Energy Efficient Algorithms in Low-Energy Wireless Sensor Networks

by

Timothy B. Lewis

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2019

Doctoral Committee:

Professor Quentin F. Stout, Chair
Professor Jerome Lynch
Associate Professor Chris Peikert
Professor Seth Pettie

Timothy B. Lewis

timlewis@umich.edu

ORCID iD: 0000-0002-6448-3641

For my wife Anna,
who is always by my good side.

Acknowledgments

I have so much to be grateful for in my journey throughout graduate school. First and foremost I owe much of my success to my advisor Professor Quentin Stout, whose wisdom and breadth of knowledge were indispensable to my research and growth over the last five years. I would also like to thank my other committee members Professor Jerome Lynch, Professor Chris Peikert, and Professor Seth Pettie for their insight.

I am grateful to have such wonderful members in my research group Yujie An and Amy Nesky, and the other wonderful individuals in my office, to open me up to different ideas and research topics. I appreciate working with such amazing professors and staff, Ilya Volkovich, Grant Schoenebeck, Jeremy Gibson, and many others, throughout my time teaching during the school year. I also appreciate working with the incredible people of Google, including my managers Matt Lang and Mohsen Taheriyan, during my internships with the company.

I feel indebted to those who helped lighten the load of grad school, not through my work but by keeping my spirits high. I have so many friends to thank, from undergrad: Ryan, Josh, Ben, David, and others, from running: Ben, Avi, Sean, and Brad, and from board game nights: Emily, Tim, Arash, Aaron, Shashank, and others.

Finally, I must absolutely thank my parents Bryan and Carol who encouraged me and impressed in me the work ethic I needed to complete this odyssey. And of course, my wife Anna who makes every day a little easier and I am lucky to have by my side.

Table of Contents

List of Figures

List of Tables

Abstract

Wireless sensor networks (WSNs) consist of small autonomous processors spatially distributed, typically with the goal of gathering physical data about the environment such as temperature, air pressure, and sound. WSNs have a wide range of applications including military use, health care monitoring, and environmental sensing. Because sensors are typically battery powered, algorithms for sensor network models should not only seek to minimize runtime but also energy utilization. Specifically, to maximize network lifetime, algorithms must minimize the energy usage of the sensors that use the most energy in the network.

In extremely dense networks it may be inefficient for sensors to communicate with all neighboring sensors on a consistent basis, especially in mobile wireless sensor networks (MWSNs) where the topology of the network is constantly changing. Sensors conserve energy by going into a low-energy sleep state, and in our algorithms sensors will be asleep for the vast majority of the total runtime. Algorithms under these conditions face additional challenges because of the increased difficulty of coordinating between sensors. Because of the spatial nature of sensor networks, geometry problems are often of particular interest. For example, to detect outliers, data is often compared with the nearest neighboring sensors.

In this dissertation we provide algorithmic techniques designed for divide-and-conquer solutions to computational geometry problems. We provide a technique for

coordinating divide-and-conquer algorithms in a single-hop setting called *breadth first recursion*. We use this technique to sort data and to find the convex hull. Although most WSNs are multi-hop networks, locally very dense, expansive networks resemble single-hop networks. Thus we use algorithms for single-hop networks as a building blocks for multi-hop algorithms with *α-consolidation algorithms*. We then provide α-consolidation algorithms for all points k-nearest neighbors, the coverage boundary, and the Voronoi diagram.

We also analyze the WSN problem of propagating data to a high-energy base station. Clustering approaches, such as low-energy adaptive clustering hierarchy (LEACH) and its multi-hop variant (MR-LEACH), are extremely popular for data propagation. The energy balanced protocol (EBP) is a clustering approach like MR-LEACH where clusters pass data towards the base station but also, with some probability, send data long distances directly to the base station. We analytically and empirically show that EBP is close to optimal while approaches that do not use long hops like MR-LEACH are only close to optimal if sending messages long distances is prohibitively expensive.

Chapter I

Introduction

1.1 Overview of Wireless Sensor Networks

Wireless sensor networks (WSNs) are networks containing many small autonomous processors that are spatially distributed, typically with the goal of sampling data from the environment such as temperature, air pressure, and sound. Sensors are equipped with radio transceivers and typically communicate via a small number of broadcast channels. WSNs have use in many diverse applications including ecological studies [20], military intelligence [54], robotics [35], and health care monitoring [41]. Often sensors are placed in remote or even dangerous locations, so maintaining sensor longevity is critical. Longevity is typically determined by battery lifetime, hence energy dissipation is a primary concern [26, 40, 49]. To this end, sensors are often designed to have a low energy sleep state where messages cannot be sent or received but energy consumption is drastically cut [14].

Data gathered by sensors is often sent, either directly or through intermediate sensors, to a high-power base station which acts as a gateway between sensor nodes and the end user [4, 5]. Clustering approaches, such as low-energy adaptive clustering

hierarchy (LEACH) [22] and its multi-hop variant (MR-LEACH) [17], are extremely popular for data propagation. The energy balanced protocol (EBP) [15] is a clustering approach like MR-LEACH where clusters pass data towards the base station but also, with some probability, send data long distances directly to the base station.

In Chapter IV we analyze the exact cost of optimal data propagation with the goal of extending network lifetime. We show that EBP is close to optimal and that other approaches that use only short hops such as MR-LEACH are only close to optimal if transmitting long distances is prohibitively expensive. In large expansive networks, transmitting data to a powerful base station may be too expensive in terms of energy, as our analysis in Chapter IV shows, and in other cases such base stations may not even exist [45]. In these situations sensors do computations to analyze data and manage the network in a distributed manner on the network itself.

Creating distributed algorithms on wireless sensor networks presents additional challenges because of the added energy constraints. In particular, coordinating processors is much more difficult because sensors are asleep for the vast majority of the algorithm's runtime. In Chapter II, we present *breadth first recursion* as a framework for executing recursive algorithms where the runtime of each recursive call is random or data dependent in single-hop networks. Although single-hop networks are rare in practice we use single-hop algorithms as building blocks for creating multi-hop algorithms in Chapter III, where we present *consolidation algorithms*. In consolidation algorithms sensors are involved in multiple overlapping single-hop regions where they compute a local solution to the problem. Then once all the local solutions are generated these solutions are consolidated into a single global solution. Both breadth first recursion and consolidation algorithms are specifically designed to assist in divide-and-conquer algorithms. Divide and conquer algorithms are specifically relevant in wireless sensor networks because many computational geometry problems can typi-

cally be solved using divide-and-conquer [44].

Due to the spatial nature of the way sensors are distributed, geometric problems are often relevant in WSNs. For example, sensors may compare data with nearby neighboring sensors to determine outliers [33]. As another example, in [53] sensors are used to monitor the boundary of some environmental contaminant. Also in [1] the Voronoi diagram is used to optimize deployment of sensors to maximize the coverage of an area of interest. We provide fast, energy-efficient solutions for these computational geometry solutions in Chapters II and III.

The breadth of problems and areas of study relevant to WSNs is particularly expansive. Localization, which is the process of determining sensor position relative to neighboring sensors, is important because sensors are often not equipped with expensive GPS technology [36]. Clock synchronization and managing clock drift is critical for managing sleep wake-up schedules [52]. Clustering sensors helps conserve energy by combining data into larger packets [8, 19]. Radio interference detection [60] and medium access control protocols [57] are often used to manage the radio broadcast channels. Because adversaries may have easy access to the radio channel being used or even the sensor devices themselves, security is particularly important in WSNs, especially in military settings [11]. Because sensors are small and low cost, hardware failures are inevitable and for this reason algorithms for WSNs are often designed to be fault tolerant in the case that one or multiple sensors go down [2]. While we consider these problems when creating our model and when discussing future work, they are beyond the scope of the main work of this dissertation.

1.2 Model Assumptions and Justifications

Throughout this dissertation we primarily use two major WSN models, a broadcast model and a message-passing model, and consider different variants of both models. The broadcast model is used in Chapters II and III and the message-passing model is used in Chapter IV. The primary difference between the two is that the message-passing model assumes use of a medium access control (MAC) protocol which allows for direct communication between sensors, and the broadcast model does not. Instead, in the broadcast model transmissions are sent to all sensors within range and may possibly interfere with other simultaneous broadcasting sensors.

There are two major categories of MAC protocols, contention-based and schedule-based. In contention-based MAC protocols there are contention windows where sensors use backoff to vie for each transmission slot. On the other hand, schedule-based MAC protocols where each transmission slot is assigned for a message to be sent at regular intervals. In Chapter IV we analyze different protocols for propagating data generated by sensors to a high-energy base station. In the data propagation protocols we analyze each sensor only sends messages to a small number of other sensors and these messages are repeated each time data is generated. Thus it is practical to use a schedule-based MAC protocol in this scenario.

In Chapters II and III we analyze distributed algorithms on very large, dense WSNs such as Smart Dust [28]. In such networks, transmitting data to a powerful base station may be too expensive in terms of energy, and in other cases such base stations may not even exist [45]. In distributed algorithms, communication patterns are far more unpredictible in terms of which sensors interact. Additionally, interacting with all neighboring sensors may be far too expensive in terms of energy and if the network topology is changing over time this further complicates communication

protocols. This means that establishing MAC protocols will be much less practical in this scenario. Furthermore, MAC protocols typically do not fully use the broadcast capability of the radio channels. That is, sensors are not able to broadcast to small varying subnetworks. In Chapters II and III we utilize partial broadcasts to subnetworks and thus our broadcast model does not use a MAC protocol.

The following is a more formal list of assumptions our models make and justifications for these assumptions. Additionally, Table 1.1 provides quick reference comparing the broadcast and message-passing models.

Homogeneous network: All sensors have the same battery life and are identically configured except for a $O(\log n)$ bit unique ID. The sensors' IDs are not necessarily $1, \ldots, n$ and the sensors may not even know n. These IDs are primarily only used for identification but in Subsection 3.2.2 we use the unique IDs to help resolve the hidden node problem.

For each sensor there is a point in $\mathbb{R}^2$ representing that sensor's location which may or may not be known by the sensor. There has been some work with sensors in three-dimensional space [27], however applications where the third dimension is relevant are uncommon and algorithms for two dimensions are much simpler [7].

In the message-passing model there is a powerful base station which is only used as a sink for sensors to send data to. In the broadcast model there are no powerful base stations in the network.

Low-energy sleep state: In terms of the energy needed per operation, broadcasting is the most energy intensive, receiving is the second-most intensive, and calculations are third. To limit energy usage, sensors often have a very low energy sleep state with no communication and limited processing. For instance the ZigBee microcontroller uses 9,300 μA of current while active, and only 4.18 μA while in deep sleep [14]. We do not assume that sleeping sensors can be woken up by other sensors

on command. Thus a sensor must determine the duration that it will sleep before entering the sleep state, and it stays asleep for that entire duration.

By the *energy dissipation* of a sensor we mean the energy in excess of that of the sleeping state and assume this is proportional to the time the sensor is awake. Particularly we care about minimizing the maximum energy dissipated by any sensor, since the sensor using the most energy drains its battery the quickest.

Broadcast vs. message passing: Sensors are typically equipped with a radio transceiver that can send and receive broadcasts among a small number of radio channels, and typically only a small number of channels are used to reduce radio spectrum pollution. Our results generalize for some small number of broadcast channels but for simplicity we assume only one channel is available.

Often sensors coordinate usage of the broadcast channel using a medium access control (MAC) protocol [57]. There are two main categories of MAC protocols: contention based and schedule based. In contention-base MAC protocols there are contention windows where sensors use backoff to vie for transmission slots. On the other hand, in schedule-based MAC protocols each transmission slot is assigned to be used by specific sensors at regular, predetermined intervals. While MAC protocols do usually allow sensors to broadcast to all other sensors they usually do not fully utilize the broadcast capability of the radio transceivers. That is, they do not allow sensors to broadcast to varying small subnetworks while the rest of the network sleeps.

The primary difference between our two models is the usage of a MAC protocol. The broadcast model does not use a MAC protocol and allows for broadcasts to all nearby listening sensors. The message-passing model does use a MAC protocol and assumes sensors can send messages to other individual sensors.

We assume that a transceiver can send and receive only small $O(\log n)$ sized packets in unit time. That is, the "word size" of our models are $O(\log n)$ to allow IDs

to be sent in constant time.

Radio interference detection: For the message-passing model the MAC protocol already ensures that no messages will interfere, so radio interference is only relevant to the broadcast model. If two or more sensors within broadcast range of a listening sensor broadcast at the same time then these transmissions will interfere with each other and the listening sensor will hear neither message. This is often called a *collision*. Collision detection is an important area of study and we assume some method such as in [60] is being used. While listening a sensor may either receive no broadcast, a single broadcast, or interference, and can distinguish between the three.

Sensors can broadcast and listen at the same time, meaning a broadcasting sensor can detect if some other sensor within broadcast distance is also broadcasting simultaneously. It cannot, however, read this message because of the interference. In a multi-hop network it is also possible for there to be interference for some sensors within broadcast range but for the broadcasting sensor not to hear interference. If a sensor just further than r_c away broadcasts simultaneously then the broadcasting sensors will not hear interference but the sensors in between will hear interference.

Fixed vs. variable transmission distance: Radio transceivers are typically able to transmit at varying broadcast strengths. For the broadcast model we assume that this broadcast strength is fixed and thus the communication radius r_c is also fixed.

For the message-passing model we consider the data propagation problem and in this problem deciding how far to send the data each sensor has is the main focus of the problem. Thus in the message-passing model we allow transmission distance to be variable, where the energy cost $E(d)$ for transmitting unit-sized data d distance is

$$E(d) = e_0 + e_t d^\alpha$$

where e_0 is the cost to run the transmitter circuitry, e_t is the coefficient of energy cost due to transmission amplification, and α is the path loss exponent. We assume that e_0 and e_t are positive constants and α is a constant between 2 and 4. In practice e_0 is about 50 nJ/bit and e_t is about 100 pJ/bit/m$^\alpha$ [22]. In an ideal environment with a perfectly clear radio channel we have $\alpha = 2$, but due to multi-path loss α is higher in realistic scenarios [39].

Notably, in the message-passing model we ignore the energy for receiving messages because, as will be discussed in Chapter IV, the vast majority of data a sensor sends in the data propagation setting has been received from another sensor. Thus to account for energy spent receiving data one simply needs to add this energy cost to the e_0 term.

Single-hop vs. multi-hop network: In the broadcast model with fixed broadcast radius we will use both a single-hop model in Chapter II and a multi-hop model in Chapter III. In the single-hop model every sensor is within broadcast range of every other sensor. In the multi-hop model two sensors are within broadcast range of each other if the Euclidean distance between the two is at most the communication radius r_c.

Although single-hop networks are unrealistic in practice, the main technique we present in Section 3.2 is designed for divide-and-conquer algorithms, which typically require local solutions to be solved before merging the solutions together [36]. When a problem is divided into sufficiently small subproblems a large multi-hop network locally resembles a single-hop network and thus algorithms for single-hop networks can be building blocks for algorithms for a multi-hop network. Because of the spatial nature of sensor networks, geometric problems often arise very naturally and these problems are often solved using divide-and-conquer algorithms. These algorithms will be most useful for extremely dense sensor networks like Smartdust [28] where the

local networks may have hundreds or even thousands of nodes.

Global coordinate system vs. no global coordinate system: In some networks sensors are equipped with GPS devices which allows them to know their exact position in $\mathbb{R}^2$ [52]. These networks are said to have a known *global coordinate system*. A global coordinate system can be used as a method of coordination between sensors where sensors know at certain times only sensors with specific GPS coordinates will be allowed to broadcast.

However, GPS can be quite expensive for these small, power-sensitive devices and thus many sensor networks are not equipped with GPS. If GPS coordinates are not available sensors may need to create local coordinate systems. In these situations each sensor can determine its position relative to other neighboring sensors using techniques such as triangulation [12]. While the relative positions of sensors are consistent in these local coordinate systems, they may be subject to rotation and translation. That is, distance and angle information between sensors are preserved in all local coordinate systems but the exact coordinates of each sensor may vary wildly in different local coordinate systems.

Throughout most of this dissertation the results will hold for both networks with and without a known global coordinate system. However this distinction will be important for Section 3.2 because global coordinate systems can be used for coordinating radio channel usage.

Synchronized clocks: Synchronizing clocks is a fundamental problem in distributed systems, especially in wireless sensor networks. Many techniques are discussed in [52]. We assume that sensors have already synchronized their clocks by some method and that this procedure will account for any drift between clocks.

Synchronization will be an important assumption for the broadcast model that allows sensors to schedule sleep-wake cycles and to coordinate radio broadcast channel

	Broadcast model (Ch. II & III)	Message-passing model (Ch. IV)
Homogeneous network	X	X
High-power base station		X
Unique ID	X	X
Positions in $\mathbb{R}^2$	X	X
Synchronized clocks	X	X
Radio interference detection	X	X
MAC protocol		X
Variable transmission strength		X

Table 1.1: Comparison between the broadcast model and the message-passing model.

usage. We will not explicitly use this assumption in the message-passing model but it is used implicitly because MAC protocols typically require clock synchronization.

1.3 Organization of Dissertation

The remainder of this dissertation is structured as follows. First Chapter II examines single-hop networks and introduces *breadth first recursion*, a framework for overcoming problems that arise when creating randomized recursive algorithms. We use this technique to create algorithms for sorting and computing the convex hull and we run simulations showing the time and energy usage of these algorithms. Then Chapter III introduces *consolidation algorithms* as a framework for using single-hop algorithms as building blocks to create multi-hop algorithms. We use this technique to compute all points k-nearest neighbors, the coverage boundary, and the localized Voronoi diagram. Then Chapter IV provides analysis of the exact cost of optimally propagating data from sensors to a high-power base station. This motivates the previous chapters by comparing the cost of performing calculations in a distributed manner versus collecting data and performing calculations at the high-power base station. Finally, Chapter V provides a detailed conclusion and future work related to

this dissertation.

Chapter II

Single-Hop Algorithms

2.1 Introduction

In this chapter we discuss single-hop wireless sensor networks and overcome challenges when creating distributed algorithms for these networks. In a single-hop network every processor is within transmission range of every other processor. While this is unrealistic in practice we will use these single-hop algorithms as building blocks for multi-hop algorithms in Chapter III. Our primary goals are not only to minimize runtime and total energy usage but also to minimize the maximum energy used by any sensor, ensuring that the network lifetime is maximized. To conserve energy sensors will go into a low energy sleep state for the vast majority of the runtime of our algorithms.

Coordinating processors can be significantly more difficult in settings where each processor is inactive for the vast majority of the time. Specifically, hibernating processors may not be certain of what happened while they were asleep. For many simple algorithms this is straightforward; one can easily sum n numbers among n ordered processors in $O(n)$ time with each processor being awake for $O(1)$ time. However,

even something as simple as stepwise simulation of serially merging two lists is more involved in this setting because it is not known when a given value will be needed to merge [30]. Having all processors awake until their value is merged will take optimal $O(n)$ time, but each processor will use $O(n)$ energy.

Randomization is a useful approach for minimizing expected time and for load balancing [37]. However, if a processor is sleeping during a task with a random completion time then it may require waking many times to determine when the task is finished. In Section 2.2 we discuss *task completion testing*, a simple method for executing randomized algorithms in this setting. It is yet more complex for recursive algorithms where the runtime is an accumulation of the runtime of subproblems with random completion times where the processors working on the subproblems must themselves determine when to wake for their subtasks. To execute recursive algorithms in this setting we introduce *breadth first recursion* in Section 2.4. This process uses elected leaders to coordinate recursion one entire recursive level at a time. We discuss different election protocols used throughout this chapter in Section 2.3.

We then use breadth first recursion to create an algorithm for sorting based on quick sort in Section 2.5. We show that this algorithm is faster than previous results both in theory and through simulation, and that the energy usage is similar to previous results. We also use breadth first recursion to create an algorithm for finding the convex hull in Section 2.6. This algorithm again is energy efficient and its runtime is output sensitive, meaning that the runtime is dependent upon the size of the convex hull. For many realistic distributions of sensors this algorithm's runtime is sublinear. Additionally, we provide simulations showing the performance of this convex hull algorithm on different sensor distributions.

2.2 Task Completion Testing

Since sensors are asleep for the vast majority of the time, if the runtime of a task is a random variable then a coordination strategy must be used to determine when the task actually completes. *Task completion testing* is a simple concept which allows the sensors in a single-hop network to coordinate the completion of a task with, in expectation, only a constant factor overhead in the runtime and an additive constant overhead in energy for any sensor.

The execution of the task is interrupted at predetermined intervals proportional to the expectated runtime of the task. During an interruption, if the task is not completed then every sensor that knows the task is not complete will broadcast an announcement. Each sensor waiting for the algorithm to complete will wake up at these intervals and listen for an announcement or for interference from multiple sensors broadcasting an announcement. If it hears this then it goes back to sleep until the next scheduled interrupt.

Lemma 1. *In a single-hop network if a task with random completion time T is interrupted at intervals τ then the expected number of interruptions is at most $\mathbb{E}[T]/\tau + 1$.*

Proof. Let I be the number of interruptions. Then

$$\begin{aligned}
\mathbb{E}[I] &= \sum_{i=0}^{\infty} P(T = i) \left\lceil \frac{i}{\tau} \right\rceil \\
&\leq \sum_{i=0}^{\infty} P(T = i) \left(\frac{i}{\tau} + 1 \right) \\
&= \frac{\mathbb{E}[T]}{\tau} + 1
\end{aligned}$$

□

Thus the total time including interruptions is at most $(\tau + a) \cdot (\mathbb{E}[T]/\tau + 1)$, where a is the time taken for an interruption. Notice that if $\tau = c \cdot \mathbb{E}[T]$ then the expected number of interrupts is at most $\frac{1}{c} + 1$ and the total expected time is at most $\mathbb{E}[T](1 + c) + a(\frac{1}{c} + 1)$.

2.3 Leader Election

Our algorithms will require groups of sensors to coordinate usage of the communication channel. To assist with this, the groups elect a leader to make announcements for the group. In general a sensor knows if it is in the group, but no sensor knows which, or how many, other sensors are in the group.

Leader election is well studied in the sensor network setting, and for a specific application more appropriate leader election techniques may be substituted. We will provide two algorithms for electing leaders in a single-hop sensor network. The first algorithm (BackoffElect) uses a simple backoff mechanism in which sensors drop out of the election until only one sensor is left. In the second algorithm (EstimateElect) the group of sensors use an estimate of the total number in the group. Using this estimate each sensor then selects itself with some probability and the estimate is improved until a single sensor has been selected.

BackoffElect will be used primarily throughout this chapter. It uses in expectation $O(\log n)$ time, $O(1)$ average energy per sensor, and $O(\log n)$ maximum energy by any sensor. This algorithm has better average energy usage so it will be used throughout when there is no estimate on the number of sensors in the set or the faster second election procedure is not needed.

To perform the election each sensor flips a fair coin and broadcasts its ID if the coin lands heads. If only a single sensor announces its ID then that sensor is now

```
BackoffElect(S) =
 1: while a leader has not been elected
 2:    each sensor that has not dropped out
       broadcasts its ID with probability 1/2
 3:    if exactly one sensor broadcasted its ID then
 4:       that sensor is the leader
 5:    else if multiple sensors broadcasted their IDs
 6:       all sensors that did not broadcast drop out
 7:    else if no sensor broadcasted its ID
 8:       no sensors drop out
 9:    end if
10: end while
```

Figure 2.1: Algorithm for electing a leader from a set S of sensors.

the leader. If more than one sensor broadcasts then the sensors that flipped a tail drop out of the election and the remaining sensors repeat this process. If no sensor broadcasts then no sensors drop out of the election and the process continues. The exact algorithm is given in Figure 2.1.

Lemma 2. *The expected runtime of BackoffElect in a single-hop network of n sensors is $O(\log n)$ with an exponentially small tail.*

Proof. Notice that for a single sensor s, if any other sensors broadcast and s does not then s drops out of the election, and if no other sensors broadcast and s does then s is elected. So each coin flip has exactly a $\frac{1}{2}$ probability of being the last coin flip a sensor makes. Then the expected number of coin flips a single sensor will make is 2 and forms a geometric distribution. Thus Lemma 36 in the Appendix completes the proof. □

If E elections are performed with a set of n sensors then the expected energy per sensor is $O(E \log n)$. This can be improved if the number of sensors in the election is known or at least an estimate is known. We do so by allowing sensors that drop out

of the election to go to sleep and use task completion testing. Then the total time remains within a constant factor of the actual election time and the energy a sensor uses is proportional to the number of coin flips that it makes which is shown to be $O(E + \log n)$ in the following lemma.

Lemma 3. *If a single-hop network of n sensors is involved in E elections using BackoffElect, then the expected maximum number of coin flips any sensor makes is $O(E + \log n)$ with an exponentially small tail.*

Proof. Since each coin flip that a sensor makes has exactly a $\frac{1}{2}$ probability of being its last of an election the distribution of the number of coin flips for a single sensor is a negative binomial distribution with expectation $2E$. Thus Lemma 36 in the Appendix completes the proof. □

EstimateElect is only used in Section 2.6 when we have an approximation of the number of sensors involved in the election. The runtime of this algorithm is dependent upon the accuracy of our estimation, m, and every sensor must be awake for the entire election.

This algorithm is similar to the algorithm used for maximum finding as described in [38]. We begin by having each sensor decide to announce itself as leader with probability $\frac{1}{m}$. Then if no sensor broadcasts, m is lowered, and if multiple sensors broadcast, m is raised. This process is repeated until a single sensor elects itself. The exact algorithm is given in Figure 2.2.

Lemma 4. *The expected runtime of EstimateElect on a single-hop network of n sensors with estimate m is $O(|\log \frac{n}{m}| + 1)$ with an exponentially small tail.*

Proof. Throughout we will assume $n \geq 3$. Also note that no attempts are made to optimize the constants.

```
EstimateElect(S, m) =
 1: while a leader has not been elected
 2:    each sensor broadcasts its ID with probability 1/m
 3:    if exactly one sensor broadcasted its ID then
 4:       that sensor is the leader
 5:    else if multiple sensors broadcasted their IDs
 6:       m = 2m
 7:    else if no sensor broadcasted its ID
 8:       m = max(m/2, 1)
 9:    end if
10: end while
```

Figure 2.2: Electing a leader from a set S of sensors with estimate m of $|S|$.

