in each row should sum to 1 i.e. $\sum_{j=1}^{s} r_{ij} = 1$.

7.2.2 Classification of States

Before classifying the states of a Markov chain, we define the concept of reachability between states:

Definition 7.2.2 *Given two states i and j, a path from i to j is a sequence of transitions that begins in state i and ends in state j, such that each transition in the sequence occurs with positive probability. A state j is reachable from state i if there is a path leading from i to j.*

The states of the Markov chain can be classified as absorbing, transient, recurrent, periodic and aperiodic.

Definition 7.2.3 *Absorbing State: State i is an absorbing state if $r_{ii} = 1$.*

Once the chain reaches a state i that is absorbing, it always remains in state i.

Definition 7.2.4 *Transient State: State i is transient if there exists a state j such that j is reachable from i but i is not reachable from j.*

For a transient state i, there is a way to leave state i without returning to it, which implies that $r_{ii} < 1$.

Definition 7.2.5 *Recurrent State: A non transient state is a recurrent state.*

Definition 7.2.6 *Periodic State: A state i is periodic with period $k > 1$ if k is the smallest number such that all paths leading from state i back to itself have a length that is a multiple of k.*

Definition 7.2.7 *Aperiodic State: A non-periodic recurrent state is said to be aperiodic.*

7.2.3 Analyzing Time to Absorption

Markov chains have been extensively analyzed for their transient and steady state behavior. In this section, we briefly describe a key result in the analysis of Markov chains with absorbing states.

Quite often it is of interest to find out what is the expected time it takes for a chain (with s states) starting at a transient state i to reach the absorption state. This time is also called the time to absorption from state i. It is known that starting at a transient state i, the expected number of times each transient state j is entered is given by ij entry in $(I-R)^{-1}$, where I is an $s \times s$ identity matrix [35]. Thus, for a Markov chain with a single absorbing state, using linearity of expectation, the expected time to reach the absorption state is given by the following matrix equation:

$$A = (I - R)^{-1}E \tag{7.5}$$

where A and E are $s \times 1$ column vectors. E is a vector of all 1s. A_i gives the expected time to absorption from state i.

7.3 Linear Program (LP) Duality

A linear program typically consists of optimizing (maximizing or minimizing) a linear function of variables subject to some constraints on those variables. Corresponding to every LP (or primal), there is another LP called the dual. In this section, we give a

simple introduction to the concept of LP-duality which is very useful in solving several combinatorial optimization problems. Most of the material in this section is based on [55]. Using a sample LP which has a minimization objective function, we first define a few important terms before illustrating the concept of LP-duality. These concepts can be extended in a straightforward manner to a LP with a maximization objective function.

Consider the following LP:

Minimize $z = 5x_1 + 7x_2 + 2x_3$

subject to

$$\begin{aligned} 2x_1 + 4x_2 + 3x_3 &\geq 12 \\ x_1 + 2x_2 - x_3 &\geq 5 \\ x_1,\ x_2,\ x_3 &\geq 0 \end{aligned} \tag{7.6}$$

A solution to the above LP consists of setting values for the variables x_1, x_2 and x_3.

Definition 7.3.1 *Feasible Solution: A solution $x = \{x_1, x_2, x_3\}$ that satisfies all the constraints of the LP is said to be feasible.*

Definition 7.3.2 *Optimal Solution: A solution $x^* = \{x_1, x_2, x_3\}$ that is feasible and optimizes the given linear objective function is the optimal solution of the LP. The optimal value of the linear objective function is denoted by z^*.*

While the objective of the LP in Eqn. 7.6 is to obtain the minimum z^*, a simple lower bound on z^* can also be obtained by adding the right hand side of the two constraints giving the following:

$$\begin{aligned} z = 5x_1 + 7x_2 + 2x_3 &\geq (2x_1 + 4x_2 + 3x_3) + (x_1 + 2x_2 - x_3) \\ &\geq 3x_1 + 6x_2 + 2x_3 \\ &\geq 12 + 5 \\ &\geq 17 \end{aligned} \tag{7.7}$$

The basic idea behind the process of finding a lower bound for z^* is to find suitable non-negative multipliers for each of the constraints in the primal LP such that when we take their sum, the coefficient of each x_i in the sum is dominated by the corresponding coefficient in the linear objective function z. The lower bound should be as tight as possible (for the minimization problem in the primal, the lower bound should be as large as possible). This can be naturally formulated as another linear program called the dual LP. For the primal LP in Eqn. 7.6, the dual LP is as follows:

Maximize $12y_1 + 5y_2$

subject to

$$\begin{aligned} 2y_1 + y_2 &\leq 5 \\ 4y_1 + 2y_2 &\leq 7 \\ 3y_1 - y_2 &\leq 2 \\ y_1, y_2 &\geq 0 \end{aligned} \tag{7.8}$$

In the above example, y_1 and y_2 are the multipliers for the first and second constraint of the primal LP in Eqn. 7.6. There are several interesting properties that relate the primal and dual LPs. If the primal is a minimization problem, the dual is a maximization problem. The dual of the dual is the primal LP. A feasible solution to the dual LP gives a lower bound on the optimal value z^* of the primal. Similarly, a feasible solution to the primal gives an upper bound on the optimal value of the dual. Hence if we find feasible solutions for both the primal and the dual LP with the same value for the respective objective functions, then both solutions must be optimal for the respective problems. In our example, $x^* = \{0, 2.7, 0.4\}$ and $y^* = \{1.1, 1.3\}$ both achieve the objective function value of 19.7 for the primal and dual respectively. This relationship between the optimal solutions of the primal and dual LPs is one of the main results in the theory of linear programming and is known as the *LP-duality theorem*. Before stating the theorem, we

formally define the primal and dual LPs. Consider the following minimization problem written in standard form:

Minimize $\sum_{j=1}^{n} c_j x_j$

subject to

$$\begin{aligned} \forall i \in [1, m] : \sum_{j=1}^{n} a_{ij} x_j &\geq b_i \\ \forall j \in [1, n] : x_j &\geq 0 \end{aligned} \tag{7.9}$$

where a_{ij}, b_i and c_j are given rational numbers.

Using y_i as the variable (multiplier) for the i'th inequality, we get the following dual LP:

Maximize $\sum_{i=1}^{m} b_i y_i$

subject to

$$\begin{aligned} \forall j \in [1, n] : \sum_{i=1}^{m} a_{ij} y_i &\leq c_j \\ \forall i \in [1, m] : y_i &\geq 0 \end{aligned} \tag{7.10}$$

The *LP-duality* theorem is formally stated as follows:

Theorem 7.3.1 *LP-Duality: The primal LP has a finite optimum iff its dual has a finite optimum. Moreover, if* $x^* = \{x_1^*, x_2^*, ..., x_n^*\}$ *and* $y^* = \{y_1^*, y_2^*, ..., y_m^*\}$ *are the optimal solutions for the primal and dual LPs respectively, then*

$$\sum_{j=1}^{n} c_j x_j \quad = \quad \sum_{i=1}^{m} b_i y_i$$

The conditions for the feasible solutions of the primal (and dual) to be optimal are given by the *Complementary slackness conditions* which are formally described as follows:

Theorem 7.3.2 *Complementary slackness conditions: Let* x *and* y *be feasible solutions for the primal and dual LPs respectively.* x *and* y *are both optimal iff all of the following conditions are satisfied:*

Primal complementary slackness conditions

$$\forall j \in [1, n] : \textit{either } x_j = 0 \textit{ or } \sum_{i=1}^{m} a_{ij} y_i = c_j$$

and

Dual complementary slackness conditions

$$\forall i \in [1, m] : \textit{either } y_i = 0 \textit{ or } \sum_{j=1}^{n} a_{ij} x_j = b_i$$

The complementary slackness says that either the primal (dual) variable is 0 or the constraint corresponding to that variable in the dual (primal) is exactly satisfied.

LP-duality and complementary slackness play a very important role in the design of algorithms (both exact and approximation) for combinatorial optimization problems.

7.4 Game Theory

Typically, game theory studies interactions of entities (individuals, agents, countries, etc.) that have opposing (or competing) interests. In this section, we restrict our discussion to the non-cooperative games in which each entity behaves selfishly and is interested in optimizing its payoff (gain). We first give a very simple and intuitive discussion of some important concepts in game theory in section 7.4.1 and conclude with a few formal definitions in section 7.4.2. Most of this discussion can be found in [16] [38].

7.4.1 Introduction

One of the most common motivating examples for game theory is the Prisoner's dilemma, that involves the following situation: "Two burglars, Bob and Al, are captured near the scene of a burglary and are given the "third degree" separately by the police. Each has to choose whether or not to confess and implicate the other. If neither man confesses, then both will serve one year on a charge of carrying a concealed weapon. If each confesses and implicates the other, both will go to prison for 10 years. However, if one burglar confesses and implicates the other, and the other burglar does not confess, the one who has collaborated with the police will go free, while the other burglar will go to prison for 20 years on the maximum charge."