We will say that m is a good estimate if $\frac{n}{2} < m < 2n$. Suppose m is a good estimate; then the probability a sensor is elected in a single step is

$$n \cdot \frac{1}{m} \cdot \left(1 - \frac{1}{m}\right)^{n-1} \geq n \cdot \frac{1}{2n} \cdot \left(1 - \frac{2}{n}\right)^{n-1} \geq \frac{1}{18}$$

If m is initially a bad estimate, we will say that m is improved during a time step if after the time step m is closer to the "good" interval $\left(\frac{n}{2}, 2n\right)$. Suppose $m < \frac{n}{2}$; then we will look at the probability that m improves or that a sensor is elected. This is the same as the probability that at least one sensor announces itself as leader. The probability that at least one sensor announces is

$$1 - \left(1 - \frac{1}{m}\right)^{n} \geq 1 - \left(1 - \frac{2}{n}\right)^{n} \geq 1 - e^{-2} \approx 0.86$$

Suppose $m > 2n$; then the probability that m improves is the probability that no sensor announces, which is

$$\left(1 - \frac{1}{m}\right)^{n} \geq \left(1 - \frac{1}{2n}\right)^{n} \geq \frac{9}{16}$$

Let $d = \left\lfloor |\log_2 \frac{n}{m}| \right\rfloor$, which is the minimum number of steps before $\frac{n}{2} \leq m \leq 2n$. Let's assume that every time step where m is a good estimate and a sensor is not elected that m falls out of range of being a good estimate. Let k be the number of time steps where m is a good estimate, let s be the number of time steps where m is not a good estimate but m improves or a leader is elected, and let f be the number of steps where m is not a good estimate and does not improve. Notice if $s - f \geq k + d$ then a sensor must have been elected and the runtime T is bounded by $s + f + k$. So we can use the following loose upper bound

$$P(T = t) \leq P\left(k \geq \frac{t}{100}\right) + P\left(s - f < k + d \;\middle|\; k < \frac{t}{100}\right)$$

We know $P\left(k \geq \frac{t}{100}\right) \leq \left(\frac{1}{18}\right)^{\frac{t}{100}-1}$ and given $k < \frac{t}{100}$, $P(s - f < k + d) \leq P\left(f < \frac{d+t}{2}\right)$. Then, using the Hoeffding/Chernoff bound with $\epsilon = \frac{9}{16} - \frac{\frac{d}{t}+1}{1.98}$, we get

$$P\left(s < \left(\frac{9}{16} - \epsilon\right)(s + f)\right) \leq e^{-2\epsilon^2(s+f)} \leq \left(e^{-1.98\epsilon^2}\right)^t$$

If $t \geq 100d$ then $\epsilon \geq 0.05$. Therefore when $t = \Omega(|\log \frac{n}{m}|)$, $P(T = t)$ is bounded by the sum of two negative exponentials which is itself a negative exponential. This completes the proof. □

2.4 Breadth-First Recursion

2.4.1 Overview

Task completion testing is effective for coordinating any number of consecutive tasks in a single-hop network. However this procedure is not effective if used for tasks that are nested recursively. This is because the total runtime is slowed by a constant

factor c for each recursive level, leading to a c^ℓ factor slowdown if ℓ is the number of recursive levels. In this section we provide a technique that resolves this issue for recursive algorithms in a single-hop network by completing the tasks in a consecutive manner instead. We call this technique *breadth-first recursion*.

Figure 2.3 gives the general form of a recursive algorithm for which this technique applies. We require that the two recursive calls are independent, which is to say that the results of one recursive call are not required to complete the other. However the recursive calls are allowed to depend upon the results of A, the procedure executed before the recursive calls. Similarly B, the procedure executed after the recursive calls, can depend upon the results of either the recursive calls or A.

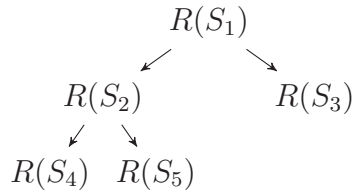
Notice that if the algorithm is executed as written in Figure 2.3 then the tasks A and B will be executed in a depth-first manner. As the name would suggest, during breadth-first recursion these routines will instead be executed in a breadth-first manner. We complete the A tasks top to bottom, one recursive layer at a time. Once all layers of A tasks are complete we execute the B tasks similarly but from bottom to top. The reason for this breadth first execution is that it allows us to eliminate the recursive nature of the dependencies between tasks and instead treat them like consecutive tasks. Figure 2.4 gives an example recursion tree and lists the order of execution for tasks A and B for both breadth-first and depth-first.

2.4.2 Execution of a Single Recursive Layer

From this point forward, we will refer to a set of sensors S_i involved in the execution of a task $R(S_i)$ as a *group* of sensors. Similarly we will refer to a subset S_j of S_i involved in the execution of a task $R(S_j)$ as a *subgroup* of S_i. If $R(S_j)$ is recursively called in $R(S_i)$ then we will also say that S_j is a *child* of S_i and S_i is the *parent* of

$R(S) =$
1: $A(S)$
2: $R(S_1)$ $\{S_1 \subseteq S\}$
3: $R(S_2)$ $\{S_2 \subseteq S$ and $S_1 \cap S_2 = \emptyset\}$
4: $B(S)$

Figure 2.3: General form of a recursive algorithm R which utilizes routines A and B.



Depth-first	Breadth-first
$A(S_1)$	$A(S_1)$
$A(S_2)$	$A(S_2)$
$A(S_4)$	$A(S_3)$
$B(S_4)$	$A(S_4)$
$A(S_5)$	$A(S_5)$
$B(S_5)$	$B(S_5)$
$B(S_2)$	$B(S_4)$
$A(S_3)$	$B(S_3)$
$B(S_3)$	$B(S_2)$
$B(S_1)$	$B(S_1)$

Figure 2.4: Example recursion tree and task execution order.

S_j. If a group S_i has a leader we will denote that leader l_i.

To complete the tasks in a recursive layer we alternate between coordination phases and work phases. During a work phase all groups that have not completed task A are allocated a portion of time to use the broadcast channel and work on their task. During a coordination phase the groups determine how many groups are not yet complete, how much time is needed for the next work phase, and what portion of that work phase each group is allocated. The amount of time allocated to a group during a work phase is a constant fraction of the total amount of time used by the group's parent to complete routine A. The execution of each layer consists of multiple coordination and work phases. This entire task of completing a layer is executed using task completion testing so that once a group has completed task A it can ignore the

```
1: initial coordination phase
2: begin task completion testing
3:    while some group is not done do
4:       work phase
5:       coordination phase
6:    end while
7: end task completion testing
```

Figure 2.5: Overview of the execution of a single layer of recursion.

remaining coordination phases, which in expectation will be logarithmic in the number of groups. Figure 2.5 gives an outline of how a layer of the recursion tree is executed. Figure 2.6 shows an example recursion tree with two layers completed and Figure 2.7 provides an example timeline of the completion of the third layer of that tree.

2.4.3 Coordination Phase

As a broad overview, each coordination phase is essentially a serial scan of the number of groups still in the layer and the total time all the groups will take.

To assist in the coordination phase each group elects a leader so that only one sensor attempts to broadcast at a time. If the A task does not already include an election we simply add the election of a leader to the beginning of A. We will show that these elections do not add substantially to the overall runtime or energy usage of the algorithm. Since leader election takes place during the work phase, a leader for a group may not have been elected by the first few coordination phases. To handle this case, the leader of the parent of the group will listen to the election and act as the leader for the group if the election is not complete.

Each layer begins with an initial coordination phase where each group from left to right in the previous layer wakes up. First every sensor in the left subgroup broadcasts so that the sensors know whether or not the left subgroup is empty, and then the right

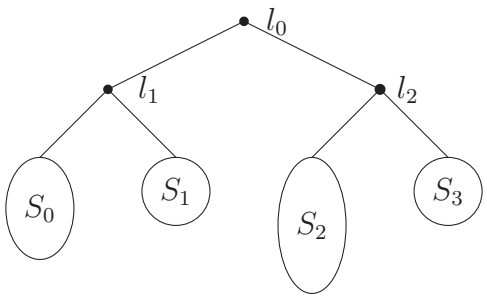
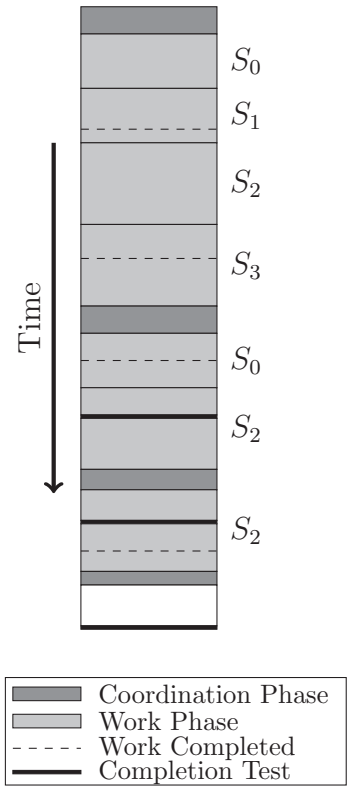


Figure 2.6: Example recursion tree with two layers completed.



Coordination Phase 1	
Broadcasts	Listens
S_0	l_1, S_1
S_1	l_1
l_1	l_2, S_2, S_3
S_2	l_2, S_3
S_3	l_2
l_2	all sensors

Coordination Phase 2	
Broadcasts	Listens
S_0	l_1
	l_1
l_1	l_2, S_2
S_2	l_2
	l_2
l_2	l_1, S_0, S_2

Coordination Phase 3	
Broadcasts	Listens
	l_1
l_1	l_2, S_2
S_2	l_2
l_2	S_2

Figure 2.7: Sample timeline for completing the third layer of the recursion tree in Figure 2.6.

subgroup follows suit. Then the leader announces the total number of subgroups so far in the current layer and the total amount of time all groups will use in the next work phase. The next group listens to this announcement so that they know when to wake up during the next coordination and work phases. All sensors wake up and listen to the last leader's announcements so they know how long the next work and coordination phases will last. The time announced here is the estimate of the runtime of the layer used in task completion testing.

All remaining coordination phases are executed similarly. From left to right each group in the current layer that was allocated time in the last work phase wakes up and all sensors that know that the task is not completed broadcast. Then the leader announces the total number of groups so far that are not complete and the total amount of time these groups will use in the next work phase. Each group listens to the previous group's announcement so that every sensor knows the group's allocated time in the work phase and when to wake up in the next coordination phase. Every sensor that is in a group that is not complete will listen to the last leader's announcements so that it knows how long the next work and coordination phases will last.

2.4.4 Work Phase

During a work phases each group of sensors wakes up at the beginning of the time frame allocated to them during the last coordination phase and begins work on the A task (which will begin with an election). They then go to sleep when this time frame ends even if they haven't completed the task. Notice that this phase does not explicitly use task completion testing but a similar process is occurring. The task is interrupted at intervals proportional to the expected runtime and during the interruptions the sensors determine if the task is complete, albeit after a long wait

period. So the analysis of the time and energy overhead for task completion testing holds here as well.

2.4.5 Traversing Up the Tree

The procedure for completing the B tasks is almost identical, with the only differences being that it works from the bottom up, the initial coordination phase is not a special case from the rest of the coordination phases, and the amount of time given to each group for work phases is proportional to the expected time for the B task (not the A task). The initial coordination phase is not needed because each sensor can just remember its group's place for each layer. In an extremely low memory setting instead we can just have each leader remember the position when it was leader and then use a pre-coordination phase to remind each sensor of its position in the next layer. In many cases at this point we can actually keep track of the exact number of sensors in the subtree so the work estimation can be more precise. In the case where B is not randomized these tasks can instead simply be completed in a single reverse breadth-first traversal of the tree.

2.4.6 A Note on Completion Testing

Before analyzing the time and energy usage of our procedure it is worth pointing out that when we use task completion testing, the estimate of the runtime is actually a random variable and is not precisely the expectation of the runtime. However, the time and energy analysis in Section 2.2 will still hold if the estimate E has the properties that $\mathbb{E}[E] = O(\mathbb{E}[T])$ and $\mathbb{E}\left[\frac{1}{E}\right] = O\left(\frac{1}{\mathbb{E}[T]}\right)$. We call a random variable that has these properties a *good* estimate of T.

An easy way to obtain an estimate for the runtime of a task A (resp. B) is to use

the parent's (resp. childrens') runtime. This will be a good estimate if the probability that the task finishes well below the expectation is low and the size of each child is expected to be on within a constant factor of the size of the parent. We call such a task whose runtime is a good estimate of the child's runtime *nice*. Notice that if a task has the property that there exist constants $k > 0$ and $c > 1$ such that $P(T = k\mathbb{E}[T] - j) \leq (1/c)^j$ then it is a nice task. This implies that electing a leader is a nice task. Although not explicitly proven the tasks used for the algorithms presented here have this nice property.

2.4.7 Analysis

Let R be the execution of the recursive algorithm on a single-hop network if each sensor knew, via some oracle, the time that each subtask will be complete and let R' be the execution of the recursive algorithm using breadth first recursion. Also let $T_X(S)$, $E_X(S)$, $M_X(S)$ be the expected runtime, energy dissipation, and maximum energy dissipation, respectively, of a task X over the set of sensors S. Let H be the height of the recursion tree and $S_1, \ldots, S_m$ be the subsets of S that appears in the recursion tree of $R(S)$.

Theorem 5.

$$T_{R'}(S) = O\left(T_R(S) + \sum_{i=1}^{m} \log |S_i|\right)$$

$$E_{R'}(S) = E_R(S) + O(H)$$

$$M_{R'}(S) = M_R(S) + O(H + \log |S|)$$

Proof. First notice that because of our strategic use of task completion testing the total time of R' is proportional to the runtime of R plus the overhead of the coordination phases and electing leaders. However the time spent on coordination phases

is less than the time spent during the work phases, so this time is negligible for the purposes of our analysis. Also notice that, as previously discussed, the expected time spent electing a leader for a group of n sensors is $O(\log n)$. Therefore we achieve the desired total runtime for R'.

Next, the expected energy dissipation per sensor during R' is the sum of the expected energy dissipation during R, during elections, and during coordination phases. The expected energy per election is proportional to the average number of coin flips sensors make which is constant. Since each sensor is involved in at most H elections the expected energy dissipation during elections is $O(H)$. Next by Lemma 1 the expected number of coordination phases each sensor participates in is constant per layer. Then, since each coordination phase takes constant energy per sensor, this energy is also $O(H)$ in expectation. Therefore we achieve the desired average energy dissipation.

Finally, the expected maximum energy dissipated for a sensor during R' is bounded by the sum of the maximum energy dissipated for a single sensor during R, during the elections, and during coordination phases. By Lemma 3 the maximum energy spent for a single sensor on elections is $O(H + \log |S|)$. At each recursive layer a sensor is expected to be involved in a constant number of coordination phases. However if the number of groups is $O(|S|)$ then the expected number of coordination phases is $O(\log |S|)$. But by a similar proof as Lemma 36 in the Appendix the expectation of the maximum number of coordination phases any sensor is involved in is $O(H + \log |S|)$. Therefore we achieve the desired maximum energy dissipation. □

2.5 Sorting

2.5.1 Sorting Algorithm

Sorting is one of the most fundamental computational building blocks and is used in many important problems in a plethora of areas. An algorithm for sorting n values on n sensors is provided in [49]. It requires only $O(\log n)$ energy per sensor, but takes $O(n \log n)$ time and hence is not optimal. A more practical algorithm was presented in [30] but it still requires $O(n \log n)$ time. Because of the broadcast capabilities of sensor networks sorting can actually be done in $O(n)$ time. Since at least $n-1$ sensors must announce their values, $O(n)$ is time optimal. A simple algorithm that achieves this bound is just to have each sensor one by one announce their value with every other sensor listening and determining the rank of its element. This of course is not energy efficient as each sensor is awake for $O(n)$ time. A simple work analysis shows that sorting requires $\Omega(\log n)$ average energy per processor. We now provide a randomized sorting algorithm that meets both these bounds and is energy balanced. We will prove:

Theorem 6. *Given a single-hop network S of n sensors with one data element each, the elements can be sorted in expected time $O(n)$ and expected maximum energy dissipation of any sensor $O(\log n)$.*

In this context sorting means that each sensor in the network determines its rank among all sensors in the network. In Figure 2.8 we give a rank-finding algorithm based on a quicksort approach. It is of the same form as Figure 2.3 and hence can executed using breadth-first recursion.

Notice that once the election is complete the time of the A and B tasks are constant. This implies that the total runtime is expected to be $O\left(\sum_{S_i} \log |S_i|\right)$ where

FindRanks(S) =
1: ElectLeader(S)
2: Leader broadcasts its ID and value
3: FindRanks($S_<$) {sensors with value less than the leader's}
4: FindRanks($S_>$) {sensors with vlaue greater than the leader's}
5: Leader of $S_<$ broadcasts $|S_<|$
6: Leader of $S_>$ broadcasts $|S_>|$

Figure 2.8: Recursive rank-finding algorithm.

each S_i is a subset of sensors that appears in the recursion tree. This runtime is similar to the runtime of constructing a binary heap which is $O(n)$, except in this case our tree is not perfectly balanced. Despite the imbalance in our recursion tree the expected runtime is still $O(n)$ and we prove this using the substitution method. Suppose there exist constants a and b such that for all $m < n$ the expected runtime $T(m) \leq am - b\log m$. Then if the constants a and b are sufficiently large,

$$\begin{aligned}
T(n) &= \frac{1}{n}\sum_{i=1}^{n}\left(T(i-1)+T(n-i)\right)+O(\log n)\\
&= \frac{2}{n}\sum_{i=0}^{n-1}T(i)+O(\log n)\\
&\leq \frac{2}{n}\sum_{i=0}^{n-1}ai-\frac{2}{n}\sum_{i=0}^{n-1}b\log i+O(\log n)\\
&\leq \frac{2a}{n}\cdot\frac{n(n-1)}{2}-\frac{2b}{n}\cdot\frac{3n}{4}\log\frac{n}{4}+O(\log n)\\
&= an-\frac{3b}{2}\log n+O(\log n)+\frac{3b}{2}\log 4-a\\
&\leq an-b\log n
\end{aligned}$$

Since A (after electing a leader) and B take constant time the expectation of the maximum energy dissipation is $O(H+\log n)$ where H is the expected height the

recursion tree. Since each sensor has an equal chance of being elected leader this is just the expected height of a binary search tree with random insertions. It is well known that this is $O(\log n)$ and more precise bounds are available [47]. Thus our algorithm has expected maximum energy dissipation per sensor of $O(\log n)$. Finally, Theorem 5 completes the proof of Theorem 6.

2.5.2 Sorting Simulations

Here we discuss the results of simulations of a basic implementation of our sorting algorithm. We simply counted the number of number of time units each sensor was awake and the total number of time steps the algorithm took. We define a single time unit to be the time it takes a sensor to broadcast a message with a logarithmic number of bits, and a single energy unit is the energy spent to stay awake for a single time step.

This implementation uses task completion testing with interruptions beginning at the expected runtime and interrupts every $\frac{1}{5}$ of the expected runtime. Since the B task takes constant time, we simply perform a reverse breadth-first traversal of the tree instead of using the general procedure outlined in Section 2.4 to complete the B tasks. The simulations were performed for networks ranging from 10 to 4,000 sensors with data averaged over 1,000 trials.

We compare the results for our sorting algorithm (BFR-QS) to the best performing algorithm (BSORT) from [49] and a more practical algorithm (MK-MS) presented in [30]. Table 2.1 provides the runtime and energy usage of these algorithms.

Figure 2.9 compares the runtimes of these sorting algorithms. Asymptotically, the runtime of BFR-QS outperforms BSORT and MK-MS, running in $O(n)$ expected time compared to $O(n \log n)$. The figure shows that the theoretical speed advantage

Algorithm	Runtime	Avg. Energy	Max. Energy
BSORT	$\sim 13n \log n$	$\sim 33 \log n$	$\sim 82 \log n$
MK-MS	$2n \log n$	$\frac{1}{2} \log^2 n + \frac{7}{2} \log n$	$\frac{1}{2} \log^2 n + \frac{7}{2} \log n$
BFR-QS	$\sim 7n$	$\sim 8 \log n$	$\sim 17 \log n$

Table 2.1: Runtime and energy dissipation of sorting algorithms.

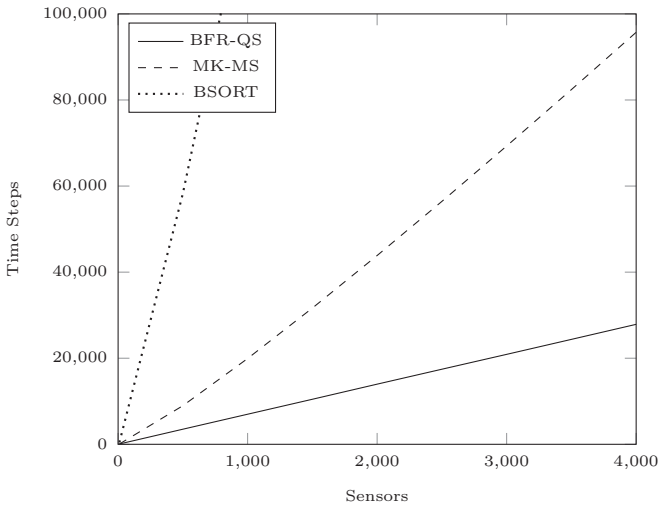


Figure 2.9: Runtime of sorting algorithms.

of BFR-QS also translates into practice as BFR-QS seems to be several times faster than MK-MS and orders of magnitude faster than BSORT.

Figure 2.10 compares the energy usage of these algorithms—omitting BSORT to keep the figure's scale reasonable. Also error bars representing plus or minus one standard deviation are provided for BFR-QS. In terms of energy usage BFR-QS and BSORT outperform MK-MS asymptotically with $O(\log n)$ energy per processor compared to $O(\log^2 n)$. However, in practice, BFR-QS and MK-MS perform similarly while BSORT uses much more energy. BFR-QS uses less energy than MK-MS when there are more sensors and MK-MS does better when there are fewer sensors.

One disadvantage of BFR-QS is that it is not as perfectly energy balanced as MK-MS is. The typical maximum energy dissipation of BFR-QS seems to be around

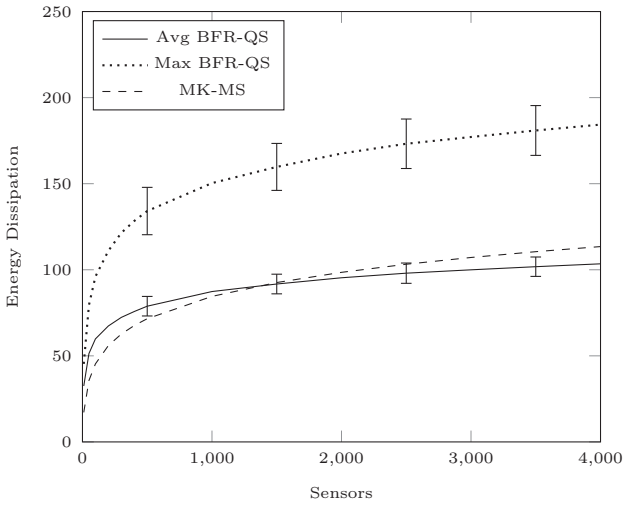


Figure 2.10: Energy dissipation per processor of sorting algorithms, including average energy dissipation and maximum energy dissipation of any processor for BFR-QS.

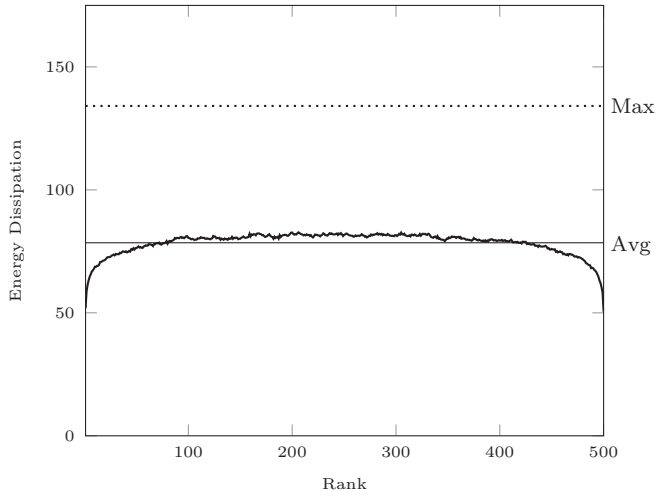


Figure 2.11: Energy dissipation by rank of BFR-QS sorting algorithm with 500 sensors, also including the average energy dissipated per processor over all ranks and expected maximum energy dissipation over all ranks.

double the energy dissipation of MK-MS. However, Figure 2.11 shows that energy dissipation of BFR-QS is nearly data independent. That is, no sensor with a given rank is—on average—going to use significantly more energy than the average energy dissipation over all ranks. This implies that if this algorithm is repeated many times

the maximum energy usage of any sensor will approach the average energy used per sensor, even if the data is adversely distributed.

2.6 Convex Hull

2.6.1 Convex Hull Algorithm

Due to the spatial nature of sensor networks many problems in computational geometry such as the Coverage Problem, All Points Nearest Neighbor, and 2-Dimensional Convex Hull are particularly important [24, 51, 56]. For example, in [48] sensors compute the convex hull of a region in a distributed manner to approximate the shape of forest fires. For the 2-Dimensional Convex Hull problem in this setting each sensor has a single two-dimensional point which typically represents the sensor's physical location and is usually obtained either through GPS or localization techniques [36]. The goal is for each sensor to determine if its point is an extreme point of the convex hull of the set of all points. Given a set of points P the convex hull of P is the set of convex combinations of the points in P, in other words, the set of linear combinations of points in P where the coefficients sum to exactly one.

$$\mathrm{CH}(P) = \left\{ \sum_{i=1}^{|P|} a_i p_i : \ p_i \in P \text{ and } a_i \geq 0 \ \forall 1 \leq i \leq |P| \text{ and } \sum_{i=1}^{|P|} a_i = 1 \right\}$$

Here we present a randomized algorithm using breadth-first recursion for the 2-Dimensional Convex Hull problem in a single-hop sensor network. Our result is output sensitive just like the serial result in [32] which, in our case, means that the runtime will be sublinear on many distributions.

Theorem 7. *Given a single-hop broadcast network S of n sensors with one 2-dimensional*

FindConvexHull(S, b_1, b_2) =

1: p = ElectLeader(S)
2: p broadcasts its coordinates
3: M = line perpendicular to (b_1, b_2) that crosses p
4: L = set of vertices from S left of M
5: R = set of vertices from S right of M
6: find edge of convex hull (l, r) that crosses M
7: FindConvexHull(L, b_1, l)
8: FindConvexHull(R, r, b_2)

Figure 2.12: Recursive algorithm for finding the convex hull between two base points b_1 and b_2.

point each, the convex hull of these points can be found using a single broadcast channel in expected time $O(\min(H \log n, n))$ *where* H *is the size of the convex hull, with expected energy dissipation per processor* $O(\log n)$*, and expected maximum energy dissipation of any processor* $O(\log^2 n)$*.*

We use a divide-and-conquer algorithm that uses breadth first recursion. At each recursive step we begin with two vertices known to be on the convex hull; we call these vertices base points b_1 and b_2. At each recursive step we find two adjacent vertices l and r—or in some cases a single vertex labeled both l and r—in the convex hull that lie between the two base points. Figure 2.13 provides an illustration of the vertices found at a single recursive step. We then recursively find the remaining vertices in the convex hull between l and b_1 and between r and b_2. Initially we find the leftmost and rightmost points to be b_1 and b_2 and make two recursive calls to find the points on the convex hull above and below (b_1, b_2), respectively. Without loss of generality, we will assume we are computing the convex hull above (b_1, b_2).

Now we describe the recursive step in more detail. First a vertex is elected to act as a pivot p. Let M be the line that goes through p that is perpendicular to (b_1, b_2). Then M partitions the possible points in the convex hull into a left and right half,

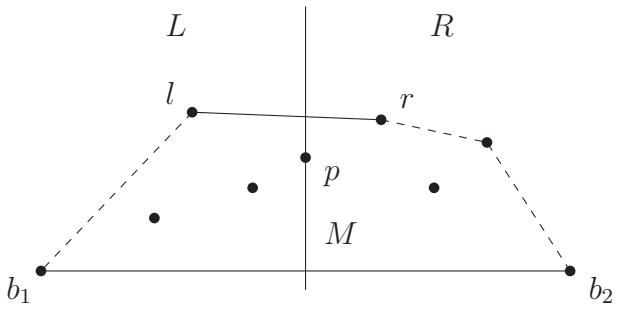


Figure 2.13: An example of a single recursive call of the algorithm in Figure 2.12. If p is the pivot then the algorithm finds l and r before recursing.

L and R respectively. For simplicity we when we refer to a vertex being more left or more right we mean when projected on the line (b_1, b_2). In other words we will treat (b_1, b_2) as the x-axis. Then the edge—or single point—of the convex hull that M crosses will be used for the base points in the recursive calls.

The line with one end point in L and the other in R that crosses M at the highest point is the edge of the convex hull that crosses M. For $v \in L$ (resp. R) define $h(v)$ to be the maximum height that a line between v and a point in R (resp. L) crosses M. We will keep track of two vertices $l \in L$ and $r \in R$ that maximize $h(l)$ and $h(r)$, respectively. We will alternate between electing a vertex from L and electing a vertex from R. Without loss of generality we describe the process of electing a vertex from L. The idea is that each vertex v that is elected will either improve our value of $h(l)$ or will eliminate vertices in L from being in contention for having the maximum value for h.

For the first election we use the algorithm from Figure 2.1. For the remaining elections we use the previous election to estimate the number of sensors and use the algorithm in Figure 2.2. Once a new vertex $l' \in L$ is elected it broadcasts its point and every vertex in R that lies above the line (l, l') broadcasts.

If no sensor announces then

- the point that is further to the right of l and l' becomes the new maximum, and
- every point that is below or on the line (l, l') and is to the left of the new maximum drops out.

If at least one sensor announces then

- the point that is further to the left of l and l' becomes the new maximum, and
- every point that is below or on the line (l, l') and is to the right of the new maximum drops out.

Lemma 8 proves that every point that drops out during this process does not maximize h. This is because if no point in R lies above (l, l') then for any of the sensors that dropped out there will be no point in R above the line between that point and the new maximum. Similarly if a point $r \in R$ lies above (l, l') then for any of the sensor that dropped out r lies above the line between that point and the new maximum. See Figure 2.14 for illustration.

Lemma 8. *Let $l_1, l_2 \in L$ be such that l_1 is further left than l_2. Then $h(l_1) > h(l_2)$ if and only if some point in R lies above (l_1, l_2).*

Proof. First note that for every point $r \in R$ (l_1, r) crosses M at a higher point than (l_2, r) if and only if r lies above (l_1, l_2). So if no $r \in R$ lies above (l_1, l_2) then $h(l_2) > h(l_1)$.

Notice also that the points where (l_1, r) and (l_2, r) cross M is above the point where (l_1, l_2) crosses M if and only if r lies above (l_1, l_2). Therefore if some $r \in R$ lies above (l_1, l_2) then $h(l_1) > h(l_2)$ because the highest crossing point of M for both l_1 and l_2 will come from a point in R that lies above (l_1, l_2). □

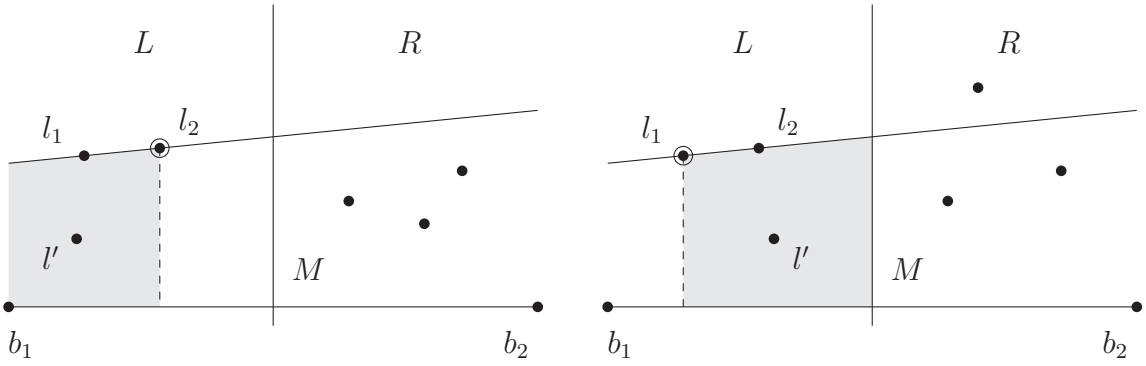


Figure 2.14: On the left no point in R lies above (l_1, l_2) and on the right a point in R does lie above (l_1, l_2). The l_i that maximizes h is circled and $h(l_i)$ is larger than $h(l')$ for any l' in the shaded region.

Further, since at least one sensor drops out after each election this process will eventually find the $l \in L$ that maximizes $h(l)$ and once both l and r have been found and once then know all other sensors have been eliminated this process is complete.

Next we analyze the runtime and energy usage of this process.

Lemma 9. *Given a set of n sensors the points in the convex hull that crosses M can be found in expected time $O(\log n)$ and the expected energy dissipation per processor $O(1)$.*

Proof. We prove this by showing that a constant fraction of the sensors drop out after a constant number of elections with constant probability. Since we alternate between time steps for elections for L and for R each sensor is only awake for twice the number of steps as if we were only doing elections for one of the sets. So in our analysis we will consider the time and energy for L to be elected.

We partition the set of vertices in L that have not yet dropped out into 3 sets:

- $\mathcal{H}$ the set of vertices that have a higher h value than our current l
- $\mathcal{L}$ the set of vertices that have a lower h value than l and are to the left of l

- $\mathcal{R}$ the set of vertices that have a lower h value than l and are to the right of l

Suppose a vertex v is elected. If $v \in \mathcal{H}$ then all the sensors with smaller h value either drop out or are now in $\mathcal{L}$ or $\mathcal{R}$. If $v \in \mathcal{L}$ then all the points in $\mathcal{L}$ who form a line with l with a larger slope drop out. This is because if (v', l) has a larger slope than (v, l) then v' lies below (v, l). Similarly if $v \in \mathcal{R}$ then all points in $\mathcal{R}$ that form a line with l with smaller slope drop out.

If $\mathcal{H}$ is larger than $\mathcal{L}$ and $\mathcal{R}$ then with at least $\frac{1}{6}$ probability a vertex from $\mathcal{H}$ is elected with h value greater than $\frac{5}{6}$ of all other vertices currently still in the election. If at least $\frac{1}{6}$ of the sensors drop out then we are done. Otherwise $\mathcal{H}$ is now smaller than at least one of $\mathcal{L}$ or $\mathcal{R}$.

If $\mathcal{L}$ is larger than $\mathcal{H}$ and $\mathcal{R}$ then with at least $\frac{1}{6}$ probability a vertex from $\mathcal{L}$ is elected whose line through l has larger slope than half the other vertices in $\mathcal{L}$. In this case these vertices dropout and at least $\frac{1}{6}$ of the vertices drop out. A similar argument holds if $\mathcal{R}$ is larger.

Thus after 2 elections there is at least a $\frac{1}{36}$ chance that $\frac{1}{6}$ of the sensors in L that were still awake drop out. Thus the expected number of elections is $O(\log n)$ and the expected number of elections each sensor is in before it drops out is $O(1)$.

From Lemma 4 the expected time for each election is $O(|\log \frac{n}{m}| + 1)$ where m is our estimate and n is the number of sensors in the election. If only a constant fraction of the sensors drop out then this will be constant. However if more than a constant fraction drop out then the next election will, in expectation, take time proportional to the negative log of the proportion of sensors that dropped. However this is just proportional to the time our process would have taken if only a constant fraction had dropped out at one time. Therefore this does not change the time analysis and only improves our energy dissipation. This means that the total time spent is in

expectation $O(\log n)$ the expected energy dissipation per processor is $O(1)$. $\square$

The initial values for b_1 and b_2 are the points with maximum and minimum x values which can be done in expected $O(\log n)$ time by using the algorithm from [38].

Notice every sensor in the recursive call with base points b_1 and l are to the left of M and every sensor in the recursive call with base points r and b_2 are to the right of M. Because the pivot p was chosen uniformly randomly the height of this recursion tree will be bounded by the height of a random binary search tree of size n so the expected number of recursive layers is $O(\log n)$, in expectation.

Thus in expectation the maximum energy per processor will be $O(\log^2 n)$ and the average energy per processor will be $O(\log n)$. Note that we cannot decrease the maximum energy per processor to $O(\log n)$ like in Section 2.5 because the energy usage per layer may be correlated with position. Also, the number of recursive calls is $O(H)$ so the total amount time taken for all recursive calls will be $O(H \log n)$. Further this time is upper bounded by $O(n)$ like in Section 2.5 because the time taken at each recursive layer is logarithmic in the number of sensors in that recursive call. Finally, Theorem 5 completes the proof of Theorem 7.

2.6.2 Convex Hull Simulations

Here we discuss the results of simulations of a basic implementation of our convex hull algorithm. As with the sorting algorithm in Section 2.5 we simulated this algorithm with task completion testing interruptions beginning at the expected runtime and subsequent interrupts occurring every $\frac{1}{5}$ of the expected runtime. This algorithm does not contain a B task so we did not need to traverse back up the recursion tree.

The simulations were performed for networks ranging from 10 to 4,000 sensors with data averaged over 1,000 trials on points that are chosen from three different

probability distributions: the uniform distribution on a circle (just the perimeter), the uniform distribution on a disk (includes the center), and the normal 2-dimensional Gaussian distribution. These distributions were chosen to represent realistic sensor placements. The uniform circle represents the case where sensors are strategically placed on the boundary of an area, the uniform disk represents sensors that are purposefully evenly distributed, and the Gaussian represents sensors that are less carefully scattered resulting in sensors being concentrated towards the center. Each distribution has different convex hull sizes which is relevant to the algorithm's runtime and energy usage. The circle is the worst case with all n points on the convex hull, the disk has $O(n^{\frac{1}{3}})$ points on the convex hull in expectation, and the Gaussian distribution has $O(\sqrt{\log n})$ points on the convex hull in expectation [46].

The results of the simulations of our convex hull algorithm on the distributions previously described are given in Figures 2.15, 2.16, and 2.17 comparing the runtime, average energy usage, and maximum energy usage, respectively. The figures include error bars representing plus or minus one standard deviation. These results largely reflect the theoretical bounds given in Section 2.6. On the uniform circle our algorithm's runtime is linear with respect to the number of processors while for the uniform disk and Gaussian the runtime is sublinear. Although our convex hull runtime on the uniform circle is significantly slower than the uniform disk and Gaussian it is still faster than the previous best sorting algorithm, MK-MS. Even though these algorithms solve different problems this illustrates the speed that this framework allows. The average energy is logarithmic for all three distributions with the constants depending upon the size of the convex hull. The maximum energy usage appears to be between log and log-squared. Interestingly, although having the highest maximum energy usage in practice, the uniform circle appears to have logarithmic maximum energy used while the disk and Gaussian appear to have $\omega(\log n)$ maximum energy.

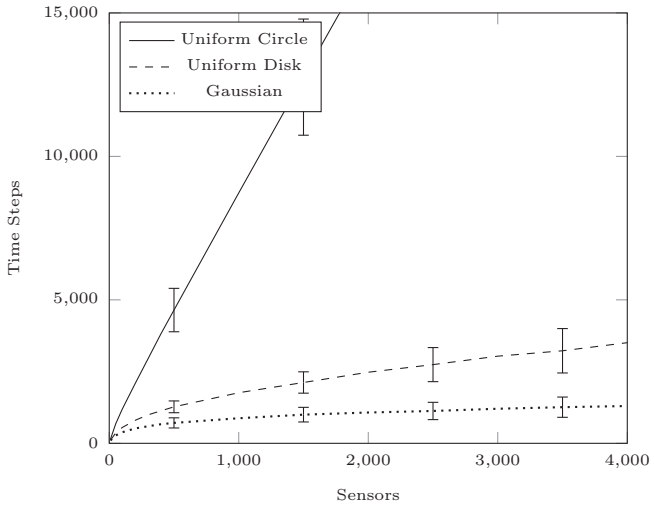


Figure 2.15: Runtime of our convex hull algorithm under different point distributions.

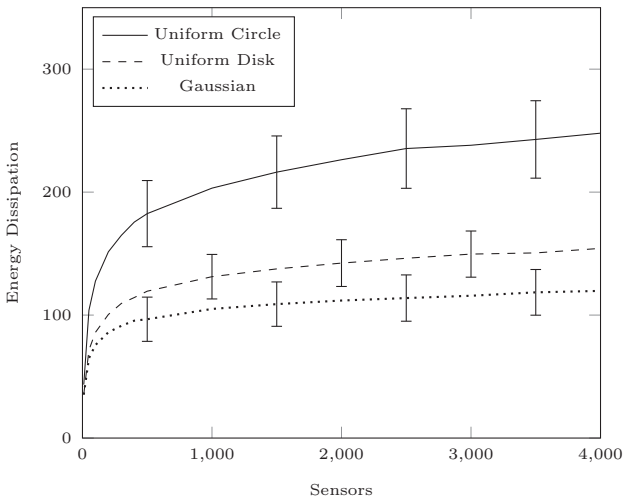


Figure 2.16: Average energy dissipation per processor of our convex hull algorithm under different point distributions.

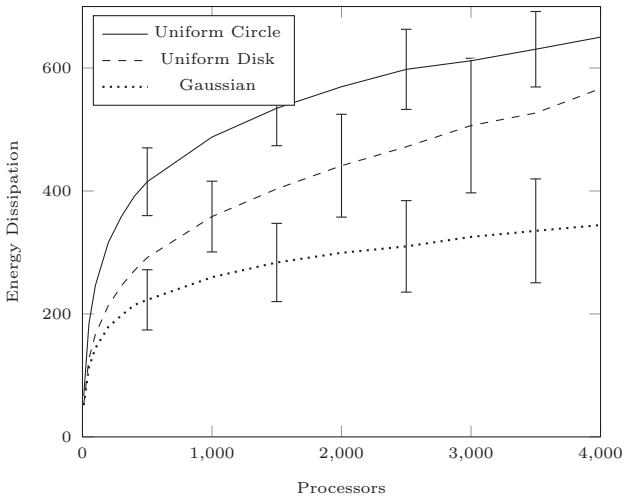


Figure 2.17: Maximum energy dissipation of any processor of our convex hull algorithm under different point distributions.

Chapter III

Multi-Hop Algorithms

3.1 Introduction

This chapter focuses multi-hop algorithms and overcomes challenges in merging local solutions into global solutions. In Section 3.2 we present a technique called *multi-hop consolidation algorithms* in which sensors execute single-hop algorithms in overlapping single-hop regions and then consolidate these local solutions into a global solution. One major consideration when creating multi-hop consolidation algorithms is whether or not the network has an established global coordinate system such as GPS. If a sensor network has a global coordinate system then these coordinates can be used to coordinate usage of the radio broadcast channel. If a sensor network does not have an established global coordinate system then we elect leaders and use the leaders to coordinate usage of the radio broadcast channel.

Multi-hop consolidation algorithms can only be used to solve inherently local problems such as many geometry problems. Because of the spatial nature of sensors these geometry problems are especially relevant. In Subsection 3.3 we create a consolidation algorithm for all points k-nearest neighbors, which is often used to detect

outliers in sensor data [18]. In Subsection 3.4 we present a consolidation algorithm for coverage boundary detection. Finally, in Subsection 3.5 create a consolidation algorithm for computing the Voronoi diagram which is an important general-purpose geometric structure.

3.2 Consolidation Algorithms

Here we define *multi-hop consolidation algorithms*. These algorithms have two parts: first a single-hop algorithm is used to find local solutions to the problem, then overlapping single-hop solutions are merged together using a specific type of merge we call a *consolidation*. We will refer to the execution of the local algorithm or the execution of the consolidation as a *local action*. We call a disk of diameter r_c, where r_c is the communication radius, a *single-hop region* in which every sensor in the region can listen to radio broadcasts from every other sensor in the region. We distinguish this from a larger disk of radius r_c that we call a *broadcast range disk* which is the area in which other sensors can listen to radio broadcasts sent by the sensor at the center of the disk. The single-hop algorithm will be executed in several overlapping single-hop regions, computing local solutions for those regions. Note that a sensor will likely be involved in multiple instances of the single-hop algorithm. Then once the local algorithms are complete the consolidation step merges the local solutions into a global solution. This consolidation may be as simple as taking the minimum of the single-hop solutions or may be much more involved.

The advantage of a multi-hop consolidation algorithm is that it is much easier to create and execute algorithms in single-hop networks. However, consolidation algorithms are also constrained to only being applicable to problems that are inherently local. Due to the spatial nature of sensor networks, inherently local geometry prob-

lems are actually quite common in sensor networks and consolidation algorithms are a powerful tool.

We define an *α-consolidation algorithm*, $0 < \alpha \leq 1$, to be a multi-hop consolidation algorithm where every sensor is involved in a single-hop instance of the algorithm with every other sensor within distance αr_c. That is, every sensor is involved with multiple single-hop instances, the union of which includes every sensor within distance αr_c.

3.2.1 Consolidation Algorithms With Global Coordinate System

Here we present a procedure for executing $(1-\epsilon)$-consolidation algorithms in $O\left(\frac{T}{\epsilon^2}\right)$ time and $O\left(\frac{\log T}{\epsilon^2}\right)$ energy overhead per sensor, where T is the maximum runtime of any local action. With a known coordinate system sensors can use the global coordinate system to pre-decide usage of the broadcast channel. The basic idea is to disregard sensor positioning and prearrange overlapping disks of of diameter r_c to cover the entire plane so that if any two points in the entire plane are within $(1-\epsilon)r_c$ then they lie in the same disk at least once. Then, based on the position of the disks and a prearranged ordering, we execute the local algorithm on disks in specific pattern such that they sufficiently far apart so there is no interference, stopping after a constant number of steps so other disks can begin working. This is continued until every disk has had an opportunity to get some work done and then we repeat the pattern.

The covering that we use is often attributed to Kershner [29]. The idea is to have the center of the disks be the vertices of a tiling by equilateral triangles. In our case we use triangles with edges of length $\frac{\epsilon r_c\sqrt{3}}{2}$. Notice that no global communication is

needed because for a given ϵ and r_c sensors always use the exact same pattern.

Lemma 10. *If disks of radius $\frac{r_c}{2}$ cover the plane with centers being the endpoints of a tiling by equilateral triangles of edge length $\frac{\epsilon r_c\sqrt{3}}{2}$ then for any two points p_1 and p_2 that are within $(1-\epsilon)r_c$ there exists a disk in the covering that both p_1 and p_2 lie in.*

Proof. Let m be the midpoint of (p_1, p_2). Notice that any point on the plane, specifically m, is within $\frac{\epsilon r_c}{2}$ at least one of endpoint t of one of the triangles in the tiling. Thus, by the triangle inequality, the distance from p_1 or p_2 and t is at most $\frac{(1-\epsilon)r_c}{2} + \frac{\epsilon r_c}{2} = \frac{r_c}{2}$. Thus p_1 and p_2 both lie on the disk centered at t. □

We then divide these centers up into different regions so that we can execute one local algorithm per region without interference from the other regions. To do this we choose centers of each region to be centers of a disk such that each center of the region is at least $2r_c$ away from each other, using the exact same tiling scheme as before. Then every disk center is associated with the region whose center it is closest to, breaking ties arbitrarily (for example the rightmost regional center it is closest to).

We divide the runtime into $O\left(\frac{1}{\epsilon^2}\right)$ repeating constant-sized time slices, one for each center in each region. Each center is assigned a time slice based on a predetermined pattern, Figure 3.1 shows one such pattern. Then each during the time slices assigned to each center every sensor within $\frac{r_c}{2}$ of the center wakes up and executes the local algorithm. Note all sensors are within a single-hop of every other sensor in the disk and no disks will ever interfere with each other because corresponding disks in different regions are at least r_c apart.

We then interrupt this cycle at exponentially increasing number of cycles to test, for each disk, if every overlapping disk is also complete. During each interruption we iterate through the disks twice in the same order that we do during execution of

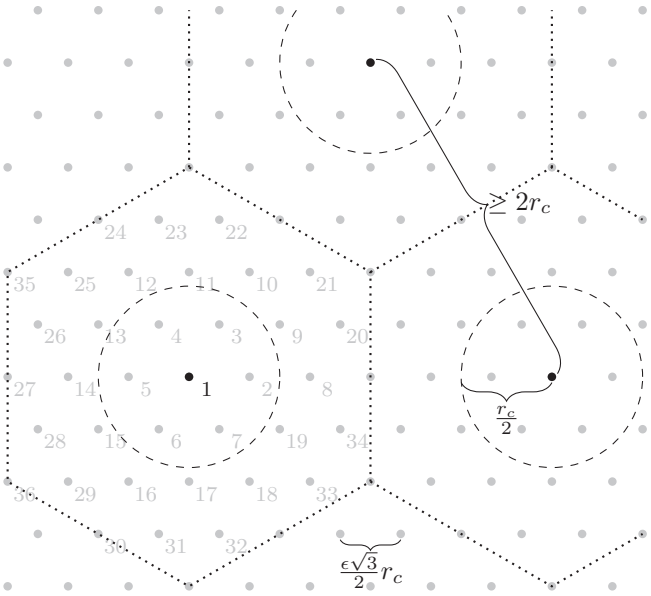


Figure 3.1: Tiling of disk centers with $\epsilon = \frac{2\sqrt{3}}{9} \approx 0.38$ showing regions, regional centers, and the order that in which disks in a specific region are executed.

the local algorithm. During the first iteration any sensor that knows that the disk is not done with the local algorithm broadcasts. During the second iteration every sensor that was in a disk where a sensor announced that the local algorithm was not complete broadcasts. After a disk discovers that every overlapping disk is also complete they then use the time steps allocation to the disk for the consolidation procedure.

3.2.2 Consolidation Algorithm Without Global Coordinate System

If no global coordinate system is available then picking disk centers while preventing interference cannot be done for free. Consolidation algorithms with no known global coordinate system will still greatly resemble consolidation algorithms with a known coordinate system but will be more involved. The major difference is that now

sensors will be elected to be disk centers and time slices will be distributed on the fly.

We present a procedure for executing $\left(\frac{1}{2}-\epsilon\right)$-consolidation algorithms. The expected runtime of our procedure is $O\left(\frac{T+\log^2 n}{\epsilon^2}\right)$ time, where T is the maximum runtime of any local action. In expectation, the average energy overhead per sensor is $O\left(\frac{\log T+\log n}{\epsilon^2}\right)$ and the maximum energy overhead for any sensor is $O\left(\frac{\log T+\log^2 n}{\epsilon^2}\right)$.

A couple of our model assumptions will be key for this procedure. The first is the assumption that every sensor has a unique id of $O(\log n)$ bits. The second is that sensors can determine the distance from a broadcasting sensor in constant time by observing the strength of the received signal.

Our procedure takes place in three phases. These phases are thought of as sequential, however we actually partition the algorithm's runtime into constant sized time slices and assign every other chunk of time slices to each phase. This is to prevent unexpected interference between sensors in different areas currently working on different phases.

- **Phase 1:** Elect leaders to be disk centers and determine which sensors lie within the disks.
- **Phase 2:** Allocate time slices and determine if local algorithms are complete.
- **Phase 3:** Run local algorithm and consolidate local solutions.

3.2.3 Phase 1: Electing Leaders

In this phase our goal is to elect leaders who will act as the centers of the $\frac{r_c}{2}$ radius disks in the known global coordinate system and will help coordinate sensors in its disk during phase 2. Sensors will remain awake for this entire phase until they are certain no more leaders will be elected nearby.

When electing leaders we disallow sensors that are within ϵr_c of another leader from becoming a leader and we continue electing leaders until every sensor is either a leader or within ϵr_c of a leader. In other words we find a maximal set of sensors such that no two leaders are within ϵr_c of any other leader. The following lemma gives an upper bound on the number of leaders whose $\frac{r_c}{2}$–disks can interfere with a given leader's $\frac{r_c}{2}$–disk. The proof follows easily from a geometric argument.

Lemma 11. *If P is a set of points on a 2D plane and $L \subseteq P$ is a maximal set such that for all $u, v \in L$ we have $d(u, v) > \epsilon r_c$ then for all $u \in L$, $|\{x \in L \text{ such that } d(x, u) \leq 2r_c\}| \leq \frac{1}{\epsilon^2}$*

During phase 1 each sensor can be in one of 4 states: *leader*, *candidate*, *potential candidate*, or *non-leader*. Each sensor begins as a candidate and eventually becomes either a leader or a non-leader. Candidates are sensors currently in contention to be elected as a leader. Potential candidates are not currently in contention to become a leader but may come back in contention at some point and become a candidate again.

To elect leaders we repeatedly use the broadcast structure given in Figure 3.2. This election is based on the exponential back off where candidates become non-candidates with probability $\frac{1}{2}$. Steps 1-3 tell potential candidates that they cannot yet become candidates again. Steps 4-5 are used for a candidate to determine no other candidates are nearby and thus become an elected leader. Finally, step 6 is used to determine if any more leaders are going to be elected, indicating whether or not sensors have completed phase 1.

If a sensor is a candidate and does not broadcast in step 1 it becomes a potential candidate. Potential candidates keep track of a variable i that represents how many iterations it has been out of contention. If a broadcast or interference is heard in steps 1-3 the potential candidate increments i. If no broadcast or interference is heard in

1: **broadcast** with probability $\frac{1}{2}$ if currently a candidate
2: **broadcast** if a broadcast or interference was heard in 1
3: **broadcast** if a broadcast or interference was heard in 2
4: **broadcast** if interference was heard in 1
5: **broadcast** if broadcasted in 1 and no broadcast or interference was heard in 4
6: **broadcast** if candidate or potential candidate

Figure 3.2: The repeating broadcast steps for electing leaders in phase 1.

steps 1-3 the potential candidate decrements i and if i becomes 0 then the potential candidate becomes a candidate again.

If a candidate broadcasts in step 5 that means that it has been elected as a leader. During this step sensors around a leader determine their distance to the leader. If this distance is less than $\frac{r_c}{2}$ then the sensor knows it is in the leader's disk and if this distance is less than ϵr_c then the sensor becomes a non-leader.

If a sensor hears no broadcast or interference in step 6 then it can sleep for the remaining time steps allocated to phase 1 because there will be no more leaders elected within $\frac{r_c}{2}$ of the sensor. When this occurs for every sensor in the network we say that phase 1 is *complete*. Notice that no sensor can ever be certain that phase 1 is complete so we continue allocating time steps to phase 1 even after every sensor falls asleep.

Lemma 12. *The expected time for phase 1 to complete is* $O\left(\frac{\log n}{\epsilon^2}\right)$.

Proof. First we will find the expected time for a given sensor s to complete phase 1. We do this using the potential method and random walks.