Each prisoner is faced with the dilemma of choosing from one of the two strategies: Confess (C) or Deny (D). The associated payoff is the sentence in prison. Each burglar is interested in minimizing his term in prison. The entire situation can be compactly summarized using a "payoff table" as shown in table 7.1:

Table 7.1: Prisoner's Dilemma

	C	D
C	(10,10)	(0,20)
D	(20,0)	(1,1)

The above table can be interpreted as follows: each prisoner chooses one of the two strategies. In effect, Bob chooses a column and Al chooses a row. The two numbers in each cell tell the outcomes for the two prisoners when the corresponding pair of strategies is chosen. The number to the left of the comma tells the payoff to the person who chooses the rows (Al) while the number to the right of the column tells the payoff to the person who chooses the columns (Bob). Thus (reading down the first column) if they both confess, each gets 10 years, but if Al confesses and Bob denies, Bob gets 20 years in prison and Al goes free.

It is of interest to observe how each prisoner may overcome the dilemma of "To Confess or Not to Confess".

Al might reason as follows: "Two things can happen: Bob can confess or deny. Suppose Bob confesses. Then I get 20 years in prison if I deny, 10 years if I do, so in that case it's best to confess. On the other hand, if Bob denies, and I deny too, I get a year;

but in that case, if I confess I can go free. Either way, it's best if I confess. Therefore, I'll confess."

A similar line of reasoning will be followed by Bob too. Thus both Al and Bob decide that it is in their best interests to confess. This results in both of them going to prison for 10 years. Ironically if both of them had decided to deny, both of them would have gone to prison only for a year! This example shows that the outcome resulting from each entity trying to optimize its gains may not necessarily be the socially desired outcome (or one that is most beneficial to all the entities). Informally, the outcome that results from each entity trying to optimize its gain is called as the *Nash Equilibrium*, which is formalized in section 7.4.2.

7.4.2 Formal Definitions

Definition 7.4.1 *Strategic Game: A strategic game consists of:*

1. n *players (entities or agents).*
2. *For each player* i*, a strategy space* S_i *that defines the set of possible strategies (actions).*
3. *Set of possible outcomes* $S \subseteq S_1 \times S_2 \times ...S_n$*.*
4. *A utility function* u_i *for each player* i*, such that* $u_i : S \rightarrow \Re$ *i.e. a function that indicates* i*'s payoff corresponding to a particular outcome.*

We now define the Nash Equilibrium as follows:

Definition 7.4.2 *Nash Equilibrium: A strategy profile* $(s_1^*, s_2^*, ..., s_n^*) \in S$ *is a Nash Equilibrium iff*

$$\forall i \in [1, n] : \forall s_i \in S_i : u_i(s_1^*, s_2^*, ..., s_{i-1}^*, s_i^*, ...s_n^*) \quad \geq \quad u_i(s_1^*, s_2^*, ..., s_{i-1}^*, s_i, ...s_n^*)$$

i.e. at Nash Equilibrium, each player i has no incentive to change strategy unilaterally.

In case of the Prisoner's dilemma:

1. The number of players $n = 2$.

2. The strategy spaces for player 1 and 2 are $S_1 = S_2 = \{C, D\}$.

3. The possible outcomes as shown in table 7.1 are

   $S = \{(C, C), (C, D), (D, C), (D, D)\}$.

4. The utility functions for the players are:

$$u_1(C, C) = -10, u_1(C, D) = 0, u_1(D, C) = -20, u_1(D, D) = -1.$$

$$u_2(C, C) = -10, u_2(C, D) = -20, u_2(D, C) = 0, u_1(D, D) = -1.$$

Let us examine the strategy of player 1 (Al) at Nash Equilibrium:

1. If player 2 (Bob) confesses, $u_1(C, C) > u_1(D, C)$, Al should confess.

2. If Bob denies, $u_1(C, D) > u_1(D, D)$, Al should confess.

Thus at Nash Equilibrium, no matter what Bob does, Al should confess. Likewise for Bob.

In this particular game, each player has exactly one optimal strategy (i.e. confess), irrespective of the strategy adopted by the other player. This leads to a unique Nash Equilibrium. More generally the optimal strategy of a player may be different based on the strategies adopted by other players. This may result in multiple Nash Equilibria. Thus a strategic game can have none, one or multiple Nash Equilibria. These are extensively discussed in [16] [38].