Our potential function f is defined below. We define l to be the number of current leaders within $3r_c$ of s, c and p_k to be the number of current candidates and potential candidates with $i = k$ within r_c of s, respectively. We define L to be the set of sensors within $3r_c$ of s that will eventually become a leaders in the future. We define a function g on the elements of L to be n if the element is the not the next sensor

to be elected and to otherwise be the number of sensors who are as close to being elected. We say a sensor u is closer to being elected than another sensor v if u is a candidate and v is not or u and v are potential candidates and u has a smaller value for i.

$$f(s) = 2^{\frac{36}{\epsilon^2}-l}\left(c + \sum_{k=1}^{\infty}\left(\frac{1}{6}\right)^k p_k\right)\prod_{v\in L} g(v)$$

Examine the following cases:

1. The next leader is elected.
2. There are no candidates within two hops of the next leader.
3. There are candidates within two hops of the next leader but none of them broadcast.
4. The next leader is a candidate, broadcasts in step 1, and at least $\frac{1}{3}$ of the candidates within two hops of the next leader drop out.
5. The next leader is a candidate, broadcasts in step 1, and less than $\frac{1}{3}$ of the candidates within two hops of the next leader drop out.
6. The next leader is a candidate and does not broadcast in step 1.
7. The next leader is not a candidate.

Notice that this in each of these cases there is a constant probability that the potential function is decreased by a constant amount. Therefore since this is a biased random walk, the expected time for a given sensor to complete phase 1 is the log of the initial value of $f(s)$ which is $2^{O\left(\frac{1}{\epsilon^2}\right)} \cdot O(n)$. Thus the expected time for a given

sensor to complete is $O\left(\frac{\log n}{\epsilon^2}\right)$ and using Lemma 36 in the Appendix completes the proof.

□

3.2.4 Phase 2: Allocating Time Slices

In this phase each leader attempts to claim time for its single-hop region to complete the local action. Local actions are completed in Phase 3 which is divided into $O\left(\frac{1}{\epsilon^2}\right)$ constant sized time slices. The leading coefficient will be important and is discussed in the proof of Lemma 13. Each leader will claim a time slice so that no leaders whose disks interfere with each other can claim the same time slice. Sensors can strategically sleep during many parts of phase 2 but for our analysis we will assume sensors are awake for all of phase 2 until it is complete with phase 1 and all of its leaders have claimed a time slice of phase 3.

Phase 2 takes place in several repeated rounds with each round consisting of $\log_2 \frac{1}{\epsilon^2}$ stages. During each stage $O\left(\frac{1}{2^i\epsilon^2}\right)$ time slices for phase 3 will be available to be claimed by each group. As an example, perhaps during the first stage time slices 1 through 64 are available, then in stage two time slices 65 through 96 are available, then in stage three time slices 97 through 112 are available, and so on. For each time slice available to be claimed in a stage that stage will contain time for that time slice to be claimed.

Each leader who has not yet successfully claimed a time slice picks a random slice that is available to be claimed and attempts to claim that time slice. In Figure 3.3 we detail the process in phase 2 for claiming a time slice. If this claim is successful then this group ignores the remaining stages and continues to claim this time slices each round until every group that could interfere also successfully claims a slice. If

the group fails to claim a time slice then the leader picks another random time slice in the next stage to attempt to claim. If a group fails to claim a time during a round then the group simply repeats this whole process in the following round.

To efficiently assign a time slice to a group of sensors we must resolve a problem related to the *hidden node problem*. This problem, which we call the *multi-interference information loss problem*, occurs when two groups of sensors attempt to exchange information (in this case claim a time slice) and only a small number of sensors are actually in communication range of the other group. If a group attempts to exchange information by having more than one sensor broadcast and use interference or lack thereof to represent a bit of information, then the groups lose the ability to detect collisions. This then essentially becomes the *loneliness detection* problem where each group is attempting to determine if there exists another group trying to claim the same time slice. As discussed in [9] it can be impossible to detect loneliness in some circumstances.

In this situation every sensor in the group must broadcast at least once because it may be the only sensor which is in range of communicating with the other group. However we cannot solve this problem by having each sensor in the convex hull announce one at a time because then this would require the other group to listen for $O(n)$ steps.

We resolve the multi-interference information loss problem by using the unique id of the group's leader. For each bit in the leader's id each sensor broadcasts if the bit is a 1 and only listens if the bit is a 0. Then if multiple groups broadcast the leaders' ids at the same time then all but at most one of the groups will have a sensor that hears a broadcast that does not correspond to its leader's id.

In step 1 any leader that decides to claim a time slice in the partition of phase 3 time slices associated with this claiming time broadcasts. If any sensor heard

1: **broadcast** if you are a leader that is attempting to claim a time slice in the corresponding partition
2: **broadcast** if you heard interference in 1
3: **broadcast** your id and the group you are attempting to claim if you are a leader who broadcast in 1 and did not hear a broadcast or interference in 2
4: **broadcast** the leader's id bit pattern if you heard your leader broadcast in 3
5: **broadcast** if you broadcast in 4 and heard a broadcast or interference that does not correspond to your leader's id in 4
6: **broadcast** if you are a leader that broadcast in 3 and did not hear a broadcast or interference in 5
7: **broadcast** if your leader broadcast in 6.
8: **broadcast** if you heard a broadcast or interference in 7

Figure 3.3: The broadcast schedule used in phase 2 for claiming a time slice of phase 3.

interference in step 1 then it cannot determine if the sensors broadcasting are its leaders and must stop all broadcasting sensors from going through with the claim. So these sensors broadcast in step 2. If a leader did not hear a broadcast or interference in step 2 then it knows every sensor in its $\frac{r_c}{2}$–disk heard it broadcast and knows its id. So the leader broadcasts a confirmation message which every sensor in its group are guaranteed to hear.

This does not guarantee that there can be no interference between two different $\frac{r_c}{2}$–disks so each disk broadcasts the bit pattern of the leader in step 4. Then if multiple interfering $\frac{r_c}{2}$–disks attempt to make a claim during this time at most one claim will go through. In step 5 sensors indicate if their groups claim did not go through by broadcasting if they heard a broadcast in step 4 that does not correspond to their leaders bit pattern. If a leader did not hear a broadcast or interference in 5 then it is safe for the $\frac{r_c}{2}$–disk to claim the time slice so the leader sends a confirmation message in step 6 finalizing the claim. Then in step 7 every sensor in the disk broadcasts that a time slice in the group was taken. Then in step 8 sensors let their leaders know that a time slice in that group was taken.

During each round when a disk claims a time slice, other sensors that are not in the disk may not hear the disk's leader announcing which time slice is claimed. They hear that their disk interferes with a disk that claimed a time slice in that group but not which time slice that is.

Lemma 13. *The expected time for a sensor to complete phase 2 is* $O\left(\frac{\log n}{\epsilon^2}\right)$ *and the expected time for all sensors to complete phase 2 is* $O\left(\frac{\log^2 n}{\epsilon^2}\right)$.

Proof. First we show that each sensor has a constant probability of finishing Phase 2 after each round. This will then imply the expected maximum number of rounds is $O(\log n)$ for any sensor by Lemma 36. Since the energy spent per round is $O\left(\frac{\log n}{\epsilon^2}\right)$ this will complete the proof.

Let $m = \frac{\pi(2r)^2}{\pi(\epsilon r_c)^2} = \frac{4}{\epsilon^2}$ which is the maximum number of disks that can interfere with a given disk while attempting to claim a time slice. For the sake of this proof we will pick the total number of time slices to be $32m$ so that at each stage a large fraction of the disks will successfully claim time slots. If at each stage only $\frac{1}{4}$ of the remaining disks within interference range fail claim a time frame then all these disks will be allocated a time slice at the end of the round. So we bound this probability.

There is one final caveat that we need to take into account: if a large number of groups many hops away fail to claim time slices this could cascade causing neighboring groups to fail to claim time slices in the next stage, which in turn would cause their neighbors to fail to claim time slices in the next time stage and so on. To count these dependencies we will use a hexagonal Kershner covering, see Figure 3.4. Our invariant for each stage will then become that at stage $\log_4(n) - i$, for each disk in the Kershner covering with side length i and radius equal to the interference range between groups, that at most $\frac{1}{4}$ of the leaders fail to claim a time slice. Note that the Kershner covering of side length i has $3i^2 + 3i + 1$ points.

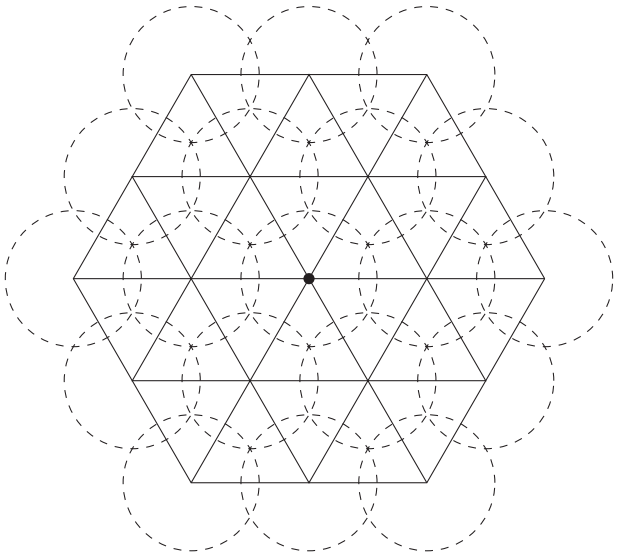


Figure 3.4: Hexagonal Kershner covering with side length of two

Assuming that in all previous rounds at most $\frac{1}{4}$ of the groups fail to claim a time slice, the probability that any group fails to claim a time slice is at most $\frac{1}{16}$ because the number of groups are decreasing at a faster rate than the number of time slices available to claim. Therefore, assuming in all previous rounds that at most $\frac{1}{4}$ of interfering groups failed to claim a time slice, the probability that at most $\frac{1}{4}$ of the interfering groups fail to claim a time slice in round $\log_4(n) - i$ in each disk of the hexagonal Kershner covering with side length i is bounded by

$$1 - \sum_{j=0}^{4^{i-1}} \binom{4^i}{j} \left(\frac{1}{16}\right)^j \left(\frac{15}{16}\right)^{4^i - j}$$

Table 3.1 shows the probability that more than $\frac{1}{4}$ of the groups within interference range fail to claim a time slice in the $\log_4(n) - i$th stage in each disk of the Kershner covering of side length i and radius equal to the interference range between groups, assuming that in previous rounds at most $\frac{1}{4}$ of groups failed to claim a time. For all stages of the form $\log_4(n) - i$ with $i \geq 4$ we use the Hoeffding/Chernoff bound to

i	equation	exact value
0	$\frac{1}{16}$	0.0625
1	$7\left(1-\sum_{j=0}^{1}\binom{4}{j}\left(\frac{1}{16}\right)^{j}\left(\frac{15}{16}\right)^{4-j}\right)$	0.151
2	$19\left(1-\sum_{j=0}^{4}\binom{16}{j}\left(\frac{1}{16}\right)^{j}\left(\frac{15}{16}\right)^{16-j}\right)$	0.044
3	$37\left(1-\sum_{j=0}^{16}\binom{64}{j}\left(\frac{1}{16}\right)^{j}\left(\frac{15}{16}\right)^{64-j}\right)$	1.01e−5
≥ 4	$\sum_{i=4}^{\infty}\left(3i^2+3i+1\right)e^{-\frac{9}{128}4^i}$	9.29e−7

Table 3.1: Upper bound on probability that more than $\frac{1}{4}$ of the groups within interference range fail to claim a time slice in the $\log_4(n)-i$th stage in each disk of the Kershner covering of side length i and radius equal to the interference range between groups, assuming that in previous rounds at most $\frac{1}{4}$ of groups failed to claim a time slice.

get an approximation. Notice that the sum of these probabilities is a constant less than 1. Thus each sensor has a constant probability of being finished with Phase 2 after each round. This implies the expected maximum number of rounds is $O(\log n)$ for any sensor by Lemma 36 in the Appendix. Since the energy spent per round is $O\left(\frac{\log n}{\epsilon^2}\right)$ this completes the proof.

□

3.2.5 Phase 3: Executing Local Actions

This phase is where sensors complete the local algorithms and merge. Phase 3 is divided into $O\left(\frac{1}{\epsilon^2}\right)$ repeating constant sized time slices. If a group is allocated a time slice it wakes up during that time slice and continues work on either the local algorithm or consolidation algorithm. At exponentially increasing intervals the execution of the local algorithm is paused and every sensor that knows that the local algorithm is not complete announces. Once the local algorithm is complete the

sensors wake up at exponentially increasing intervals to determine if they can begin the consolidation algorithm. During one of these wake-ups if a sensor is a member of a disk that is not complete or the sensor has not completed phase 1 or 2 the sensor announces.

This phase will complete in $O\left(\frac{T}{\epsilon^2}\right)$ time after phase 2 is complete and only $O\left(\frac{\log T}{\epsilon^2}\right)$ energy is used per sensor in addition to the energy used for the local actions.

3.3 All Points k-Nearest Neighbors

In this section we provide an algorithm for the 2-dimensional all points k-nearest neighbors problem for serial computers, single-hop sensor networks, and multi-hop sensor networks. The *all points k-nearest neighbors problem* is defined as follows: given a set P of n 2-dimensional points in L^p space, for each point p find $k < n$ neighboring points in P such that no other point in P lies closer to p than any of the k points found. In parallel models we assume that each processor is given only a single point. For the multi-hop model, if a sensor has fewer than k neighbors within broadcast range then we only require that the sensor find all neighbors within broadcast range.

Nearest neighbors is an extremely relevant problem to wireless sensor networks. All points k-nearest neighbors is used in [50] to solve certain coverage problems. Another common use of all points nearest neighbors is to detect outliers in sensor data [18]. Since sensors are small and unreliable detecting when sensors give faulty data is a concern. To detect faulty data, sensors may compare their data with neighboring sensors. Typically nearest neighbors are chosen because their close proximity means that their data will be more highly correlated.

Determining each sensor's nearest neighbors is trivial if each sensor has complete

information of all neighboring sensors, but in extremely dense networks this can be too expensive in terms of energy usage. However we do not assume that sensors know about neighboring sensors and still each sensor only uses at most poly-log energy.

First we present a $O(kn \log n)$ serial version of of our k-Nearest Neighbors algorithm and prove the correctness of our approach. This algorithm is, in essence, the all nearest neighbors algorithm from [7]. We describe the algorithm more explicitly and generalize the algorithm for finding the k nearest neighbors for every L^p-space, $1 \leq p \leq \infty$. Using multidimensional divide-and-conquer [7] points out that this algorithm can be generalized to any fixed d dimensions in $O(kn \log^{d-1} n)$ time. This algorithm is, to our knowledge, the simplest $O(n \log n)$ planar all nearest neighbor algorithm.

We then show how this algorithm can be implemented in a single-hop sensor network using a maximum of $O(k \log n)$ energy per sensor. Finally, we use this algorithm and the framework in Section 3.2 to create our multi-hop algorithm.

3.3.1 Serial All Points k-Nearest Neighbors

For the sake of simplicity we assume that no two points have the same y-coordinate. Pseudocode for the following algorithm is described in Figure 3.6. The basic form of our algorithm follows the divide-and-conquer pattern. We first find the point with the median x-coordinate value and then divide the given set of points P into two equal sized halves, the left half L and the right half R. We then recursively solve the All Points k-Nearest Neighbors for L and R. Now for each point in L we need to find the neighbors in R that are nearer than the k^{th} nearest neighbor from the points in L already discovered. We also need to do a symmetric task for each point in R.

To discover the nearest neighbors on from the opposing set of points we will do two

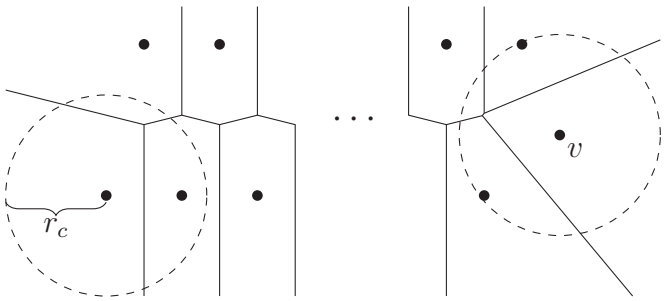


Figure 3.5: To find the 5 nearest neighbors for each processor or to compute the global Voronoi diagram sensor v must transfer information for every sensor from the top half of the network to the bottom and vice versa.

NearestNeighbor(P, k) =

1: divide P by the median x-coordinate into left points L and right points R
2: NearestNeighbor(L, k)
3: NearestNeighbor(R, k)
4: $L_{\text{help}} = R_{\text{help}} = \{\}$
5: **FOR EACH** $p \in P$ by ascending y-coordinate
6: **IF** $p \in L$
7: **FOR EACH** $p_{R\text{-help}}$ in R_{help}
8: add p to $k$nn$[p_{R\text{-help}}]$
9: add p to L_{help}
10: **FOR EACH** $p_{L\text{-help}} \in L_{\text{help}}$
11: p_{med} = (median x-coordinate, p's y-coordinate)
12: $p_k = k^{\text{th}}$ nearest neighbor to $p_{L\text{-help}}$
13: **IF** $\text{dist}(p_{L\text{-help}}, p_k) \leq \text{dist}(p_{L\text{-help}}, p_{\text{med}})$
14: remove $p_{L\text{-help}}$ from L_{help}
15: **ELSE**
16: same as lines 7-14 switching L and R
17: $L_{\text{help}} = R_{\text{help}} = \{\}$
18: **FOR EACH** $p \in P$ by descending y-coordinate
19: same as lines 6-16
20: **FOR EACH** $p \in P$
21: remove all but k points nearest to p from $k$nn$[p]$

Figure 3.6: Serial $O(kn \log n)$ k-Nearest Neighbors algorithm.

serial scans. First by ascending y-coordinate and then by descending y-coordinate. Note that we do not need to sort at each recursive level because we can just merge the sorted list of points from the previous recursive level in $O(n)$ time. During the

ascending serial scan we find each point from the opposing set that has higher y-coordinate is nearer than the k^{th} nearest neighbor already found. Similarly, during the descending serial scan we find each point from the opposing set that has lower y-coordinate than the k^{th} nearest neighbor already found.

During each scan we keep track of a set of points for each side, labelled L_{help} and R_{help} respectively. Each set will contain all points for which there could exist a neighbor from the opposing set later on in the scan that is nearer than the k^{th} nearest neighbor already found. When we reach a point from L during our scan we add the point to the set of possible nearest neighbors for each point in R_{help}. We then determine if this point belongs in L_{help} and remove any points that no longer belong in the set. To determine which points should be removed we just need to compare the distance from the k^{th} nearest neighbor already found and the point with x-coordinate equal to the median x-coordinate and y-coordinate equal to the current y value of the scan. It is easy to see that any point from the opposing set yet to come in the scan must be further away than this point. We perform a similar task when our scan reaches a point in R replacing L_{help} and R_{help} respectively.

Finally, for each point we use a linear time selection algorithm to determine the nearest k points from the potential nearest points discovered during the scans and the recursive calls. By translating the x and y axes to the current position in the scan and the median x value respectively, Lemma 14 implies that L_{help} and R_{help} are at most $2k$ at all times. This means that each step in the scan takes $O(k)$ time and that the total size of all sets of possible nearest neighbors is $O(kn)$. Thus each recursive call takes $O(kn)$ time and the total runtime of our algorithm is $O(kn \log n)$.

Lemma 14. *Let P be a set of 2-dimensional points in L^p-space with positive x and y-coordinates. At most $2k$ points in P are closer to the origin than their k^{th} nearest*

neighbor in P.

Proof. Observe the two regions obtained by dividing the first quadrant into two octants by the line $y = x$. Let $a = (x_a, y_a)$ and $b = (x_b, y_b)$ be two points in a single octant. Without loss of generality assume that they lie in the octant above the line $y = x$—that is $y_a \geq x_a$ and $y_b \geq x_b$—and assume that $y_a \geq y_b$.

If $x_a \geq x_b$ then we get $|x_a - x_b|^p + |y_a - y_b|^p \leq |x_a|^p + |y_a|^p$ directly. On the other hand if $x_a \leq x_b$ then from the triangle inequality

$$\begin{aligned} |x_a - x_b|^p + |y_a - y_b|^p &\leq |x_b - x_a + y_a - y_b|^p \\ &\leq |y_a + x_b - y_b|^p \\ &\leq |y_a|^p \\ &\leq |x_a|^p + |y_a|^p \end{aligned}$$

In either case a is as close or closer to b as the origin.

Therefore in the octant above the line $y = x$ any point that is not one of the k points with smallest y-coordinate will be as close or closer to those k points than to the origin. A symmetric argument also proves that there are also at most k in the octant below the line $y = x$ points closer to the origin than to their k^{th} nearest neighbor and completes the proof of the lemma. □

3.3.2 Single-Hop All Points k-Nearest Neighbors

Our algorithm for k-Nearest Neighbors on a single-hop sensor network will be a very similar divide-and-conquer algorithm as the serial algorithm. The first difference is that we need a different technique to sort the set of sensors by y-coordinate at each recursive call because in this low-energy sensor network model it seems unlikely for two

lists to be able to be merged with $O(1)$ energy per sensor in the worst case. We instead sort once before making recursive calls and split this list into two smaller sorted lists using a serial scan before each recursive call. Then we find the median x-coordinate by either using the selection algorithm used in [49] or more simply by maintaining the points in sorted x order in addition to sorted y order. Then each sensor broadcasts in the sorted order with the next sensor in the sorted order listening. What each sensor broadcasts is the running sum of the number of sensors in L. With this information each sensor will know its position in sorted order for the next recursive call.

Now we iterate through each sensor first in ascending y-coordinate then by descending y-coordinate. Each sensor will listen to the previous sensor broadcast then execute the iterative step from the serial algorithm and then broadcast each point in L_{help} and R_{help} and for each point will broadcast a running sum of the number of possible k-nearest neighbors already found.

We then iterate through the sensors once again. This time the sensors that removed the sensor from L_{help} or R_{help} in each previous iteration take turns broadcasting the total number of potential k-nearest neighbors found. Each sensor that was a potential nearest neighbor listens to these leaders broadcast. Now for each sensor each potential nearest neighbor

Finally, for each sensor s the set of potential k-nearest neighbors use the selection algorithm in [49] to determine which points are actually the k-nearest neighbors. During this algorithm s will act in place of the k points from the previous recursive call. To get the information of s's k nearest neighbors to s, the k-nearest neighbors broadcast one a time with s listening. As one final complication, each sensor must know when it must wake up to begin the selection algorithm. Since the selection algorithm is deterministic we can keep a running total of the time that these tasks are going to take when broadcasting the total number of potential k-nearest neighbors.

Theorem 15. *In a single-hop sensor network of n sensors with one 2-dimensional point per sensor, each sensor can determine the k nearest points in $O(kn \log n)$ time and $O(k \log n)$ maximum energy used by any sensor. These bounds are deterministic if the sensors are already in some ordering. If the sensors are not ordered and even if n is not known then these bounds can be met in expectation.*

Proof. First, if the sensors are already ordered in some way, then sorting can be done deterministically in $O(n \log n)$ time with $O(\log n)$ maximum energy per sensor [49]. However, if the sensors are not ordered and even if n is not known, we can simply use the sorting algorithm from Section 2.5 in expected $O(n)$ time with expected $O(\log n)$ maximum energy per sensor. This will be the only source of randomness in the rest of the algorithm so each sensor will know when to wake up after the recursive calls are complete.

The time at each recursive level is $O(kn)$ because selection algorithm in [49] takes linear time. This algorithm also takes $O(1)$ maximum energy for any sensor. Since each sensor is involved with $O(k)$ calls of this algorithm and one additional call where it acts as k sensors, during a single recursive call each sensor spends at most $O(k)$ energy during selection.

During each of the first two iterations each sensor broadcasts $O(k)$ information and listens to only one other sensor broadcast. In the third iteration each broadcast is constant size and each sensor broadcasts at most $O(k)$ times and listens to at most $O(k)$ broadcasts. In the remaining places a sensor is awake it is easy to see that the sensor only uses $O(k)$ energy. □

3.3.3 Multi-Hop All Points k-Nearest Neighbors

Finally, we have all the tools we need to solve the k-Nearest Neighbors on a multi-hop network of sensors. In this model we assume that each point actually represents the sensor's physical location and further we modify the problem so that each sensor is only required to find it's k-nearest neighbors if those neighbors within broadcast range. If fewer than k neighbors are within broadcast range then the sensor is required to determine all neighbors within broadcast range. We make this modification because if this is not done then, depending upon the structure of the graph, global communication may be necessary and it may be impossible to compute the k-nearest neighbors without a single sensor spending $O(n)$ energy. Figure 3.5 illustrates one such network.

Our algorithm in this model is simply a $\frac{1}{2}$-Consolidation Algorithm using the single-hop algorithm we previously described. Once the single-hop is complete for every region the sensor is involved in the sensor will know its k-nearest neighbors if the sensor has k neighbors within $\frac{1}{2}r_c$. During the consolidation phase each sensor sensor with k-nearest neighbors between $\frac{1}{2}r_c$ and r_c will be found. We split this consolidation phase into 3 steps where for each step regions wait until all regions that could interfere complete, not just intersecting regions, before moving on to the next step. To do this we simply have each of the less than k vertices that still need to broadcast one at a time during the first consolidation phase. Each sensor will listen to all time slices of the first consolidation step so they know which nearby sensors they may be a k-nearest neighbor for. Then, during the second consolidation phase each region does a single recursive level of the single-hop algorithm to determine the k-nearest neighbors for the $O(k)$ points it's sensors heard in the previous consolidation step. In the final consolidation step each sensor will broadcast for the points it may be

a k-nearest neighbor for and listens to each sensor that may be its k-nearest neighbor.

Theorem 16. *Let S be multi-hop sensor network with n sensors and a maximum of m sensors in any single-hop region, distributed on a two-dimensional plane, with a global coordinate system. All sensors in S can determine the k nearest neighbors within broadcast range in $O(km \log m)$ expected time with $O(k \log m)$ maximum energy usage in expectation.*

Theorem 17. *Let S be multi-hop sensor network with n sensors and a maximum of m sensors in any single-hop region, distributed on a two-dimensional plane, without a global coordinate system. All sensors in S can determine the k nearest neighbors within broadcast range in $O(km \log m + \log^2 n)$ expected time with $O(k \log m + \log n)$ average energy and $O(k \log m + \log^2 n)$ maximum energy usage in expectation.*

3.4 Coverage Boundary

In this section we present energy efficient algorithms for the *coverage boundary* problem. Discovering which processors lie near the outskirts of the network can be especially important in this sensor network model. The definition of boundary we use in this paper is the *coverage boundary* originally presented in [58]. There the authors presented an algorithm for determining which sensors lie on the coverage boundary using two techniques they call localized Voronoi diagram and neighbor embracing polygon. The authors argued that in some expected cases the number of neighbors each sensor interacts with may remain constant. However there are some cases where the number of neighbors each sensor interacts with may be unbounded, see Figure 3.10 for example. Our solution for this problem will have a bound on the expected maximum energy usage for any process regardless of the sensor's distribution.

Given some radius r_s, which in the sensor model typically represents the radius around which the processor can sensor relevant information from the environment, we define the *coverage area* of a point p to be the disk around p with radius r_s. This is denoted by

$$Cover(p) = \left\{q \in \mathbb{R}^2 : \|p - q\| \leq r_c\right\}$$

. We define the *coverage area* of a set of points S to be the union of the coverage areas of each point in the set. Using similar notation as before we write this

$$Cover(S) = \bigcup_{p \in S} Cover(p)$$

.

Then we use the topological definition of boundary when talking of the coverage boundary. This definition implies that a point is in the boundary if it is exactly r_s away from some node in S and all other node in S are at least r_s away. We denote this boundary

$$\delta Cover(p) = \left\{q \in \mathbb{R}^2 : \|p - q\| = r_c\right\}$$

$$\delta Cover(S) = \{q \in Cover(S) : \forall p \in S, \|p - q\| \geq r_c\}$$

We define a *boundary arc* of a node p to be a maximal set of points in $\delta Cover(p) \cap \delta Cover(S)$ that form a circular arc. That is, if $p + r_c \cdot (\sin(\theta), \cos(\theta)) \in \delta Cover(S)$ for all $\theta_1 \leq \theta \leq \theta_2$ but $p + r_c \cdot (\sin(\theta_1 - \epsilon), \cos(\theta_1 - \epsilon)) \notin \delta Cover(S)$ and $p + r_c \cdot (\sin(\theta_2 + \epsilon), \cos(\theta_2 + \epsilon)) \notin \delta Cover(S)$ for small $\epsilon > 0$ then $\{p + r_c \cdot (\sin(\theta), \cos(\theta)) : \theta_1 \leq \theta \leq \theta_2\}$ is a boundary arc of p. A node may have multiple boundary arcs and we denote the set of all boundary arcs for a node $p \in S$ to be $C\text{-}Arcs(p, S)$. Note then that

$$\bigcup C\text{-}Arcs(p, S) = \delta Cover(p) \cap \delta Cover(S)$$

Finally, we say that a node p in S is a *boundary node* if some point in $\delta Cover(p)$ is also in $\delta Cover(S)$. In other words $p \in BN$ if and only if p has at least one boundary arc. All other nodes in S are called *interior nodes*. These are denoted as following

$$BN(S) = \{p \in S : \delta Cover(p) \cap \delta Cover(S) \neq \emptyset\}$$

$$IN(S) = S \setminus BN(S)$$

The *coverage boundary* problem is then for each sensor to determine if it is a boundary node. This definition of boundary has the several advantages that make it a good definition. For one this definition captures coverage holes as well as the external boundary. Secondly for this definition is entirely local, meaning that whether or not a sensor is on the boundary only depends upon the position of other sensors within $2r_s$. However, if $r_s > \frac{r_c}{2}$ then computing the boundary may require global communication or if the network is disconnected may be impossible to compute. See Figure 3.7 for illustration. As previously pointed out requiring global communication can lead to bottle-necks where individual sensors are required to spend far too much energy. Therefore for our algorithm we require $r_s < \frac{r_c}{2}$. Fortunately most commercially available sensors meet these requirements. This makes computing the boundary in an energy efficient manner feasible with the techniques presented in this chapter.

The strategy for our coverage boundary algorithm is to use a $2r_c$-consolidation algorithm with single-hop algorithm using a divide-and-conquer approach. First we present the single-hop algorithm for the coverage boundary problem.

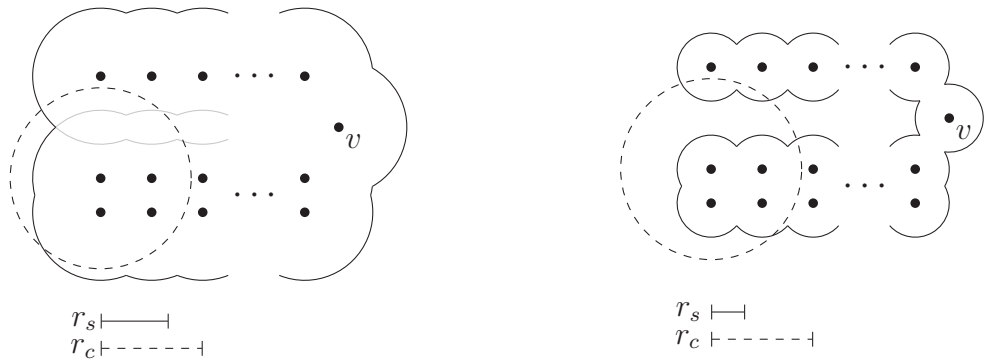


Figure 3.7: On the left, $r_s > \frac{r_c}{2}$ and in order to determine that the light grey arcs are not boundary arcs and, thus, determine $BN(S)$, sensor v must transfer information for every node in from the top of the network to the bottom of the network. On the right, $r_s < \frac{r_c}{2}$ so only local communication is needed to determine the boundary arcs and $BN(S)$.

3.4.1 Single-Hop Coverage Boundary

Theorem 18. *The coverage boundary of a set S of n 2-dimensional points can be computed on a single-hop broadcast network in $O(n \log n)$ expected time with maximum energy usage $O(\log^2(n))$ in expectation for any processor.*

This result directly follows from Theorem 27 presented later in Section 3.5. This is because a point $v \in S$ is in the coverage boundary of S if and only if there exists a point on one of v's Voronoi edges that is within r_s of v. See Figure 3.8 for illustration of this fact and see Section 3.5 for clarification on the Voronoi diagram. However in this section we will present a simpler algorithm with better coefficients that also meets these bounds.

The single-hop algorithm for coverage boundary presented here follows the same basic pattern as the single-hop 2-dimensional k-nearest neighbor algorithm. First we pre-sort the set of nodes by x and y-coordinate so each sensor knows it's rank in each dimension. Then, based upon the median x-coordinate m, S is divided into left

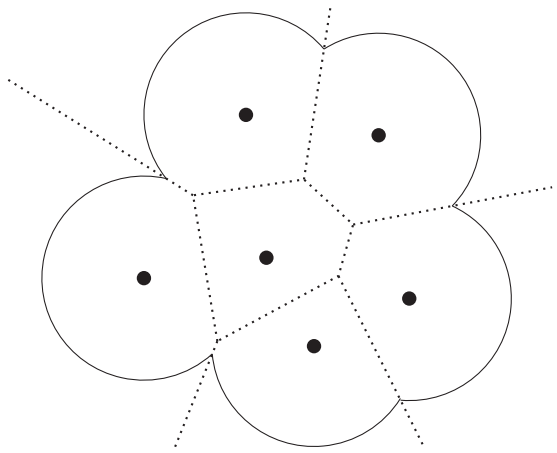


Figure 3.8: Each point p is in the coverage boundary if and only if p's Voronoi face is not completely contained in the r_s ball around v. Also the intersecting arcs of the coverage boundary all lie on the Voronoi edge of the two equidistant points.

CoverageBoundary(P) =

1: divide P by the median x-coordinate m into left points L and right points R
2: CoverageBoundary(L)
3: CoverageBoundary(R)
4: Construct interval tree of arcs from $\delta Cover(L)$ to the right of m
5: Sweep down the tree to determine where $\delta Cover(L)$ and $\delta Cover(R)$ intersect to the right of m
6: Sweep up the tree and use a scan to determine the new arcs in $\delta Cover(S)$ from $\delta Cover(R)$ that are to the right of m
7: Use a scan to determine the new arcs in $\delta Cover(S)$ from $\delta Cover(L)$ that are to the right of m
8: Repeat steps $4-7$ replacing L and R and "left" and "right" respectively
9: Redistribute arcs so that each processor has a constant number of arcs

Figure 3.9: Structure of the single-hop coverage boundary algorithm.

and right halves which we call L and R respectively. Then we recursively find the boundary arcs of L and R. Then using interval trees and serial scans we determine which boundary arcs must be truncated or removed entirely. Figure 3.9 gives a more detailed overview of the algorithm.

We maintain the following invariants for each recursive call:

- Each boundary arc is maintained by a single processor and each processor main-

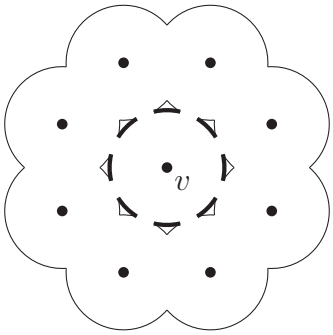


Figure 3.10: Sensor v has a boundary arc for every other sensor in the network.

tains $O(1)$ boundary arcs.

- In addition to maintaining each node's y-coordinate rank, each node's boundary arcs maintain their ranks in a rotational ordering.

Each node may have up to $\Theta(n)$ boundary arcs so it is not possible for each processor to maintain all of its boundary arcs in an energy efficient manner. See Figure 3.10 for illustration. However Lemma 19 implies that it is possible to reorder the boundary arcs so that each processor only maintains $O(1)$ boundary arcs.

Lemma 19. *For a set S of n 2-dimensional points $\delta\text{Cover}(S)$ contains $O(n)$ boundary arcs.*

Proof. The intersection of two boundary arcs must be exactly r_C away from two points $u, v \in P$ and every other point in P must be more than r_C away. Therefore this intersection lies on the Voronoi edges separating the Voronoi faces of u and v. See Figure 3.8 for an illustration of this fact and see Section 3.5 for further discussion of the Voronoi diagram. Further there may only be at most two intersections of boundary arcs lying on the Voronoi edge because there are only two points on the Voronoi edge that are exactly r_C away from u and v. Since the number of Voronoi edges of $\text{Vor}(P)$ is at most $3n-6$ [44], the total number of boundary arc intersections is $O(n)$ and finally the total number of boundary arcs is also $O(n)$. □

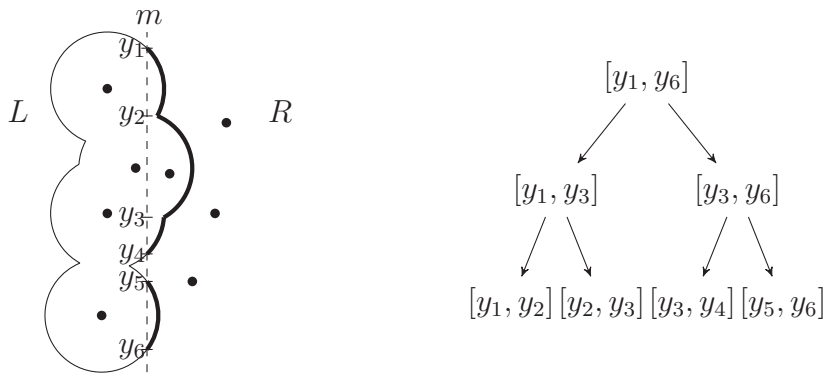


Figure 3.11: On the left, $\delta Cover(L)$ with shoreline bolded. On the right, corresponding interval tree of the shoreline of L.

Once the boundary arcs of L and R are found recursively we determine which arcs must be truncated and which arcs must be removed completely. To do this we determine each point where $\delta Cover(L)$ intersects $\delta Cover(R)$. First we determine where $\delta Cover(L)$ intersects $\delta Cover(R)$ to the right of the median x-coordinate m and then where they intersect to the left of m. For sake of simplicity we will only describe how to find the intersections to the right of m.

We call the points of $\delta Cover(L)$ with x-coordinate greater than m to be the *shoreline* of L. To determine where the boundary arcs of L and boundary arcs of R intersect to the right of m we will construct an interval tree of the shoreline of L. The leaves of our tree contains information for a single boundary arc from L's shoreline. These boundary arcs are keyed by their maximum and minimum y-coordinates. Lemma 20 implies that the y-coordinates of these arcs do not overlap. Each non-leaf node in the tree contains, as the key, the maximum and minimum y-coordinate of each boundary arc in the subtree rooted at that node. See Figure 3.11 for an example beachline and corresponding interval tree. Along with the minimum and maximum y-coordinates we also maintain the minimum and maximum x-coordinate of any point on an arc in the subtree.

Lemma 20. *The shoreline of L has at most a single point with any given y-coordinate.*

Proof. Suppose, by way of contradiction, that $a, b \in \delta Cover(L)$ such that $a_y = b_y$ and without loss of generality $m < a_x < b_x$. There must be some point $p \in L$ such that $\|b - p\| = r_s$. Then

$$\begin{aligned}
\|a - p\| &= \sqrt{(a_x - p_x)^2 + (a_y - q_y)^2} \\
&= \sqrt{(b_x - p_x - (b_x - a_x))^2 + (b_y - p_y)^2} \\
&< \sqrt{(b_x - p_x)^2 + (b_y - p_y)^2} \\
&= r_s
\end{aligned}$$

This contradicts a being in $\delta Cover(L)$. □

In order to construct the interval tree first each arc in the shoreline must know it's y-coordinate rank to form the leaves of the interval tree. Lemma 21 implies that these ranks can be determined with a simple scan using each sensor's y-coordinate and the arc's ranks in the rotational ordering. Then deterministically based upon the rank of the leaf each sensor picks a non-leaf node in the interval tree to maintain. Then we perform a sweep up the tree. Each leaf node broadcast in order and each parent listens to each of its children broadcast. We then repeat this for the nodes in the second level and so on, until we reach the top of the tree.

Lemma 21. *Let $p, q \in L$ such that $p_y > q_y$, then $u_y \geq v_y$ for every point u on a boundary arc of p and every point v on a boundary arc of q that each lie on shoreline of L.*

Proof. Suppose, by way of contradiction, that $u_y < v_y$. If $p_x \leq q_x$ then

$$\|q - u\| = \sqrt{(q_x - u_x)^2 + (q_y - u_y)^2}$$

$$= \sqrt{(p_x - u_x - (p_x - q_x))^2 + (p_y - u_y - (p_y - q_y))^2}$$
$$< \sqrt{(p_x - u_x)^2 + (p_y - u_y)^2}$$
$$= r_s$$

This contradicts u lying on a boundary arc. In the other case if $q_x \leq p_x$ the same argument shows that $\|p - u\| < r_s$, contradicting v lying on a boundary arc. □

Once the tree is constructed we now perform a sweep down the interval tree to determine each intersection of $\delta Cover(L)$ and $\delta Cover(R)$ to the right of m. To sweep down the tree first the root broadcasts, then each node in the second level will broadcast, and so on until we reach the leaf nodes. For each boundary arc $\hat{a}$ in $\delta Cover(R)$ the sensor maintaining the arc listens to the root broadcast and then continues down the tree for each branch for which it is possible for $\hat{a}$ to intersect. When we say $\hat{a}$ possibly intersects with an arc in the subtree we mean that some point in $\hat{a}$ is within the minimum and maximum y and x values stored in the root of the subtree.

If each processor with an arc $\hat{a}$ in $\delta Cover(R)$ to the right of m listens to all nodes in the tree where its arcs may intersect then some processors may need to listen to $\Theta(n)$ nodes which will excessive amounts of energy. To remedy this issue each processor will only keep track of the subtrees it possibly intersects with highest and lowest y-coordinate in the interval tree. For every other node in the tree the processors maintaining the tree will be responsible for determining where $\hat{a}$ intersects. This is done by allocating time slots after each node broadcasts for sensors responsible for arcs such as $\hat{a}$ to broadcast.

For example, when the original processor keeping track of $\hat{a}$ could possibly intersect two children of the subtree it is maintaining with highest y-coordinate it continues

down the subtree of the child with higher y-coordinate and broadcasts during one of these time slots after the child with lower y-coordinate broadcasts. Then that child may broadcast during these time slots after its children broadcast depending upon whether $\hat{a}$ could possibly intersect the arcs in those subtrees. This continues until either we reach a leaf node or it is determined that it is not possible for $\hat{a}$ to intersect any arcs in the childrens' subtrees.

Lemma 22 implies that only a constant number of arcs will need to be passed to each node in the tree. Additionally, which time each possibly intersecting arc is needed to be broadcast without radio interference can be deterministically decided simply using the positioning of the arc. This means that each intersection of $\delta Cover(L)$ and $\delta Cover(R)$ to the right of m can be determined in $O(n)$ time and $O(\log n)$ maximum energy used by any processor.

Lemma 22. *Let T be the subset of $\delta\text{Cover}(L)$ with x-coordinate at least m and y-coordinate between y_{min} and y_{max}. Also let x_{min} and x_{max} be the minimum and maximum x-coordinate of T respectively. Then no more than 4 boundary arcs in $\delta\text{Cover}(R)$ have maximum y-coordinate greater than y_{max}, minimum y-coordinate less than y_{min}, and have some point with x-coordinate between x_{min} and x_{max}. In other words no more than 4 arcs in $\delta\text{Cover}(R)$ go from above to below the bounding box of T and intersect the bounding box.*

Proof. We will prove this lemma by observing the center of the arcs in $\delta Cover(R)$ that meet the criteria of the theorem. We will show that only one center can be to the left of the bounding box, one center can be to the right of the bounding box, and at most two centers can be neither to the left or right of the bounding box.

Suppose, for the sake of contradiction, that there are two boundary arcs meeting the criteria of the theorem whose centers u and v have $u_x \leq x_{\min}$ and $v_x \leq x_{\min}$.

First notice that these arcs cannot intersect for any y-coordinate between $y_{\min}$ and $y_{\max}$. Without loss of generality, let the arc of u have lower x-coordinate for each y-coordinate between $y_{\min}$ and $y_{\max}$. Let p be on the arc of u such that $p_x \geq x_{\min}$ and let q be on the arc of v such that $p_y = q_y$. Then

$$\begin{aligned}
\|v - p\| &= \sqrt{(v_x - p_x)^2 + (v_y - p_y)^2} \\
&= \sqrt{(v_x - q_x - (p_x - q_x))^2 + (v_y - q_y)^2} \\
&< \sqrt{(v_x - q_x)^2 + (v_y - q_y)^2} \\
&= r_s
\end{aligned}$$

This contradicts p being on a boundary arc. Additionally, a symmetric argument holds if two boundary arcs meet the criteria of the theorem and have centers to the right of $x_{\max}$.

Now suppose that there are three boundary arcs whose centers u, v, and w are not to the left or right of the bounding box, or in other words $x_{\min} \leq u_x < v_x < w_x \leq x_{\max}$. Notice that if L contains a single point then $y_{\max} - y_{\min} = \sqrt{{r_s}^2 - (r_s - x_{\max} + x_{\min})^2}$ and adding points to L can only increase $y_{\max} - y_{\min}$ while maintaining $x_{\max} - x_{\min}$ constant. Therefore $x_{\max} - x_{\min} \leq \sqrt{{r_s}^2 - (r_s - y_{\max} + y_{\min})^2}$ if $y_{\max} - y_{\min} < r_s$ and otherwise $x_{\max} - x_{\min} \leq r_s$. Notice also that $|v_y - y_{\max}| < r_s$ and $|v_y - y_{\min}| < r_s$ so there must be some point p on the boundary arc of v with $y_{\min} \leq p_y \leq y_{\max}$ and $|v_x - p_x| \geq \sqrt{{r_s}^2 - (r_s - y_{\max} + y_{\min})^2} \geq x_{\max} - x_{\min}$. This implies that the x-coordinate of some point q on the boundary arc of v and inside the bounding box is either u_x or w_x. Without loss of generality let's say that $q_x = u_x$. Then $\|u - q\| \leq r_s$ because $|u_y - y_{\max}| < r_s$ and $|u_y - y_{\min}| < r_s$. This contradicts q being on a boundary arc and completes our proof. □

Now that each intersection between $\delta Cover(L)$ and $\delta Cover(R)$ to the right of m has been discovered, we need to next fix the boundary arcs from $\delta Cover(R)$. To do this we perform a modified predecessor search on the interval tree. First we perform a sweep up the interval tree from the leaves to the root where each node in the tree broadcasts for each arc that it was responsible for during the sweep down the tree. Each node broadcasts the total number of times that arc intersects with the arcs in the subtree and the intersections with minimum and maximum y-coordinate. Each node then possibly constructs the arcs based upon the intersections with maximum y-coordinate from the child with smaller y-coordinate and the intersections with minimum y-coordinate from the child with larger y-coordinate. Additionally, the original sensor that was keeping track of the arc listen to each of the nodes that it transmitted to in the original sweep to construct the first and last new arc.

Once these arcs are constructed their ranks can be determined using a sweep up the tree, a serial scan, and a sweep down the tree. During the sweep up the tree we count the total number of new arcs an old boundary arc is broken into. Then during the scan each old arc announces the uses this new number of arcs to compute the new ranks of the first and last new arc. Then these values are passed down the tree so that each new arc also knows its rank.

Now we need to fix the boundary arcs from $\delta Cover(R)$. We simply sort the intersections by y-coordinate. To do this we simply need to order the intersections of $\delta Cover(L)$ and $\delta Cover(R)$ by y-coordinate. If it were true that the y-coordinate of each point in R corresponding to each intersection point was monotonically increases with the y-coordinate of the intersection point itself, then this ordering could be determined by a simple scan using the ranks from the invariant.

This completes our single-hop, $O(n \log n)$ time algorithm to determine the coverage boundary with at most $O(\log^2)$ energy for any sensor and the proof of Theorem 18.

3.4.2 Multi-Hop Coverage Boundary

Now we have the tools necessary to create a multi-hop coverage boundary algorithm.

Theorem 23. *Let S be multi-hop sensor network with n sensors and a maximum of m sensors in any single-hop region, distributed on a two-dimensional plane, with a global coordinate system. All sensors in S can determine if they lie on the coverage boundary in $O\left(\frac{m \log m}{\left(\frac{1}{2}-\frac{r_s}{r_c}\right)^2}\right)$ expected time with $O\left(\frac{\log^2 m}{\left(\frac{1}{2}-\frac{r_s}{r_c}\right)^2}\right)$ maximum energy usage in expectation.*

Theorem 24. *Let S be multi-hop sensor network with n sensors and a maximum of m sensors in any single-hop region, distributed on a two-dimensional plane, without a global coordinate system. All sensors in S can determine if they lie on the coverage boundary in $O\left(\frac{m \log m+\log^2 n}{\left(\frac{1}{4}-\frac{r_s}{r_c}\right)^2}\right)$ expected time with $O\left(\frac{\log^2 m+\log n}{\left(\frac{1}{4}-\frac{r_s}{r_c}\right)^2}\right)$ average energy and $O\left(\frac{\log^2 n}{\left(\frac{1}{4}-\frac{r_s}{r_c}\right)^2}\right)$ maximum energy usage in expectation.*

Our multi-hop approach to this problem uses a consolidation algorithm. If a global coordinate system is known we use a $1-\epsilon$-consolidation algorithm where $\epsilon = \frac{1}{2} - \frac{r_s}{r_c}$ and if a global coordinate system is not known we use a $\frac{1}{2}-\epsilon$-consolidation algorithm where $\epsilon = \frac{1}{2} - \frac{2r_s}{r_c}$. First notice that this requires $\epsilon > 0$ and therefore $r_s < \frac{r_c}{2}$ if there is a known coordinate system and $r_s < \frac{r_c}{4}$ otherwise. The main idea behind this consolidation algorithm is that after the single-hop algorithm some, but not necessarily all, boundary arcs computed will *provide witness* for a sensor being in the coverage boundary.

A boundary arc computed by an execution of the single-hop algorithm will provide witness if all possible sensors within r_s of some point on the arc participated in the local algorithm. If the global coverage boundary of the network was computed at

least some this point would still be on a boundary arc, implying that the sensor is indeed on the coverage boundary for the global network. One small complication is that the sensor may not know all of its boundary arcs after the execution of the single-hop algorithm. Therefore, as a final step to the single-hop algorithm, each sensor will be allocated a time step for sensors with a boundary arc providing witness will broadcast.

First consider the case where a global coordinate system is known beforehand and $r_s < \frac{r_c}{2}$. Because of the method chosen for tiling disks in Section 3.2.1 every point is within $\epsilon \cdot r_c = \frac{r_c}{2} - r_s$ of the center of some $\frac{r_c}{2}$-disk. That implies that during the execution of the single-hop algorithm represented by this disk every sensor within r_s of this point participates in this execution. Thus to check if an arc is a witness we just need to check if some point on this arc is within $\epsilon \cdot r_c$ of the center of the disk. Additionally, if a sensor is a boundary node a boundary arc of some local execution will be a witness. This completes the proof of Theorem 23.

In the case where no known coordinate system and $r_s < \frac{r_c}{4}$, each sensor will for some execution of the single-hop algorithm be within ϵ of the some leader at the center of the single-hop algorithm. Let u be some sensor within ϵ of the center, c, and let v be some sensor within r_c of a boundary arc of u. Then

$$\begin{aligned} \|v - c\| &\leq \|v - u\| + \|u - c\| \\ &= 2r_s + \left(\frac{r_c}{2} - 2r_s\right) \\ &= \frac{r_c}{2} \end{aligned}$$

This implies v participates in the single-hop algorithm centered at c and any boundary arc of u found during this execution will provide witness for u being in the boundary. This completes the proof of Theorem 24.

3.5 Localized Voronoi Diagram

The Voronoi diagram is an important geometric structure in computational geometry and closely relates to many proximity and covering problems. The Voronoi diagram has many applications relevant to sensor networks, such as in [27] the Voronoi diagram is used in 3 dimensions to determine underwater sensors sleep schedules while maintaining coverage. Distributed algorithms to compute the Voronoi diagram are given in [6, 21]. In [6] the authors use Dulaunay triangulation to solve the exact Voronoi diagram. However in this algorithm each sensor communicates with all of it's neighbors and in cases such as Figure 3.5 these sensors may need to communicate with their neighbors many times. In [21] the authors compute the localized Voronoi diagram using a technique similar to Graham's scan for computing the convex hull so that each sensor may not need to interact with each neighboring sensor. However as in Figure 3.12 the number of neighbors some sensors interact with may still be quite large. We improve on these results for dense sensor networks by providing an algorithm for computing the localized Voronoi diagram with an energy usage bound for each sensor that does not depend upon the number of neighbors.

Given two sensors u and v we define $\mathrm{Dom}(u, v)$ to be the set of points as close or closer to u than to v using Euclidean distance. $\mathrm{Dom}(u, v)$ is the half plane bounded by the perpendicular bisector of u and v containing u.

$$\mathrm{Dom}(u, v) = \{x \in \mathbb{R}^2 : \|x - u\|_2 \leq \|x - v\|_2\}$$

Then we define u's *Voronoi polygon* or *Voronoi face* to be the set of points as close

or closer to u as any other point in S.

$$\mathrm{Vor}(u) = \bigcap_{v \in S \setminus \{u\}} \mathrm{Dom}(u, v)$$

The *Voronoi edge* of u and v is the points on the perpendicular bisector of u and v that are as close or closer to u and v than any other points in S, or in other words, the Voronoi edge of u and v is $\mathrm{Vor}(u) \cap \mathrm{Vor}(v)$. Similarly, the *Voronoi vertex* of u, v, and w to be the point equidistant to u, v, and w if no points in S. Note that a Voronoi vertex lies at the intersection of three Voronoi edges. Then the *Voronoi diagram* of S $\mathrm{Vor}(S)$ is the partitioning of the plane into the respective Voronoi faces of the sensors in S.