Apart from analyzing the Nash Equilibria arising from the interaction of selfish agents (with known utility functions), it is also of interest to devise mechanisms such that these selfish agents converge to a desired global objective. Algorithmic Mechanism Design (AMD) deals with the design of such mechanisms [37]. An excellent overview of the applicability of Nash Equilibria and AMD to the Internet is presented in [39].

Reference List

[1] M. Bhardwaj and A. P. Chandrakasan. Bounding the Lifetime of Sensor Networks Via Optimal Role Assignments. In *Proceedings of IEEE Infocom*, June 2002.

[2] P. Bonnet, J. Gehrke, and P. Seshadri. Towards Sensor Database Systems. In *Proceedings of ACM/IEEE MDM*, 2001.

[3] P. Bonnet, J. E. Gehrke, and P. Seshadri. Querying the Physical World. *IEEE Personal Communications*, 7(5), October 2000.

[4] D. Braginsky and D. Estrin. Rumor Routing Algorithm for Sensor Networks. In *Proceedings of WSNA*, September 2002.

[5] T. X. Brown, H. N. Gabow, and Q. Zhang. Maximum Flow-Life Curve for a Wireless Ad Hoc Network. In *Proceedings of ACM Mobihoc*, October 2001.

[6] J. Byers and G. Nasser. Utility-Based Decision-Making in Wireless Sensor Networks. In *Proceedings of IEEE Mobihoc*, 2000.

[7] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat Monitoring: Application Driver for Wireless Communications Technology. In *Proceedings of ACM Sigcomm Workshop on Data Communications in Latin America and the Caribbean*, April 2001.

[8] J. H. Chang and L. Tassiulas. Energy Conserving Routing in Wireless Ad-Hoc Networks. In *Proceedings of IEEE Infocom*, March 2000.

[9] J. H. Chang and L. Tassiulas. Maximum Lifetime Routing in Wireless Sensor Networks. *IEEE/ACM Transactions on Networking*, 12(4):609–619, August 2004.

[10] N. Chang and M. Liu. Revisiting the TTL-Based Controlled Flooding Search: Optimality and Randomization. In *Proceedings of ACM MobiCom*, September 2004.

[11] Z. Cheng and W. Heinzelman. Flooding Strategy for Target Discovery in Wireless Networks. In *Proceedings of ACM MSWiM*, September 2003.

[12] M. Chu, H. Haussecker, and F. Zhao. Scalable Information-Driven Sensor Querying and Routing for Ad hoc Heterogeneous Sensor Networks. *International Journal of High Performance Computing Applications*, 2002.

[13] T. H. Cormen, C. E. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw-Hill, second edition, 1998.

[14] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the World with Wireless Sensor Networks. In *Proceedings of ICASSP*, May 2001.

[15] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *Proceedings of ACM/IEEE MobiCom*, August 1999.

[16] Game Theory. http://william-king.www.drexel.edu/top/eco/game/game-toc.html.

[17] S. Gandham, M. Dawande, and R. Prakash. An Integral Flow-Based Energy-Efficient Routing Algorithm for Wireless Sensor Networks. In *Proceedings of IEEE WCNC*, March 2004.

[18] J. Gao, L. J. Guibas, J. Hershberger, and L. Zhang. Fractionally Cascaded Information in a Sensor Network. In *Proceedings of IPSN*, April 2004.

[19] N. Garg and J. Konemann. Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems. In *Proceedings of FOCS*, 1998.

[20] 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.

[21] N. J. Harvey, R. E. Ladner, L. Lovasz, and T. Tamir. Semi-Matchings for Bipartite Graphs and Load Balancing. In *Proceedings of WADS*, 2003.

[22] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In *Proceedings of HICSS*, January 2000.

[23] W. R. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. In *Proceedings of ACM MobiCom*, August 1999.

[24] A. Helmy, S. Garg, P. Pamu, and N. Nahata. CARD: A Contact-Based Architecture for Resource Discovery in Ad Hoc Networks. *Special issue of ACM Baltzer MONET Journal on Algorithmic Solutions for Wireless, Mobile, Ad Hoc and Sensor Networks*, 2004.

[25] Y. T. Hou, Y. Shi, and H. D. Sherali. On Lexicographic Max-Min Node Lifetime Problem for Energy-Constrained Wireless Sensor Networks. Technical report, The Bradley Department of ECE, Virginia Tech, September 2003.

[26] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of Network Density on Data Aggregation in Wireless Sensor Networks. In *Proceedings of ICDCS*, July 2002.

[27] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Proceedings of ACM/IEEE MobiCom*, August 2000.