For some networks computing the exact Voronoi diagram may require information to be transferred globally, see Figure 3.5. This may require a single critical sensor to transmit large amounts of information. Even more extreme if the graph is not connected it is impossible to compute the exact Voronoi diagram. To this end we instead compute the *localized Voronoi diagram* defined in [58] and [21]. We define an α-localized Voronoi face of u to be the set of points in the Voronoi face of u that are also within αr_c of u.

$$\alpha\text{-LVor}(u) = \mathrm{Vor}(u) \cap \{x \in \mathbb{R}^2 : \|x - u\|_2 \leq \alpha r_c\}$$

Then the α-localized Voronoi edge and α-localized Voronoi vertex are defined similarly in the obvious way. The α-localized Voronoi diagram α-LVor(S) is simply the set containing all the α-localized Voronoi faces. This is no longer a partition because there are inevitably some points in the plane that are not within r_c of any sensor.

Another problem that may arise when computing the Voronoi diagram is that if

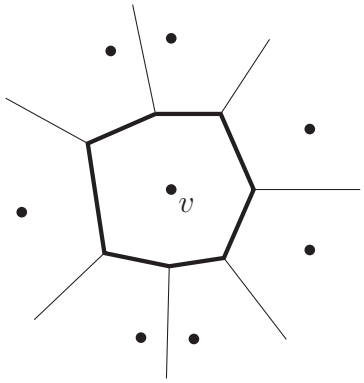


Figure 3.12: Sensor v has a Voronoi edge for every other sensor in the network.

we require every sensor to know all of its Voronoi edges then the amount of energy dissipation may be linear for some sensors because the number of Voronoi edges they have may be linear in the number of sensors, see Figure 3.12. On the other hand if we only require that some sensor in the network knows of the Voronoi edge then this information may be useless as it may be too far from the actual sensors that form the edge. To these ends we require that for each Voronoi edge at least one of the two sensors that form the edge know of the edge. Even though each sensor may may require information from its neighbors that it shares a Voronoi edge with to determine its exact Voronoi face this will be sufficient for some applications. With this requirement we are still able to distribute the Voronoi edges such that each sensor only maintains $O(1)$ Voronoi edges.

We now present our algorithm for computing the localized Voronoi diagram either with or without a known global coordinate system.

Theorem 25. *Let S be multi-hop sensor network with n sensors and a maximum of m sensors in any single-hop region, distributed on a two-dimensional plane, with a global coordinate system. The $\frac{1}{2} - \epsilon$-localized Voronoi diagram can be computed in $O\left(\frac{m \log m}{\epsilon^2}\right)$ expected time with $O\left(\frac{\log^2 m}{\epsilon^2}\right)$ maximum energy usage in expectation.*

Theorem 26. *Let S be multi-hop sensor network with n sensors and a maximum of m sensors in any single-hop region, distributed on a two-dimensional plane, without a global coordinate system. The $\frac{1}{4} - \epsilon$-localized Voronoi diagram can be computed in $O\left(\frac{m \log m + \log^2 n}{\epsilon^2}\right)$ expected time with $O\left(\frac{\log^2 m + \log n}{\epsilon^2}\right)$ average energy and $O\left(\frac{\log^2 n}{\epsilon^2}\right)$ maximum energy usage in expectation.*

In both cases with a known or no known global coordinate system we use a 2α-consolidation algorithm. Note that an α-Voronoi diagram is computed using a 2α-consolidation algorithm because to guarantee that a sensor is closest to a point that is within αr_c of the sensor it must take into account all sensors within $2\alpha r_c$. To create such a consolidation algorithm we need to define is a single-hop algorithm and a single-hop consolidation step.

Theorem 27. *Given a single-hop sensor network with n sensors each with a 2-dimensional point, the Voronoi diagram of these points can be computed in $O(n \log n)$ expected time with $O(\log^2 n)$ maximum energy used per sensor in expectation.*

3.5.1 Single-Hop Voronoi Diagram

Our single-hop algorithm is based on a PRAM algorithm presented in [16] which corrected an algorithm presented in [13]. Both of which follow the classical divide-and-conquer approach due to Shamos [44]. Due to randomness used in our algorithm we will use the breadth-first recursion technique from Section 2.4 to abstract away the difficulties of executing randomized recursive algorithms in this energy restricted sensor network model.

During the single-hop algorithm each sensor will be assigned a constant number of Voronoi edges to maintain. This is possible because the number of Voronoi edges is at most $3n - 6$ [44]. However during the single-hop algorithm each Voronoi edge

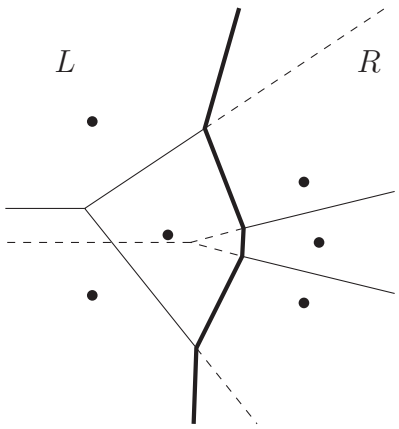


Figure 3.13: The dark line is the contour between L and R. The dashed lines are the Voronoi edges of L and R that are cut by the contour. The solid lines are the Voronoi edges from L and R that remain in $L \cup R$.

will not necessarily be assigned to one of the two sensors that for the edge. This requirement will be taken care of once at the end of the single-hop algorithm.

This approach divides the set of all points S into two halves L and R usually by x coordinate and recursively finds the Voronoi diagram for L and R. These solutions are then merged by finding the Vornoi edges between L and R–called the contour–and trimming or removing Voronoi edges that are cut by the contour (see Figure 3.13 for example).

An overview of a single recursive step can be found below:

1. For each Voronoi vertex determine if it is closer to a point in L or a point in R and for each infinite Vornoi edge determine determine as the edge goes to infinity if it is closer to a point in L or a point in R, in the limit.

2. Using this information for each Voronoi edge determine if it is intersected by the contour. Doing so we determine if the Voronoi edge remains in the Voronoi diagram, if it must be trimmed, or if it must be completely removed.

3. Find the order that the faces of Vor(L) and Vor(R) are intersected by the

contour, using list ranking.

4. Find the new endpoints for each Voronoi edge that must be trimmed and create the contour.

Determining which Voronoi Edges are Intersected by the Contour

For each Voronoi vertex we are to determine if it is closer to the nearest point in L or the nearest point in R and for each infinite Voronoi edge we determine if as the edge goes to infinity if it is closer to the nearest point in L or the nearest point in R, in the limit.

For the infinite edges we simply compare the slope of the line to the slope of the perpendicular bisector of the line tangent to the convex hulls of L and R. This is because the perpendicular bisector lies on the infinite Voronoi edges of the contour. See Figure 3.14 for illustration. The tangent line of the convex hulls of L and R can be computed in $O(\log n)$ time by either maintaining the convex hull for each recursive call and using a binary search or using the randomized algorithm in Section 2.6.

To determine if each endpoint p of a Voronoi edge, without loss of generality, from $\text{Vor}(L)$ is closer to L or to R we simply need to use $\text{Vor}(R)$ to find whether the closest point to p in R. We do this using Kirkpatrick's triangle refinement technique [31].

Lemma 28. *Given a single-hop sensor network with n sensors and a set of $O(n)$ query points, A, and the Voronoi diagram of a set of $O(n)$ points, B, distributed so that each sensor has $O(1)$ query points and Voronoi edges. For each query point in A, the closest point in B can be determined in $O(n)$ expected time and $O(\log n)$ maximum energy per sensor in expectation.*

Proof. We begin by bounding the Voronoi diagram by a triangle. Then we triangulate the Voronoi faces, for each Voronoi face we pick a single Voronoi vertex in that face

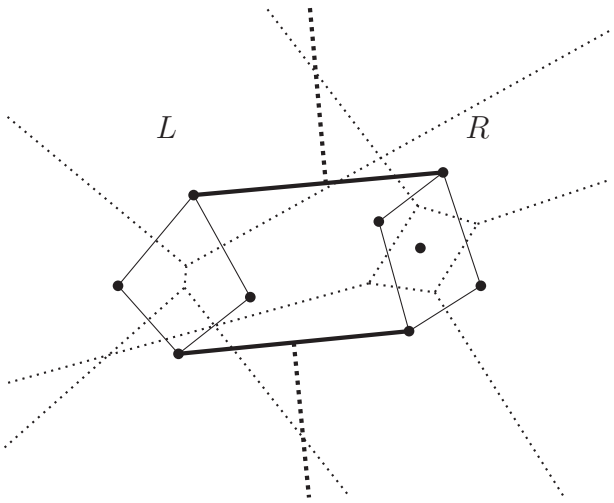


Figure 3.14: The solid lines are the convex hull of L and R with the dark solid lines being the tangent lines between the convex hulls. The dotted lines are the Voronoi edges of L and R and the dark dotted lines are the perpendicular bisectors of the tangent lines between the convex hulls of L and R.

and add an edge from every other vertex on the face to that vertex. Now we remove an independent set of Voronoi vertices and retriangulate. We must pick vertices that form an independent set and each vertex's degree must be bounded by a constant. These triangles created by this process form the vertices in a directed acyclic search graph, where each triangle points to the triangles that it intersects with in previous triangulation. We continue removing sets of independent vertices until every vertex has been removed. See Figure 3.15 for an example of this process and Figure 3.16 for the corresponding directed acyclic search graph.

A bounding triangle can easily be found and broadcast in $O(\log n)$ expected time three runs of the maximum finding algorithm from [38]. First for every Voronoi vertex we elect one of the sensors with an edge incident that vertex to maintain that vertex. Then we can triangulate the Voronoi faces by ordering the faces and for each electing a Voronoi vertex and adding an edge connected to every other Voronoi vertex on the face. Now we can order the vertices and for each vertex we can order the edges

incident to this vertex in rotational order, say clockwise starting from 12 o'clock. This can all be done in $O(n)$ expected time and with at most $O(\log n)$ energy per sensor in expectation using the sorting algorithm from Section 2.5.

Now can determine which vertices to remove and construct the directed acyclic search graph just using a few scans. While doing so we maintain the vertex ordering and the rotational ordering on the edges incident to each vertex. First we determine the number of edges incident to each vertex. Then we determine which vertices to remove by iterating through each vertex having any edge incident to the vertex broadcast if the other endpoint has been selected and if not selecting the vertex if it's degree is bounded by a predetermined constant. For each vertex v that was removed we replace each edge incident to that vertex to an edge incident to an arbitrary vertex u that was incident to v–except of course u and the vertices already adjacent to u which can be determined because of the rotation ordering of the edges. Then we perform a scan to maintain the number of edges incident to each vertex and the rotational ordering of these edges for the new triangulation. One final scan can then determine the order in which the faces will broadcast in the directed acyclic search graph. Since these are all just scan operations they take $O(n)$ time and $O(1)$ energy per sensor.

Since the number of vertices is decreasing geometrically the number of rounds is $O(\log n)$ and the total time for all rounds is $O(n)$. Then we can finally determine for each query point which Voronoi face the point lies in using the directed acyclic search graph, in $O(n)$ time with $O(\log n)$ energy per sensor. □

Now we describe how to determine if a Voronoi edge is intersected by the contour and if the edge must be trimmed or removed. We only consider the finite case where our Voronoi edge has two endpoints but the infinite case is handled similarly. Let $\overline{pq}$

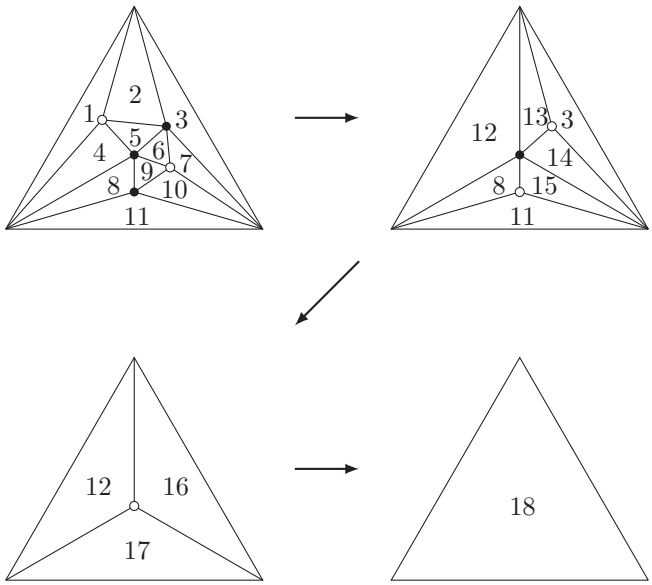


Figure 3.15: Kirkpatrick's triangule refinement steps, removing points and retriangulating.

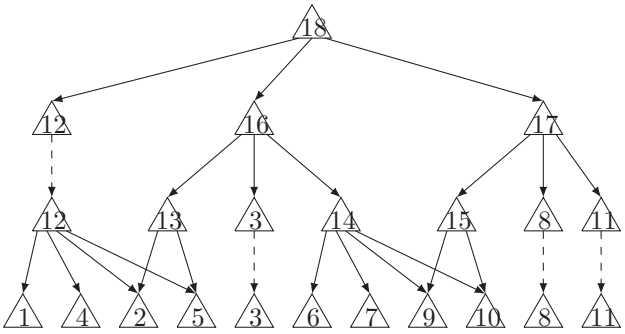


Figure 3.16: Directed acyclic search graph formed by the refinements in Figure 3.15.

be a Voronoi edge in Vor(L) with $a \in L$ being the point closest to $\overline{pq}$ with largest x coordinate. The following cases exhaust all possibilities for us to consider:

- If both p and q are closer to L than R, then $\overline{pq}$ lies completely to the left of the contour and $\overline{pq}$ remains a Voronoi edge in the new Voronoi diagram.
- If both p and q are closer to R than L, then:
  - If $\overline{pq}$ intersects the horizontal line through a, then the contour intersects

$\overline{pq}$ exactly twice and $\overline{pq}$ is replaced by $\overline{p'q'}$ in the new Voronoi diagram for some points p' and q' on $\overline{pq}$.

- Otherwise if $\overline{pq}$ does not intersect the horizontal line through a, then $\overline{pq}$ lies completely to the right of the contour and $\overline{pq}$ is completely removed from the new Voronoi diagram.

- If, without loss of generality, p is closer to L and q is closer to R, then the contour intersects $\overline{pq}$ exactly once and $\overline{pq}$ is replaced by $\overline{pq'}$ in the new Voronoi edge for some point q' on $\overline{pq}$.

Most of these cases are straightforward and a more detailed analysis of these cases can be found in [16]. However, we will briefly explain the cases where p and q are closer to R and L. If $\overline{pq}$ does not intersect the horizontal line through a then all points in $\overline{pq}$ must either be above a or must be below a. Without loss of generality we assume $\overline{pq}$ lies above a and that p has smaller y-coordinate than q. Then any point in R that is closer to p than a is must also be closer to any other point on $\overline{pq}$ than a is. If however $\overline{pq}$ intersects the horizontal line through a then this point is closer to a than any point in R. Thus $\overline{pq}$ must intersect the contour at least twice. Additionally, the contour cannot intersect $\overline{pq}$ more than twice because, as before, if x lies on $\overline{pq}$ above a and is closer to a point in R than to a then any point above x that lies on $\overline{pq}$ is also closer to some point in R than to a.

Finding new Voronoi edges

We are now going list subsections of the faces of $\mathrm{Vor}(L)$ in order that they are intersected by the contour from bottom to top and do the same for $\mathrm{Vor}(R)$ respectively. As in [3] we define a *conduit* as a triangular subsection of Voronoi face with one apex being a point in L and the other two apexes being the endpoints of a Voronoi edge

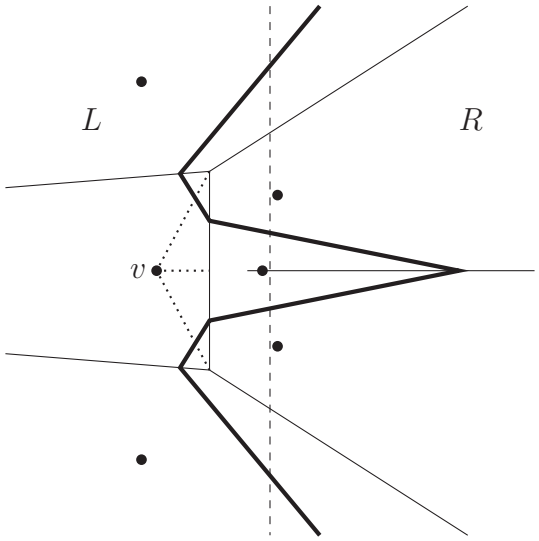


Figure 3.17: The Vornoi edge to the right of v is intersected twice and so the conduit of v on the right is divided into two smaller conduits denoted by the dotted lines.

of the first apex. Each conduit is intersected by the contour at most twice and in the case that it is intersected twice we further subdivide the conduit into two smaller conduits. We divide the conduit into two triangles using the point which the horizontal line through a intersects $\overline{pq}$ as the third apex point for each triangle. Each conduit is only intersected twice if and only if it's Voronoi edge $\overline{pq}$ is intersected twice by the contour so all remaining conduits and the semi-conduits are intersected at most once. See Figure 3.17 for example.

For the edges that are intersected by the contour we know that their conduit is also intersected by the contour. However for the edges that lie completely to the right of the contour we do not now if the edge's conduit is intersected by the contour. For each conduit we will determine if it intersected by the contour and if it is what the previous conduit and next conduit intersected by the contour. As pointed out in [16] the contours of a given face are intersected by the contour in counter-clockwise order beginning directly to the left of the point in L. Therefore if we maintain the edges in this ordering at each recursive level we can do a simple scan to determine for each

contour if it is intersected by the contour and what the next conduit intersected by the contour are.

Our goal now is to determine for all conduits of $\mathrm{Vor}(L)$ to use the ordered pairs we generated to determine what order they are intersected by the contour globaly, and to do the same for $\mathrm{Vor}(R)$. This is a more general problem often called *list ranking*. List ranking is formally defined as given a set S of n sensors each with a single ordered pair representing the sensor's ID and it's predecessors ID determine the rank of each sensor in the linked list formed by these pairs. For example on input $(c, a), (d, c), (a, b), (f, d), (b, e)$ form the linked list (f, d, c, a, b, e) and $\mathrm{rank}(f) = 1$, $\mathrm{rank}(d) = 2, \ldots, \mathrm{rank}(e) = 6$.

Lemma 29. *In a single-hop sensor network list ranking can be computed in $O(n)$ expected time with a maximum of $O(\log n)$ energy for any sensor in expectation.*

Proof. First we assign the sensors a random ordering by sorting with a random comparator. Then we sort by ID and use a binary search so that each sensor knows its predecessor's rank in the random ordering. Each sensor will keep track of a single descendant–initially just the predecessor–and the distance from the sensor to this descendant in the linked list. Then the sensors take turns broadcasting the descendant they currently are keeping track of and their distance to this descendant. The sensors broadcast in the randomly chosen ordering. Each sensor will listen when their descendant broadcasts and then they will then keep track of that sensors descendant and calculate their distance to this descendant. After the last sensor in the chosen ordering has broadcast each sensor will know their distance to the end of the list which is sufficient to compute their rank. See Table 3.2 for example.

By Theorem 6 sorting in a single-hop sensor network can be done in $O(n)$ expected time with $O(\log n)$ maximum energy per sensor. Also n simultaneous binary searches

time	broadcasting	descendant	distance	listening
1	c	h	1	g
2	b	e	1	h
3	d	g	1	f
4	h	e	2	c, g
5	a	f	1	
6	f	g	2	a
7	g	e	4	a, f, d

Table 3.2: List ranking algorithm with input (g, c), (a, f), (d, g), (b, e), (c, h), (f, d), (h, b) using the random broadcast ordering c, b, d, h, a, f, g.

take $O(n)$ time and use $O(\log n)$ energy per sensor. Since the sensors broadcasting in the randomly chosen order takes $O(n)$ time all that remains to show is that each sensor listens to at most $O(\log n)$ descendants in expectation.

When a sensor first begins keeping track of a descendant if that descendant broadcasts in the second half of the remaining time slots then we will call this a "good" descendant. Notice that each sensor can listen to at most $O(\log n)$ "good" descendants and the probability of a descendant being "good" is $\frac{1}{2}$. So the energy each sensor uses is the number of fair coin flips until the sensor flips $O(\log n)$ heads. Even though the coin flips are correlated between sensors the maximum number of coin flips is $O(\log n)$ in expectation by Lemma 36 in the Appendix. □

Now we determine the new Voronoi vertices that lie on the contour and these are also the new endpoints of the edges intersected by the contour. We say a conduit from $\mathrm{Vor}(L)$ and a conduit from $\mathrm{Vor}(R)$ *interact* if some point on the contour lies in both faces. As pointed out in [3] we can easily determine if two conduits interact by simply checking both conduits overlap and some point in the perpendicular bisector of the point in L and the point in R lies in this overlap. Thus we need to merge the list of conduits of $\mathrm{Vor}(L)$ and the list of conduits of $\mathrm{Vor}(R)$ in the sense that we need to determine the order of pairs of conduits of $\mathrm{Vor}(L)$ and conduits of $\mathrm{Vor}(R)$

that interact. Suppose we knew the conduits of Vor(L) that the contour intersects is a, b, c, then d and the conduits of Vor(R) that the contour intersects is x, y, then z then we want to determine, for example, that the contour intersects (a, x), (b, x), (b, y), (c, y), (d, y), then (d, z). Importantly since we divided the conduits that were intersected twice by the contour each conduit is now only intersected once.

To merge the lists of conduits the median of the list of conduits of Vor(L) intersected by the contour is broadcast. Then using local information of neighboring conduits each sensor containing a conduit of Vor(R) determines if it is either the first or last conduit of Vor(R) that interacts with the median conduit from Vor(L) and consecutively broadcast the ranks of these first and last conduits. Then the only conduits of Vor(R) with rank at least as high as the last conduit that interacts with the median can interact with the conduits of Vor(L) after the median. So we simply split the lists of conduits intersected by the contour in two and recurse.

It is possible that only a single conduit interacts with the median which would mean that the sensor containing this conduit would have to follow both recursive calls. To fix this each sensor in containing a conduit of Vor(L) simply remembers the first and last conduit of Vor(R) in the range for it's recursive call. Then when a conduit is the first or last conduit that interacts with the median it does not need to continue with either recursive call and if no sensor broadcasts the first or last conduit of Vor(R) that interacts with the median it is assumed to be the first or last conduit of Vor(R) in the entire recursive call.

After the recursive calls we can use a scan to determine the rank of each pair of interacting conduits and create a new Voronoi edge for each pair. Additionally, a scan can be used to update the endpoints of each of the original Voronoi edges that intersect the contour. Thus we now have the complete Voronoi diagram for $L \cup R$ and all of this can be done in $O(n)$ time with at most $O(\log(n))$ energy per processor.

Final Redistribution of Voronoi Edges

Once the single-hop algorithm has computed the Voronoi diagram of all sensors we need to redistribute the edges so that each edge is assigned to one of the two sensors that form the edge. During each recursive call we ensured that each sensor only maintains $O(1)$ Voronoi edges but each sensor's location does not have any bearing on the edges that sensor is assigned. However for the multihop algorithm we need each Voronoi edge to be assigned to one of the two sensors that the edge is the perpendicular bisector of while still keeping the number of edges assigned to each sensor $O(1)$.

First we each sensor whose Veronoi face has 6 or fewer edges takes all of it's Voronoi edges. If both sensors try to take the same edge we use some arbitrary tie breaker to decide which sensor takes the edge. Since there are at most $3n - 6$ Voronoi edges [44] at least $\frac{n}{2}$ sensors will take their edges. If there are n' remaining sensors then the number of remaining Voronoi edges will be $3n' - 6$ because each remaining Voronoi edge will correspond to a Voronoi edge in the Voronoi diagram of the remaining n' sensors. Thus if we repeat this process with each sensor taking all of it's remaining Voronoi edges if the number of remaining edges is 6 or less all edges will be taken after $\lceil \log_2(n) \rceil$ iterations.

This can easily be implemented so that each of the $O(\log(n))$ iteration takes $O(n)$ time with $O(1)$ energy per sensor using simple scans. Therefore overall each edge will be assigned to one of the two sensors that the edge is the perpendicular bisector of, each sensor will be assigned at most 6 edges, each sensor will use at most $O(\log^2(n))$ energy, and this will take $O(n \log(n))$ time. This completes the proof of Theorem 27.

3.5.2 Multi-Hop Localized Voronoi Diagram

During the consolidation phase our goal is for each vertex to determine if the endpoints of each of its edges have been cut shorter–or if the edge has been removed–in another run of the single-hop algorithm. If this has happened then either this sensor or the other sensor which this edge is the perpendicular bisector of must contain the shorter of these endpoints.

First every sensor consolidates each edge with the information it currently has. If a sensor has the perpendicular bisector–possibly with different endpoints–between itself and another sensor for multiple runs of the single-hop algorithm then the sensor updates the edge picking the endpoints that cause the edge to be as short as possible. It picks the leftmost right endpoint and the rightmost left endpoint of all versions of the perpendicular bisector to be the actual endpoints of the edge. If the left endpoint is actually to the right of the right endpoint then the edge is instead removed.

Now, during the consolidation phase, each sensor broadcasts the consolidated versions of the edges it was distributed during the corresponding single-hop algorithm. Also for each edge a sensor has it will pick a run of the single-hop algorithm where it was not assigned a version of the edge that includes the other sensor that the edge is a perpendicular bisector of and listens to this sensor broadcast the edges it was distributed. The sensor then consolidates these version of the edge it has and the version the other sensor broadcasts the same way as before. If a sensor was distributed a version of the edge in all runs of the single-hop algorithm then it contains the final version of the edge. If the other sensor does not broadcast a version of the edge then that it must have been discarded by the other sensor or must not appear in the single-hop Voronoi diagram, so the sensor can discard this edge.

The runtime of this single-hop algorithm is $O(m \log m)$ and the energy each sensor

uses for broadcasting is $O(\log m)$ and across all runs of the consolidation algorithm the energy each sensor uses for listening is $O\left(\frac{\log^2 m}{\epsilon^2}\right)$. Then the analysis in Sections 3.2.1 and 3.2.2 completes the proof of Theorems 25 and 26 respectively.

Chapter IV

Data Propagation

4.1 Introduction

Often the goal of a wireless sensor network is to collect data from the environment and send that data to the end user via a high-power base station—or root r—acting as a data sink. In expansive networks it may be too expensive to send the data directly to the base station so instead data is propagated from one sensor to another until the data finally reaches the base station. We will assume that each sensor receives a unit of data at a fixed rate and the goal is to maximize network longevity or, equivalently, to minimize the maximum energy usage of any sensor while propagating all sensors' data to the base station.

For this problem we consider the case where sensors can vary the radio transmission power and in turn vary their broadcast radius. Sending unit data distance d requires $E(d) = e_0 + e_t d^\alpha$ energy, where e_0 is a fixed overhead for using the radio transmitter, e_t is the energy cost above e_0 to send a single unit of data a single unit distance, and α is the path loss exponent. Typically $2 \leq \alpha \leq 4$ with the exact value of α depending upon many environmental factors such as refraction, diffraction, absorp-

tion, and multipath interference [39]. In perfect free space $\alpha = 2$ and environmental factors increase α.

We do not consider any additional energy is needed to receive data from other sensors. The reason for this assumption is that, as will be apparent in our analysis, the vast majority of data sent by a sensor is also data it has received. Since the energy for receiving does not depend upon the distance of the transmission, the cost of receiving a unit of data can simply be added to e_0 without changing the results.

The model used in this chapter assumes usage of a medium access control (MAC) protocol, meaning that messages can be passed freely from sensor to sensor without fear of interference from other transmitting sensors. Schedule-based and contention-based MAC protocols are common in sensor networks [57] and will be particularly useful because of the consistent nature of the sensor transmission patterns. Because this protocol only needs to be established once when sensors are first distributed we do not consider the cost of initializing the MAC protocol.

Data propagation is already a well studied problem in the literature [4, 5]. The following is a sample of the protocols that have been designed for data propagation in different settings.

- Low Energy Adaptive Clustering Hierarchy protocol (LEACH) [22, 55] is a cluster-based protocol where sensors in a local region transmit data to a local leader who makes the possibly long transmission to the root. The leadership is rotated over time to maintain energy balance in the network. Multi-hop Routing with Low Energy Adaptive Clustering Hierarchy protocol (MR-LEACH) is a multi-hop version of LEACH where data is passed cluster by cluster to the root [17].

- Direct Diffusion (DD) [25] is a data-centric protocol that solves a related prob-

lem where the root requests data from the network and nodes that match the requested data then send data back via intermediate nodes.

- Probabilistic Forwarding Protocol (PFP) [10] is a location-based protocol where each sensor transmits data to a neighbor a small distance away towards the sink. It picks which neighbor to transmit to by randomly veering from the direct path to the root with some probability to attempt to balance the energy used by the sensors receiving the data.

- Energy Balanced Protocol (EBP) [15] is a location-based protocol where each sensor transmits data a small distance toward the sink and with some probability bypasses these small hops and transmits data directly to the base station.

While much work has been put into creating and implementing these protocols, little work has been done to analytically determine the energy usage of these protocols. One analytical result was showed that in a simplified one-dimensional model with $e_0 = 0$, EBP is optimal [43]. However these results do not generalize when $e_0 > 0$ and the authors did not analyze the exact cost of data propagation.

In Section 4.2 we generalize the one-dimensional model introduced in [43] and analyze the case when $e_0 > 0$. First we show an exact linear programming solution for data propagation and discuss the intuition this provides. Then we provide a lower bound for the energy cost of data propagation and show that EBP provides an upper bound that is close to this lower bound. Additionally we provide empirical results which compute the exact leading coefficients for the energy cost EBP. Finally, in this section we show that any protocol such as MR-LEACH which only uses short hops is only close to optimal if sending long distances is prohibitively expensive in terms of energy. In Section 4.3 we analyze a more realistic two-dimensional model and generalize the lower bound and EBP upper bound.

4.2 1-Dimensional Data Propagation

The *one-dimensional slice model* discussed in this section was first proposed in [43]. This simplified model will aid in the analysis of the more realistic two-dimensional model used in the next section and provide additional insights. In the slice model a network of n total sensors is divided into N clusters or "slices", $S_1, \ldots, S_N$. By convention if $i < j$ we say S_i is a predecessor of S_j and S_j is a successor of S_i. In the one-dimensional model the sensors are distributed at unit distances along a straight line from the root, where the root is at location 0. In other words the sensors in slice S_i are i distance from the root. To send unit data d distance costs the sending sensor $e_0 + e_t d^\alpha$ energy. Finally, to represent sensors being evenly distributed in two-dimensions, in the 1D model we assume that $|S_i| = i$. Under this assumption the total number of sensors in the network is $n = \frac{N^2+N}{2}$.

We also assume that work is evenly distributed among sensors at the same location. That is, each sensor i away from the root receives $\frac{1}{i}$ of the total data sent to sensors at location i and sends $\frac{1}{i}$ of the data sent from location i to location j for each other location j. The 1D model uses our standard data propagation assumption and goal: every sensor generates unit data and the goal is to send all data base station while minimizing the maximum energy usage of any sensor.

In this section we determine the exact cost of energy optimal data propagation in the 1D model. The following theorems are the main results of this section. Their proofs appear in Subsections 4.2.2 and 4.2.3. The first theorem, regards the exact energy cost of data propagation in the 1D network slice model.

Theorem 30. *Let S be a sensor network in the one-dimensional network slice model with n sensors, radius N, and energy cost $e_0 + e_t d^\alpha$ of sending unit data d distance*

away. If $\alpha > 2$ then the optimal data propagation on S takes

$$C(\alpha, e_0, e_t)(\alpha - 2)\max\left(e_t,\, e_0^{1-\frac{2}{\alpha}} e_t^{\frac{2}{\alpha}}\right) n + l.o.t.$$

maximum energy per sensor, where C is a function of α, e_0, and e_t bounded above and below by positive constants. If $\alpha = 2$ then optimal data propagation on S takes

$$\frac{e_t n}{\ln(N)} + l.o.t.$$

maximum energy per sensor.

For $\alpha > 2$ the leading coefficient contains a $(\alpha - 2)$ term, meaning that as α decreases the leading coefficient also decreases until at $\alpha = 2$ becomes a $\frac{1}{\ln N} \approx \frac{2}{\ln n}$ term. This follows the intuition that the network should be more efficient with smaller values of α.

The theorem also shows that as the fixed overhead for transmitting data e_0 decreases the energy usage decreases but there is a limit to how much this can help. On the other hand decreasing e_t will always decrease the energy usage. The larger α is the more important it is to keep e_0 small. As α approaches 2 keeping e_t small becomes much more important to the point where e_0 does not appear in the high order terms at all if $\alpha = 2$.

The other main theorem of this section states that EBP is close to optimal.

Theorem 31. *On a sensor network in the one-dimensional network slice model the energy balanced protocol (EBP) is within a constant factor optimal and if $\alpha = 2$ is within low order terms of optimal.*

4.2.1 Exactly Optimal Linear Programming Results

There exists a simple centralized linear programming solution to data propagation which exactly minimizes the maximum energy usage of any sensor, as [43] points out. This linear program is not specific to the network slice model and finds the exact optimal solution regardless of the position of the sensors. Additionally, a linear program can be designed for nearly any energy cost model. However this solution is not as useful in practice because it is centralized and requires knowing the exact locations of all sensors. Furthermore this solution may not be robust to small perturbations in the parameters of the network. Still, this solution can provide intuition and clarification as to what a more practical solution may look like.

We will present the linear program that is specific to our energy cost model but is general in terms sensor positioning, including the dimensionality of the sensor positions. Let d_i be the rate at which sensor s_i produces data and let $f_{i,j}$ be the units of data sent from sensor s_i to sensor s_j. Additionally, assume that the base station is located at the origin and that the location of each sensor s_i is x_i. Our goal then becomes to minimize E where the following system of linear constraints are met for each sensor s_i.

$$E \geq (e_0 + e_t \|x_i\|^\alpha) \left(d_i + \sum_{j \neq i} f_{i,j} \right) - \sum_{j \neq i} e_t \left(\|x_i\|^\alpha - \|x_i - x_j\|^\alpha \right) f_{j,i} \tag{4.1}$$

$$d_i + \sum_{j \neq i} f_{j,i} \geq \sum_{j \neq i} f_{i,j} \tag{4.2}$$

$$f_{i,j} \geq 0 \tag{4.3}$$

Equation 4.1 requires that E is bounded by the energy spent for each sensor s_i.

Equation 4.2 requires that the data generated by sensor s_i plus the data s_i receives is not exceeded by the data that s_i sends to sensors other than the base station. Finally, Equation 4.3 simply requires that the data sent from one sensor to another is non-negative. Then the minimum value for E under these constraints which are linear in $f_{i,j}$ is the maximum energy usage of any sensor.

Table 4.1 shows the optimal communication pattern for data propagation of the 1D network slices model with $N = 8$, $e_0 = 1$, $e_t = 0.2$, and $\alpha = 2$. Notice that in this solution each sensor sends some data along fixed distance short hops and sends the rest directly to the base station. This provides empirical evidence that EBP may be an optimal or near optimal solution and motivates the usage of EBP to provide an upper bound on optimal maximum energy usage in Subsection 4.2.3.

One concern that arises from this empirical result is that some sensors send to multiple clusters along the short hops. In EBP typically sensors only to a single cluster along the short hop a fixed number of hops away δ. This reduces the overhead of the MAC protocol used to set up communication and removes the problem of determining how to distribute data among the different short hops a single sensor uses. A related concern is how each sensor determines how much data to send to the root which seems to vary seemingly without pattern in the empirical result. However with a target maximum energy usage a sensor can easily determine how much energy it can send to the root given the amount of data it receives from other sensors.

Another challenge is that EBP is not always optimal if small changes are made to the one-dimensional network slices model. Consider the case where there is a hole in the middle of the middle of the network. Table 4.2 shows the optimal linear programming solution to this problem with $N = 12$, $e_0 = 1$, $e_t = 0.2$, and $\alpha = 2$ but slices $S_5, S_6, \ldots S_12$ have no sensors. In some ways this solution resembles EBP in that it appears most sensors send data along a short hop and a long hop. However

Optimal Data Propagation

		Receiving								
		Root	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
Sending	S_1	5.52	0	0	0	0	0	0	0	0
	S_2	7.35	0	0	0	0	0	0	0	0
	S_3	7.09	0	0	0	0	0	0	0	0
	S_4	6.3	0	0	0	0	0	0	0	0
	S_5	0.96	4.52	2.98	0	0	0	0	0	0
	S_6	3.63	0	2.37	0	0	0	0	0	0
	S_7	2.63	0	0	4.09	0.28	0	0	0	0
	S_8	2.52	0	0	0	2.02	3.46	0	0	0

Table 4.1: The optimal data propagation pattern for the one-dimensional network slice model with $N = 8$, $e_0 = 1$, $e_t = 0.2$, and $\alpha = 2$. The entry of row i column j is the total data sent from sensors in slice S_i to sensors in slice S_j. This configuration results in each sensor using 6.62 units of energy.

the destination of the long hop is not the root for sensors on the far side of the hole. Interestingly, sensors in slice S_{16} send more data to sensors in slice S_3 than sensors in slice S_{15} do and sensors in slice S_{15} also send data to sensors in slice S_4 where as sensors in slice S_{16} do not.

4.2.2 1-Dimensional Lower Bound

This subsection contains lower bounds on the maximum energy usage of any sensor to propagate data to a high-energy base station in the one-dimensional network slice model. This along with Subsection 4.2.3 provides proof of Theorems 30 and 31.

First let us assume that each sensor spends all of its energy sending data directly to the root. That is, assume that sending data between sensors takes no energy. Then for each unit of energy the network can transmit $\sum_{i=1}^{N} \frac{i}{e_0+e_t i^\alpha}$ units of data to the root. The network produces a total of n units of data which must all be sent to the root. Therefore the maximum energy usage of any sensor E must obey the following bound.

Optimal Data Propagation with a Hole

		Receiving								
		Root	S_1	S_2	S_3	S_4	S_{13}	S_{14}	S_{15}	S_{16}
Sending	S_1	19.8	0	0	0	0	0	0	0	0
	S_2	26.39	0	0	0	0	0	0	0	0
	S_3	21.8	5.66	0	0	0	0	0	0	0
	S_4	0	13.14	24.39	11.93	0	0	0	0	0
	S_{13}	0	0	0	0	17.95	0	0	0	0
	S_{14}	0	0	0	0	15.84	0	0	0	0
	S_{15}	0	0	0	2	11.67	1.32	0	0	0
	S_{16}	0	0	0	10.53	0	3.63	1.84	0	0

Table 4.2: The optimal data propagation pattern for a modified one-dimensional network slice model with a hole and with $N = 16$, $e_0 = 1$, $e_t = 0.2$, and $\alpha = 2$. The only modification of the network is the hole in the network from slice 5 through 12 where there are no sensors. That is $|S_5| = |S_6| = \ldots = |S_{12}| = 0$ The entry of row i column j is the total data sent from sensors in slice S_i to sensors in slice S_j. This configuration results in each sensor using 23.75 units of energy.

$$E \geq n \Big/ \sum_{i=1}^{N} \frac{i}{e_0 + e_t i^{\alpha}} \tag{4.4}$$

An easy lower bound for this term is to ignore the e_0 term and get the following.

$$\begin{aligned} E &\geq n \Big/ \sum_{i=1}^{N} \frac{i}{e_0 + e_t i^{\alpha}} \\ &\geq e_t n \Big/ \sum_{i=1}^{N} \frac{1}{i^{\alpha-1}} \\ &\geq e_t n \Big/ \left(1 + \int_1^N \frac{1}{x^{\alpha-1}} dx\right) \end{aligned}$$

If $\alpha = 2$ then

$$E \geq e_t n \Big/ (1 + \ln N)$$

$$= \frac{e_t n}{\ln N} - O\left(\frac{n}{\log^2 n}\right)$$

On the other hand if $\alpha > 2$ then

$$\begin{aligned} E &\geq e_t n \Big/ \left(1 + \frac{1}{\alpha - 2}\right) \\ &= \frac{1}{\alpha - 1}(\alpha - 2) e_t n \end{aligned} \tag{4.5}$$

This bound however is not very tight when $\alpha > 2$ and e_0 is much larger than e_t. To find a tighter bound for Equation 4.4 we find an upper bound for $\sum_{i=1}^{N} \frac{i}{e_0 + e_t i^\alpha}$. We let y be the maximum of the function $\frac{i}{e_0 + e_t i^\alpha}$ and use that as an upper bound when i is small. After some point w, for all $i > w$ we have $\frac{1}{e_t i^\alpha} < y$. Thus before w we estimate $\frac{i}{e_0 + e_t i^\alpha}$ with y and after w we use $\frac{1}{e_0 + e_t i^\alpha}$. See Figure 4.1 for illustration. This gives us the following lower bound.

$$\sum_{i=1}^{N} \frac{i}{e_0 + e_t i^\alpha} \leq wy + \int_{w}^{N} \frac{1}{e_t x^{\alpha - 1}} dx$$

First we find the value $x_{\max}$ which maximizes $\frac{x}{e_0 + e_t x^\alpha}$.

$$\frac{d}{dx} \frac{x}{e_0 + e_t x^\alpha} = \frac{e_0 - e_t(\alpha - 1)x^\alpha}{(e_0 + e_t x^\alpha)^2}$$

$$x_{\max} = \left(\frac{e_0}{(\alpha - 1)e_t}\right)^{\frac{1}{\alpha}}$$

Then we find y, which is the maximum of the function $\frac{x}{e_0 + e_t x^\alpha}$, by plugging in $x_{\max}$.

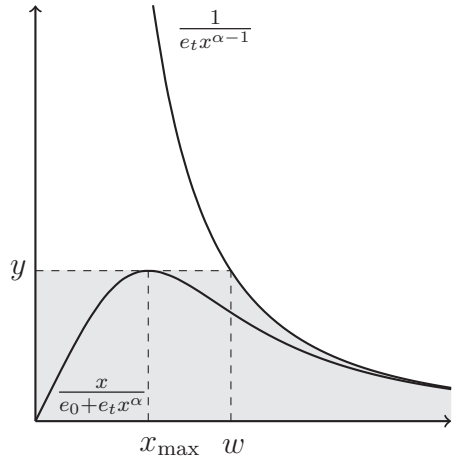


Figure 4.1: We estimate $\sum_{i=1}^{N} \frac{i}{e_0+e_t i^\alpha}$ with the shaded region above with area $wy + \int_w^N \frac{1}{e_t x^{\alpha-1}} dx$. In this figure $e_0 = \frac{1}{2}$, $e_t = \frac{1}{10}$, and $\alpha = 3$.

$$\begin{aligned} y &= \frac{x_{\max}}{e_0 + e_t {x_{\max}}^\alpha} \\ &= \frac{\left(\frac{e_0}{(\alpha-1)e_t}\right)^{\frac{1}{\alpha}}}{e_0 + e_t \frac{e_0}{(\alpha-1)e_t}} \\ &= \frac{1}{(\alpha-1)^{\frac{1}{\alpha}} \left(1 + \frac{1}{\alpha-1}\right) {e_0}^{1-\frac{1}{\alpha}} {e_t}^{\frac{1}{\alpha}}} \\ &= \frac{(\alpha-1)^{1-\frac{1}{\alpha}}}{{e_0}^{1-\frac{1}{\alpha}} {e_t}^{\frac{1}{\alpha}} \alpha} \end{aligned}$$

Then we solve for w in the equation $y = \frac{1}{w^{\alpha-1} e_t}$.

$$\begin{aligned} w &= \left(\frac{1}{e_t y}\right)^{\frac{1}{\alpha-1}} \\ &= \left(\frac{\alpha {e_0}^{1-\frac{1}{\alpha}}}{(\alpha-1)^{1-\frac{1}{\alpha}} {e_t}^{1-\frac{1}{\alpha}}}\right)^{\frac{1}{\alpha-1}} \end{aligned}$$

$$= \frac{\alpha^{\frac{1}{\alpha-1}} {e_0}^{\frac{1}{\alpha}}}{(\alpha-1)^{\frac{1}{\alpha}} {e_t}^{\frac{1}{\alpha}}}$$

Now we compute wy.

$$\begin{aligned} wy &= \frac{\alpha^{\frac{1}{\alpha-1}} {e_0}^{\frac{1}{\alpha}}}{(\alpha-1)^{\frac{1}{\alpha}} {e_t}^{\frac{1}{\alpha}}} \cdot \frac{(\alpha-1)^{1-\frac{1}{\alpha}}}{\alpha {e_0}^{1-\frac{1}{\alpha}} {e_t}^{\frac{1}{\alpha}}} \\ &= \frac{(\alpha-1)^{1-\frac{2}{\alpha}}}{\alpha^{1-\frac{1}{\alpha-1}} {e_0}^{1-\frac{2}{\alpha}} {e_t}^{\frac{2}{\alpha}}} \end{aligned}$$

Then by solving the integral we get the following.

$$\begin{aligned} wy + \int_w^N \frac{1}{e_t x^{\alpha-1}} dx &= \frac{(\alpha-1)^{1-\frac{2}{\alpha}}}{\alpha^{1-\frac{1}{\alpha-1}} {e_0}^{1-\frac{2}{\alpha}} {e_t}^{\frac{2}{\alpha}}} + \frac{1}{(\alpha-2)e_t w^{\alpha-2}} - \frac{1}{(\alpha-2)e_t N^{\alpha-2}} \\ &\leq \frac{(\alpha-1)^{1-\frac{2}{\alpha}}}{\alpha^{1-\frac{1}{\alpha-1}} {e_0}^{1-\frac{2}{\alpha}} {e_t}^{\frac{2}{\alpha}}} + \frac{\left(\frac{(\alpha-1)^{\frac{1}{\alpha}} {e_t}^{\frac{1}{\alpha}}}{\alpha^{\frac{1}{\alpha-1}} {e_0}^{\frac{1}{\alpha}}}\right)^{\alpha-2}}{(\alpha-2)e_t} \\ &= \frac{(\alpha-1)^{1-\frac{2}{\alpha}}}{\alpha^{1-\frac{1}{\alpha-1}} {e_0}^{1-\frac{2}{\alpha}} {e_t}^{\frac{2}{\alpha}}} + \frac{(\alpha-1)^{1-\frac{2}{\alpha}}}{\alpha^{1-\frac{1}{\alpha-1}} (\alpha-2) {e_0}^{1-\frac{2}{\alpha}} {e_t}^{\frac{2}{\alpha}}} \\ &= \frac{(1+\frac{1}{\alpha-2})(\alpha-1)^{1-\frac{2}{\alpha}}}{\alpha^{1-\frac{1}{\alpha-1}} {e_0}^{1-\frac{2}{\alpha}} {e_t}^{\frac{2}{\alpha}}} \\ &= \frac{(\alpha-1)^{2-\frac{2}{\alpha}}}{\alpha^{1-\frac{1}{\alpha-1}} (\alpha-2) {e_0}^{1-\frac{2}{\alpha}} {e_t}^{\frac{2}{\alpha}}} \end{aligned}$$

Finally, this gives the following lower bound.

$$E \geq \frac{\alpha^{1-\frac{1}{\alpha-1}}}{(\alpha-1)^{2-\frac{2}{\alpha}}} (\alpha-2) {e_0}^{1-\frac{2}{\alpha}} {e_t}^{\frac{2}{\alpha}} n \tag{4.6}$$

This completes the lower bound of Theorems 30 and 31.

4.2.3 1-Dimensional EBP Upper Bound

This subsection analyzes the cost of using the Energy Balanced Protocol (EBP), which provides an upper bound for data propagation, in the one-dimensional network slice model. In EBP each cluster will with some probability send data directly to the base station also called the root. The remaining data is passed along a short hop to another cluster of sensors a fixed number of hops away, δ.

To analyze the energy usage of EBP we introduce the concept of *data efficiency*. Let d_i be the amount of data sent to the base station per sensor in slice i. We define $\eta_i = \frac{d_i}{E}$ as the data efficiency of slice i.

Note that each cluster only receives data that originated from other clusters that are equivalent modulo δ. That is sensors from clusters $S_i, S_{i+\delta}, S_{i+2\delta}, \ldots$ only ever use data from other sensors in clusters $S_i, S_{i+\delta}, S_{i+2\delta}, \ldots$. We say that sensors in clusters $S_i, S_{i+\delta}, S_{i+2\delta}, \ldots$ are the ith *partition* of the network. Since each partition only interacts with other sensors in the same partition we can consider each partition a completely separate parallel network and can be analyzed independently. For the sake of simplicity we will first only analyze the δ partition containing clusters $S_\delta, S_{2\delta}, \ldots$, then argue that the results hold for all partitions.

To further simplify our analysis we assume that N is divisible by δ. Finally we assume that each generates exactly $\frac{n}{\delta}$ data. The exact quantity data contained by the δ partition is $\sum_{i=1}^{\frac{N}{\delta}} i\delta = \frac{N^2/\delta+N}{2} = \frac{n}{\delta} + O(\sqrt{n})$ and so this assumption only impacts low order terms.

Under these assumptions we bound the maximum energy usage of any sensor by simply using the definition of data efficiency.

$$E = n \Big/ \delta \sum_{i=1}^{\frac{N}{\delta}} i\delta\eta_{i\delta}$$
$$= n \Big/ \delta^2 \sum_{i=1}^{\frac{N}{\delta}} i\eta_{i\delta} \tag{4.7}$$

Now we bound the data efficiency by noting the total amount of data sent from sensors in slice $S_{(i+1)\delta}$ to sensors in slice $S_{i\delta}$ is $\sum_{j=1}^{i} j\delta(d_{j\delta} - 1)$. Note that the -1 term in the sum is due to the data generated by each sensor. This can be used to give us a lower bound on $d_{i\delta}$.

$$d_{i\delta} = \frac{E - \frac{e_0 + e_t\delta^\alpha}{i} \sum_{j=1}^{i-1} j(d_{j\delta} - 1)}{e_0 + e_t i^\alpha \delta^\alpha}$$
$$\geq \frac{E - \frac{e_0 + e_t\delta^\alpha}{i} \sum_{j=1}^{i-1} j d_{j\delta}}{e_0 + e_t i^\alpha \delta^\alpha}$$

Because $\eta_i = \frac{d_i}{E}$ we can now bound the data efficiency.

$$\eta_{i\delta} \geq \frac{1 - \frac{e_0 + e_t\delta^\alpha}{i} \sum_{j=1}^{i-1} j\eta_{j\delta}}{e_0 + e_t i^\alpha \delta^\alpha} \tag{4.8}$$

The base case for η_δ is $\frac{1}{e_0 + e_t\delta^\alpha}$ because all data sent received by these sensors is sent directly to the root. Thus we get the following bounds for the data efficiency.

$$\eta_{i\delta} \geq \frac{1 - \frac{1}{i} - \frac{e_0 + e_t\delta^\alpha}{i} \sum_{j=2}^{i-1} j\eta_{i\delta}}{e_0 + e_t i^\alpha \delta^\alpha}$$

$$\geq \frac{1 - \frac{1}{i} - \frac{e_0 + e_t\delta^\alpha}{i} \sum_{j=2}^{i-1} \frac{j-1}{e_0 + e_t j^\alpha \delta^\alpha}}{e_0 + e_t i^\alpha \delta^\alpha}$$

If $\alpha = 2$ then this simplifies to the following.

$$\eta_{i\delta} \geq \frac{1 - O\left(\frac{\log(i)}{i}\right)}{e_0 + e_t i^2 \delta^2}$$

$$\begin{aligned} E &\leq n \Big/ \delta^2 \sum_{i=1}^{\frac{N}{\delta}} \frac{i - O(\log i)}{e_0 + e_t i^2 \delta^2} \\ &= n \Big/ \left(\frac{\ln(N)}{e_t} - \frac{\ln(\delta)}{e_t} - O(1) \right) \end{aligned}$$

As long as δ is constant the $\ln(\delta)$ term is $O(1)$.

$$E \leq \frac{2e_t n}{\ln(n)} + O\left(\frac{n}{\log^2(n)}\right)$$

This bound holds regardless of the partition being considered because the slices far away from the root are important. For the jth partition $\sum_{i=0}^{\frac{N}{\delta}-1} (i\delta + j)\eta_{i\delta}$ is still $O(\log n)$. Thus $\sum_{i=0}^{\frac{N}{\delta}-1} \frac{i\delta + j - O(\log i)}{e_0 + e_t(i\delta + j)^2} = \left(\frac{\ln(N)}{e_t} - \frac{\ln(\delta)}{e_t} - O(1) \right) / \delta$

$$\begin{aligned} E &\leq n \Big/ \delta \sum_{i=0}^{\frac{N}{\delta}-1} \frac{i\delta + j - O(\log i)}{e_0 + e_t(i\delta + j)^2} \\ &= n \Big/ \left(\frac{\ln(N)}{e_t} - \frac{\ln(\delta)}{e_t} - O(1) \right) \end{aligned}$$

On the other hand if $\alpha > 2$ then we let $\delta = \left\lceil \left(\frac{e_0}{e_t}\right)^{\frac{1}{\alpha}} \right\rceil$.

$$\begin{aligned} E &\leq n \Big/ \delta^2 \sum_{i=1}^{\frac{N}{\delta}} i\eta_{i\delta} \\ &= n \Big/ \delta^2 \left(\frac{1}{e_0 + e_t\delta^\alpha} + \sum_{i=2}^{\frac{N}{\delta}} \frac{i - 1 - \sum_{j=2}^{i-1} \frac{(j-1)(e_0+e_t\delta^\alpha)}{e_0+e_t j^\alpha \delta^\alpha}}{e_0 + e_t i^\alpha \delta^\alpha} \right) \end{aligned}$$

Notice that each $\frac{(j-1)(e_0+e_t\delta^\alpha)}{e_0+e_t j^\alpha \delta^\alpha}$ term is decreasing as δ increases because $j > 1$. If $\delta \geq \left(\frac{e_0}{e_t}\right)^{\frac{1}{\alpha}}$ then

$$\begin{aligned} \frac{(j-1)(e_0 + e_t\delta^\alpha)}{e_0 + e_t j^\alpha \delta^\alpha} &\leq \frac{2e_0(j-1)}{e_0(1+j^\alpha)} \\ &= \frac{2j-2}{1+j^\alpha} \\ &\leq \frac{2j-2}{1+j^2} \end{aligned}$$

It turns out $\frac{2j-2}{1+j^2}$ reaches a maximum of $\sqrt{2}-1 \approx 0.414$ at $1+\sqrt{2}$. When $i = 2$ we have $i - 1 - \sum_{j=2}^{i-1} \frac{2j-2}{1+j^\alpha} = 1$ and then when i increases by 1 the sum gains an additional term in the sum less than $\frac{1}{2}$. Therefore $i-1-\sum_{j=2}^{i-1} \frac{2j-2}{1+j^\alpha} \geq \frac{i}{2}$ when $i \geq 2$ and $\delta \geq \left(\frac{e_0}{e_t}\right)^{\frac{1}{\alpha}}$. Therefore if $\delta \geq \left(\frac{e_0}{e_t}\right)^{\frac{1}{\alpha}}$ then

$$E \leq 2n \Big/ \delta^2 \sum_{i=1}^{\frac{N}{\delta}} \frac{i}{e_0 + e_t\delta^\alpha i^\alpha} \tag{4.9}$$

Thus we let $\delta = \left\lceil \left(\frac{e_0}{e_t}\right)^{\frac{1}{\alpha}} \right\rceil$. If $e_0 \leq e_t$ then $\delta = 1$ and Equation 4.9 simplifies to

$$E \leq 2n \Big/ \sum_{i=1}^{N} \frac{i}{e_0 + e_t i^\alpha}$$

which is exactly double the lower bound in Equation 4.4. Furthermore,

$$\begin{aligned} E &\leq 2n \Big/ \int_1^{N+1} \frac{x}{e_t(1+x^\alpha)} dx \\ &\leq 2e_t n \Big/ \int_1^{N+1} \frac{x}{2x^\alpha} dx \\ &\leq 4(\alpha - 2) e_t n \end{aligned}$$

On the other hand if $e_0 \geq e_t$ then Equation 4.9 simplifies to

$$\begin{aligned} E &\leq 2e_0^{-\frac{2}{\alpha}} e_t^{\frac{2}{\alpha}} n \Big/ \sum_{i=1}^{\left\lfloor \frac{N}{\delta} \right\rfloor} \frac{i}{e_0 + e_t \delta^\alpha i^\alpha} \\ &\leq 2e_0^{1-\frac{2}{\alpha}} e_t^{\frac{2}{\alpha}} n \Big/ \sum_{i=1}^{\left\lfloor \frac{N}{\delta} \right\rfloor} \frac{i}{e_0 + 2^\alpha e_0 i^\alpha} \\ &\leq 2(2^\alpha + 1) e_0^{1-\frac{2}{\alpha}} e_t^{\frac{2}{\alpha}} n \Big/ \int_1^{\frac{N}{\delta}} x^{1-\alpha} dx \\ &\leq 2(2^\alpha + 1) e_0^{1-\frac{2}{\alpha}} e_t^{\frac{2}{\alpha}} n \Big/ \frac{1 - \left(\frac{N}{\delta}\right)^{2-\alpha}}{\alpha - 2} \\ &\leq 2(2^\alpha + 1)(\alpha - 2) e_0^{1-\frac{2}{\alpha}} e_t^{\frac{2}{\alpha}} n + O\left(n^{2-\frac{\alpha}{2}}\right) \end{aligned}$$

These bounds still hold within a constant factor regardless of the partition being considered. To show this we consider a modified EBP protocol where sensors in slice $S_{i\delta+j}$ sends the same total amount of data to the root as sensors in slice $S_{i\delta}$. This protocol uses strictly more energy then the typical EBP protocol and each sensor in slice $S_{i\delta+j}$ uses at most a constant factor more energy than the sensors in slice $S_{i\delta}$.