[28] K. Kalpakis, K. Dasgupta, and P. Namjoshi. Efficient Algorithms for Maximum Lifetime Data Gathering and Aggregation in Wireless Sensor Networks. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 42(6):697–716, August 2003.

[29] R. Kannan, S. Sarangi, S. S. Iyengar, and L. Ray. Sensor Centric Quality of Routing in Sensor Networks. In *Proceedings of IEEE Infocom*, 2003.

[30] S. Lindsey, C. Raghavendra, and K. Sivalingam. Data Gathering Algorithms in Sensor Networks Using the Energy*Delay Metric. In *Proceedings of IPDPS Workshop on Issues in Wireless Networks and Mobile Computing*, 2001.

[31] S. Lindsey and C. S. Raghavendra. PEGASIS: Power Efficient Gathering in Sensor Information Systems. In *Proceedings of IEEE ICC*, 2001.

[32] X. Liu, Q. Huang, and Y. Zhang. Combs, Needles, Haystacks: Balancing Push and Pull for Discovery in Large-Scale Sensor Networks. In *Proceedings of ACM Sensys*, November 2004.

[33] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *Proceedings of ICS*, June 2002.

[34] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks. In *Proceedings of OSDI*, December 2002.

[35] Markov Chains. http://www.atlis.com/services/composition/samples/quark%20 sample%20pages/ winston-1076.ch05.pdf.

[36] B. Nath and D. Niculescu. Routing on a Curve. In *Proceedings of HotNets-I*, October 2002.

[37] N. Nisan and A. Ronen. Algorithmic Mechanism Design. In *Proceedings of ACM STOC*, 1999.

[38] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[39] C. H. Papadimitriou. Algorithms, Games and the Internet. In *Proceedings of ACM STOC*, 2001.

[40] A. Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, third edition, 1991.

[41] G.J. Pottie and W.J. Kaiser. Wireless Integrated Network Sensors. *Communications of the ACM*, 43(5):551–558, May 2000.

[42] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic Routing without Location Information. In *Proceedings of ACM MobiCom*, September 2003.

[43] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu. Data-Centric Storage in Sensornets with GHT, A Geographic Hash Table. *MONET Special Issue on Wireless Sensor Networks*, 8(4):427–442, August 2003.

[44] N. Sadagopan and B. Krishnamachari. Decentralized Utility-Based Design of Sensor Networks. In *Proceedings of WiOpt*, March 2004.

[45] N. Sadagopan and B. Krishnamachari. Maximizing Data Extraction in Energy-Limited Sensor Networks. In *Proceedings of IEEE Infocom*, March 2004.

[46] N. Sadagopan and B. Krishnamachari. Maximizing Data Extraction in Energy-Limited Sensor Networks. *IJDSN*, 1(1):123–147, 2005.

[47] N. Sadagopan, B. Krishnamachari, and A. Helmy. The ACQUIRE Mechanism for Efficient Querying in Sensor Networks. In *Proceedings of SNPA*, May 2003.

[48] N. Sadagopan, B. Krishnamachari, and A. Helmy. Active Query Forwarding in Sensor Networks (ACQUIRE). *AdHoc Networks Journal - Elsevier Science*, 3(1):91–113, January 2005.

[49] N. Sadagopan, M. Singh, and B. Krishnamachari. Decentralized Utility Based Sensor Network Design. *Special issue of ACM/Kluwer MONET*, to appear.

[50] A. Sankar and Z. Liu. Maximum Lifetime Routing in Wireless Ad-Hoc Networks. In *Proceedings of IEEE Infocom*, March 2004.

[51] S. Singh, M. Woo, and C. S. Raghavendra. Power-Aware Routing in Mobile Ad Hoc Networks. *Mobile Computing and Networking*, pages 181–190, 1998.

[52] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin. Habitat Monitoring with Sensor Networks. *Communications of the ACM*, 47(6), June 2004.

[53] The Great Duck Island Project. http://www.greatduckisland.net/index.php.

[54] C. Toh. Maximum Battery Life Routing to Support Ubiquitous Mobile Computing in Wireless Ad Hoc Networks. *IEEE Communications Magazine*, June 2001.

[55] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2003.

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

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

[58] P. Zhang, C. M. Sadler, S. Lyon, and M. Martonosi. Hardware Design Experiences in ZebraNet. In *Proceedings of ACM SenSys*, 2004.

[59] C. Zhou and B. Krishnamachari. Localized Topology Generation Mechanisms for Self-Configuring Sensor Networks. In *Proceedings of IEEE Globecom*, December 2003.