This subsection along with Subsection 4.2.2 completes the proofs of Theorems 30 and 31.

4.2.4 Empirical Results

In this subsection we show empirical results for the energy balanced protocol (EBP). We approximate the energy usage by using Equation 4.7 for energy usage and using Equation 4.8 for the data efficiency $\eta_{i\delta}$. Notice that multiplying e_0 and e_t by k exactly increases the energy by a factor of k. Therefore the actual magnitude of e_0 and e_t do not affect the constant C in $E = C(\alpha, e_0, e_t)(\alpha - 2)\max\left(e_t,\, e_0^{1-\frac{2}{\alpha}} e_t^{\frac{2}{\alpha}}\right) n + l.o.t.$ for EBP, just the ratio of e_0 and e_t. Similarly the lower bound from Equation 4.4 increases by a factor of k if e_0 and e_t and increased by a factor of k. Thus in this section we will only be analyzing varying ratios of e_0 to e_t ignoring their actual magnitudes.

In our analysis we let $\frac{e_0}{e_t}$ range from $\frac{1}{1{,}000}$ to $1,000$ on a log scale. We also analyze values for α ranging from 2—wherever possible—to 4. In order to minimize low order terms in our calculations we use $N = 10^8$ and compute exact values using equations 4.7 and 4.8.

First in Figure 4.2 we show the optimal value for the short hop distance δ to minimize energy usage. These values for δ are very close to the $\left(\frac{e_0}{e_t}\right)^{\frac{1}{\alpha}}$ even when $\alpha = 2$.

Next in Figure 4.3 we plot the approximation ratio for EBP which is obtained by dividing Equation 4.7 by Equation 4.4. This gives us a maximum approximation ratio for EBP of about 1.35. This ratio appears to be highest when $e_0 > e_t$ and α is around 3. When $\alpha = 2$ the ratio appears to approach 1 which is to be expected and interestingly when α increases towards 4 the ratio actually appears to go down a little.

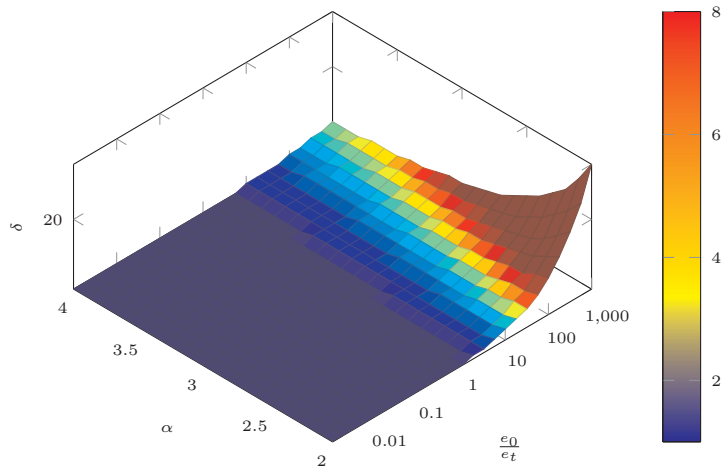


Figure 4.2: Optimal value for δ for $\frac{e_0}{e_t}$ from 0.001 and 1,000, α between 2 and 4.

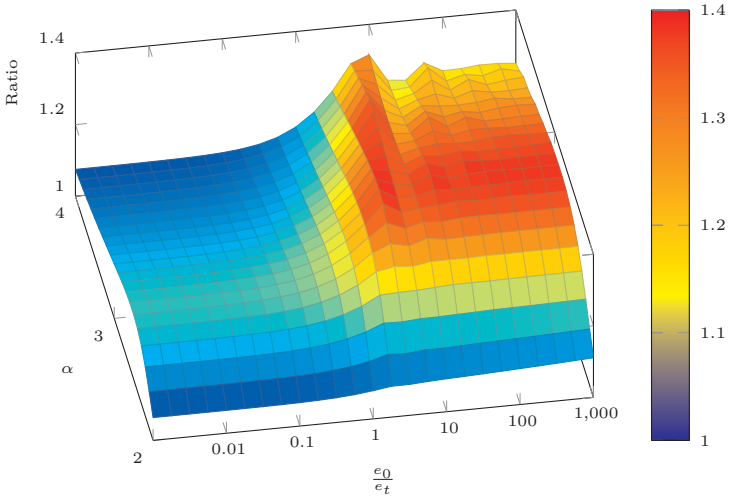


Figure 4.3: Approximation ratio for $\frac{e_0}{e_t}$ from 0.001 and 1,000, α between 2 and 4, with optimal values for d.

It is also worth pointing out there are some significant undulations that appear when α approaches 4 and when e_0 is slightly bigger than e_t. This is likely caused by the granularity of δ. That is, initially $\delta = 1$ and the ratio increases with $\frac{e_0}{e_t}$ until $\delta = 2$ where the ratio begins to decrease for a while. This affect is visible with smaller values for α but is most visible when α is large because δ increases more slowly with $\frac{e_0}{e_t}$ when α is large.

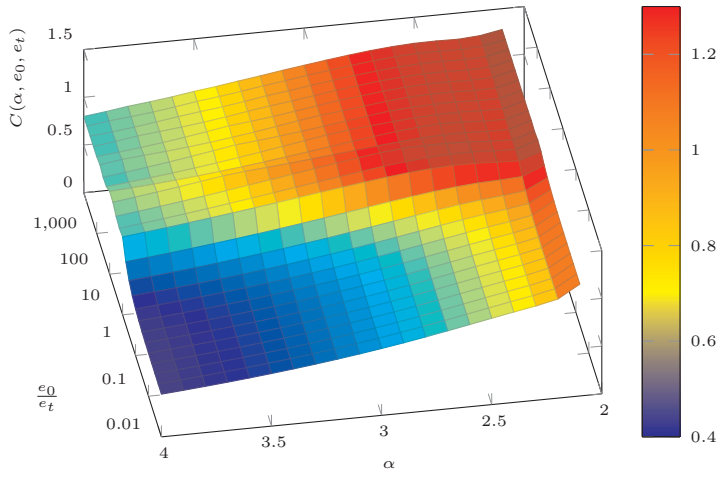


Figure 4.4: Coefficient for $\frac{e_0}{e_t}$ from 0.001 and 1,000, α between 2.1 and 4, with optimal values for δ.

Finally, from theorems 30 and 31 we know that if $\alpha > 2$ then the energy used for EBP is of the form $C(\alpha, e_0, e_t)(\alpha - 2)\max\left(e_t,\, e_0^{1-\frac{2}{\alpha}} e_t^{\frac{2}{\alpha}}\right) n + l.o.t.$ where C is a function bounded above and below by a positive constant. In Figure 4.4 plot this coefficient by dividing Equation 4.7 by $(\alpha - 2)\max\left(e_t,\, e_0^{1-\frac{2}{\alpha}} e_t^{\frac{2}{\alpha}}\right) n$.

It appears that C varies from around 0.45 to around 1.24. The coefficient appears to increase as α decreases and increases sharply when e_0 becomes larger than e_t. There is also a sharp increase present in the figure as α approaches 2 which appears to be because low order terms have a much larger impact as α is small.

4.2.5 Protocols Using Only Short Hops

In this subsection we analyze protocols such as Multi-hop Routing with Low Energy Adaptive Clustering Hierarchy protocol (MR-LEACH) and Probabilistic Forwarding Protocol (PFP) in which sensors far from the root only transfer data short distances to nearby sensors and do not send data long distances to the base station. The goal of this subsection is to show that these protocols are only close to optimal when sending messages long distances is expensive, i.e., when α is large. That is, we

will show that as α approaches 2 the energy these protocols use is much greater than the bounds found in Subsections 4.2.2 and 4.2.3.

We assume that the short hop distance is a fixed value δ depending upon the parameters of the network. There are $\frac{\delta(\delta+1)}{2}$ sensors in slices $S_1, \ldots, S_\delta$ within δ of the root and these are the sensors that send the data directly to the root. To simplify our analysis we will assume that there are only $\frac{\delta^2}{2}$ sensors in slices $S_1, \ldots, S_\delta$. Some protocols such as PFP attempt to more evenly distribute data by randomizing the transmission path. As a generous assumption we assume that the data is evenly distributed among these sensors within δ of the root.

Sensors further than δ away from the root use strictly less energy than the sensors δ away from the root because there are more sensors to distribute the data and each sensor transfers data at most distance δ. Under our assumptions the maximum energy usage of any sensor using a protocol only using short hops of distance δ satisfies

$$E \geq \frac{e_0 + e_t\delta^\alpha}{\delta^2} 2n$$

To determine the optimal choice of δ we simply take the derivative with respect to δ.

$$\frac{d}{d\delta}\frac{e_0 + e_t\delta^\alpha}{\delta^2} 2n = \frac{(\alpha-2)e_t\delta^\alpha - 2e_0}{\delta^3} 2n$$

If $\alpha = 2$ then this simplifies to $\frac{-4e_0}{\delta^3}n$ which implies that there is no fixed δ that minimizes E. In the limit as δ goes to infinity the energy usage approaches $2e_tn$ for the short hop approach. However our lower bound and the result achieved when using both short hops and long hops was $\frac{e_tn}{\ln(N)} + l.o.t.$. Therefore if $\alpha = 2$, protocols using only short hops are at least a $\ln(n)$ factor away from optimal.

On the other hand if $\alpha > 2$ then energy is minimized when

$$\delta = \left(\frac{2e_0}{(\alpha-2)e_t}\right)^{\frac{1}{\alpha}}$$

If $e_t > e_0$ then energy is minimized when $\delta = 1$ since δ must be a positive integer. In this case $E \geq (e_0 + e_t)n \geq e_t n$ when using only short hops. Using Equation 4.5 as a lower bound gives us an approximation ratio of $\frac{\alpha-1}{\alpha-2}$. Note that as α approaches 2 this ratio approaches infinity.

If $e_0 \geq e_t$ then we simply use the value we previously found for δ. Even though this value may not be a whole number it still provides a bound for E.

$$\begin{aligned} E &\geq \frac{e_0 + e_t\delta^\alpha}{\delta^2} 2n \\ &\geq 2\frac{e_0 + \frac{2e_0}{\alpha-2}}{\left(\frac{2e_0}{(\alpha-2)e_t}\right)^{\frac{2}{\alpha}}} \\ &= \frac{2^{1-\frac{2}{\alpha}}\alpha}{(\alpha-2)^{1-\frac{2}{\alpha}}} e_0^{1-\frac{2}{\alpha}} e_t^{\frac{2}{\alpha}} n \end{aligned}$$

Dividing this by the lower bound in Equation 4.6 gives us the following approximation ratio:

$$\text{Ratio} = \frac{2^{1-\frac{2}{\alpha}}\alpha^{\frac{1}{\alpha-1}}(\alpha-1)^{2-\frac{2}{\alpha}}}{(\alpha-2)^{2-\frac{2}{\alpha}}}$$

Again this ratio goes to infinity as α goes to 2. As α increases the ratio decreases and when $\alpha = 4$ the ratio is approximately 4.12. Figure 4.5 plots the exact values of this ratio.

All of these ratios show that data propagation protocols that only use short hops such as MR-LEACH and PFP do not perform well compared to optimal solutions when α is close to 2. They can only perform well when sending long distances is particularly expensive.

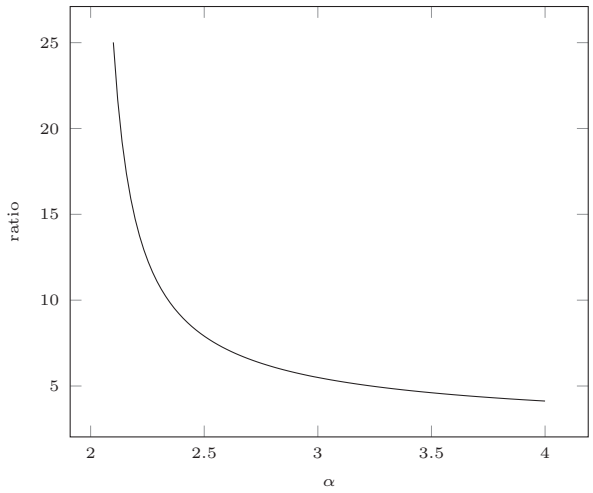


Figure 4.5: Plot of approximation ratio of the energy usage for the short hop protocol when $e_0 \geq e_t$ and the lower bound in Equation 4.6.

4.3 2-Dimensional Data Propagation

In this section we discuss the two-dimensional model and show how the bounds from Section 4.2 carry over to two dimensions. The network is distributed on a a disk of radius R which has been partitioned into equally sized sectors, each of which is subtended by an angle of θ from the center of the disk. Each sector is partitioned into N evenly spaced *slices*, $S_1, \ldots, S_N$, such that all sensors at distance $(\frac{(i-1)R}{N}, \frac{iR}{N}]$ from the base station are in slice S_i. This is the 2D model used in [15].

In this slice model the distance between the leftmost sensor in slice S_i and the rightmost sensor in S_{i+1} may be may be proportional to i. This means that sensors further away from the root may spend large amounts of energy on the "short hops" between slices. This is insufficient for the analysis in this chapter, so we introduce a new variant of this model we call the *splitting slice model*. In this model we split each slice S_i into 2^k smaller clusters where $k = \lfloor \log_2 i \rfloor$ by evenly spaced lines emanating from the base station. That is, S_1 remains a single cluster but S_2 and S_3 are split

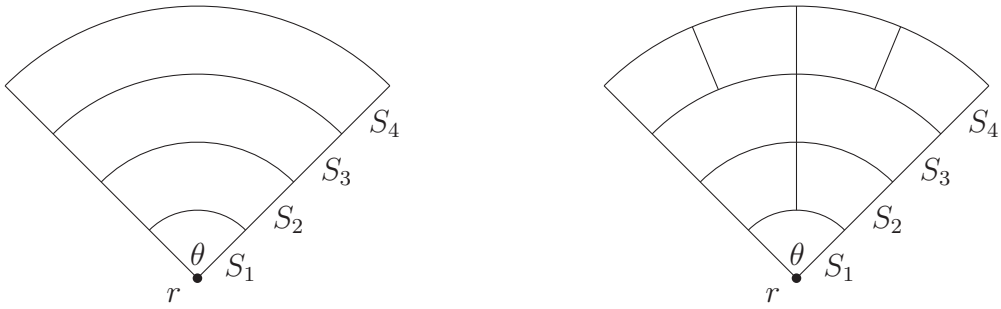


Figure 4.6: On the left is the slice model with 4 slices. On the right is the splitting slice model with 4 slices.

into 2 clusters, $S_4, \ldots, S_7$ are split into 4 clusters, and so on. We say a cluster C_i in slice S_i is a predecessor of another cluster C_j in slice S_j if $i < j$ and C_i is on the direct path to the root from C_j; we also say C_j is a successor of C_i. Throughout this section, sensors will only receive data from successive clusters and will only send data to preceding clusters or the root. Figure 4.6 compares the the slice model and splitting slice model on a sector with 4 slices.

The splitting slice model has some major advantages. First, the area of each cluster remains close to constant even as the radius goes to infinity, meaning in an evenly distributed network each cluster has about the same number of sensors. Secondly, each cluster has only one predecessor in each slice, which simplifies the process of using the short hops to send data to a predecessor. Finally, the distance between any sensor in a cluster and any sensor region in the cluster's predecessor is bounded by a fixed constant regardless of the total number of hops in the network.

A fundamental assumption we make in this section is that sensors are *evenly spaced*, by which we mean each slice has exactly n sensors and each cluster has exactly 1 or 2 sensors. Throughout the analysis we will also assume that $\theta = \frac{\pi}{2}$. Additionally, instead of n being the number of sensors in the entire network n will be the number of sensors in a single sector of the network and we will analyze the energy usage of

the sensors in this sector alone. These previous two assumptions about n and θ will not affect the statements of our main theorems since if $\alpha > 2$ these assumptions will impact the leading constant C and if $\alpha = 2$ the assumptions will impact the low order terms. As in the one-dimensional case the following are main theorems of this section and their proofs appear in Subections 4.3.1 and 4.3.2.

Theorem 32. *Let S be an evenly spaced sensor network with n sensors on a two-dimensional disk of radius R with energy cost $e_0 + e_t d^\alpha$ of sending unit data d distance away. If $\alpha > 2$ then the optimal data propagation on S takes*

$$C(\alpha, e_0, e_t)(\alpha - 2) \max\left(\frac{e_t R^\alpha}{n^{\frac{\alpha}{2}-1}},\, e_0^{1-\frac{2}{\alpha}} e_t^{\frac{2}{\alpha}} R^2\right) + l.o.t.$$

maximum energy per sensor, where C is a function of α, e_0, and e_t bounded above and below by positive constants. If $\alpha = 2$ then optimal data propagation on S takes

$$\frac{e_t R^2}{\ln(\min(n,\, R^2))} + l.o.t.$$

maximum energy per sensor.

The constants e_0, e_t, and α behave similarly in this theorem as in the 1D case of Theorem 30 but the network behaves much differently as n increases. As the number of sensors increases the energy actually decreases from $O(R^\alpha)$ to $O(R^2)$. The first term $\frac{e_t R^\alpha}{n^{\frac{\alpha}{2}-1}}$ in the first equation can be rewritten in the form $e_t \left(\frac{R}{\sqrt{n}}\right)^{\alpha-2} R^2$. It is then apparent that if more sensors are added to the network the energy usage decreases until the density reaches a constant value at which point the energy usage which is $O(R^2)$. Also notice that if $\alpha = 2$ the logarithmic factor in the denominator also depends upon the density of the network. If n is too small then the denominator becomes $\ln(n)$ but adding more sensors can only increase the term up to $\ln(R^2)$.

If $\alpha > 2$ and the density of the network is decreasing, that is $R = \omega(\sqrt{n})$, then the first term in the maximum function will dominate. However if the density of the network is increasing, that is $R = o(\sqrt{n})$, then the second term will dominate. If the density of the network is fixed at $\rho = \frac{n}{R^2}$ then either term may be larger depending on the exact constants. This gives us the following corollary which more closely resembles Theorem 30 with e_t replaced by $\frac{e_t}{\rho^{\frac{\alpha}{2}}}$.

Corollary 33. *Let S be an evenly spaced sensor network with n sensors on a two-dimensional disk with fixed density $\rho = \frac{n}{R^2}$ and energy cost $e_0 + e_t d^\alpha$ of sending unit data d distance away. If $\alpha > 2$ then the optimal data propagation on S takes*

$$C(\alpha, e_0, e_t)(\alpha - 2) \max\left(\frac{e_t}{\rho^{\frac{\alpha}{2}}}, \frac{e_0^{1-\frac{2}{\alpha}} e_t^{\frac{2}{\alpha}}}{\rho}\right) n + l.o.t.$$

maximum energy per sensor where C is a function of α, e_0, and e_t bounded above and below by positive constants. If $\alpha = 2$ then optimal data propagation on S takes

$$\frac{e_t n}{\rho \ln(n)} + l.o.t.$$

maximum energy per sensor.

The other main theorem of this section, as in the 1D case, states that EBP is close to optimal.

Theorem 34. *In a evenly spaced sensor network on a two-dimensional disk, energy balanced protocol (EBP) is within constant factor optimal and if $\alpha = 2$ is within low order terms of optimal.*

4.3.1 2-Dimensional Lower Bound

Our general approach to our lower bound for data propagation in the two-dimensional model will be similar to that of the one-dimensional case in Subsection 4.2.2. The major differences being that the distance data is being sent from the ith cluster is now at least $\frac{(i-1)R}{N}$ as opposed to i in the 1-dimensional model. With this in consideration our bound for maximum energy becomes the following

$$E \geq n \Big/ \sum_{i=1}^{N} \frac{i}{e_0 + e_t(i-1)^\alpha R^\alpha / N^\alpha} \tag{4.10}$$

The sensor in the first slice s may be arbitrarily close to the root. This causes problems if we attempt to replicate the analysis of Subsection 4.2.2. Thus we will modify Equation 4.10 based on the distance s is from the root. In the first case if s is at least $\frac{R}{2N}$ away from the root then we bound energy the usual way with the following

$$\begin{aligned} E &\geq n \Big/ \left(\frac{1}{e_0 + e_t R^\alpha / 2^\alpha N^\alpha} + \sum_{i=2}^{N} \frac{i}{e_0 + e_t(i-1)^\alpha R^\alpha / N^\alpha} \right) \\ &\geq n \Big/ \sum_{i=1}^{N} \frac{i}{e_0 + e_t R^\alpha / 2^\alpha N^\alpha} \end{aligned}$$

If the s is less than $\frac{R}{2N}$ away from the root, then we bound the energy to get all data to either r or s. For a sensor in slice i the closest that s or r could be is $\frac{\left(i-\frac{1}{2}\right)R}{N}$. Essentially, we are treating s as the new root and bounding the energy to send all data to s. The data generated by s does not need to be sent at all so only $n-1$

sensors need to transfer data to r or s. Thus we get the following bound

$$\begin{aligned} E &\geq (n-1) \Big/ \sum_{i=2}^{N} \frac{i}{e_0 + e_t(i-\frac{1}{2})^\alpha R^\alpha / N^\alpha} \\ &\geq (n-1) \Big/ \sum_{i=1}^{N} \frac{i}{e_0 + e_t R^\alpha 3^\alpha / 4^\alpha N^\alpha} \\ &\geq (n-1) \Big/ \sum_{i=1}^{N} \frac{i}{e_0 + e_t R^\alpha / 2^\alpha N^\alpha} \end{aligned} \tag{4.11}$$

These two bounds are exactly the same as Equation 4.4 with e_t replaced by $\frac{e_t R^\alpha}{2^\alpha N^\alpha}$, with the exception of the -1 term in the numerator of Equation 4.13 which will turn into low order terms. Thus using the same method as Subsection 4.2.2 if $\alpha > 2$ we get the following bounds

$$E \geq \frac{\alpha^{1-\frac{1}{\alpha-1}}}{2^3(\alpha-1)^{2-\frac{2}{\alpha}}}(\alpha-2){e_0}^{1-\frac{2}{\alpha}}{e_t}^{\frac{2}{\alpha}}R^2 - O\left(\frac{R^2}{n}\right)$$

$$E \geq \frac{1}{4^\alpha(\alpha-1)}(\alpha-2)\frac{e_t R^\alpha}{n^{\frac{\alpha}{2}-1}} - O\left(\frac{R^\alpha}{n^{\frac{\alpha}{2}}}\right)$$

This technique, however, is not tight enough to give a result within low order terms of optimal if $\alpha = 2$. We break the case where $\alpha = 2$ into two subcases depending on whether R or N is larger. The first case is if $R = \Omega(N)$ and we use both equations 4.12 and 4.13. If the first sensor is at least $\frac{R}{2N}$ away from the root then

$$\begin{aligned} \frac{1}{e_0 + e_t R^2/4N^2} + \sum_{i=2}^{N} \frac{i}{e_0 + e_t(i-1)^2 R^2/N^2} &\leq \sum_{i=1}^{N-1} \frac{i+1}{e_t i^2 R^2/N^2} + O\left(\frac{N^2}{R^2}\right) \\ &\leq \int_1^N \frac{N^2}{e_t x R^2} dx + O\left(\frac{N^2}{R^2}\right) \\ &= \frac{N^2}{e_t R^2}\ln(N) + O\left(\frac{N^2}{R^2}\right) \end{aligned}$$

If the first sensor is more than $\frac{R}{2N}$ then

$$\begin{aligned}\sum_{i=2}^{N} \frac{i}{e_0 + e_t(i-\frac{1}{2})^2 R^2/N^2} &\leq \int_{\frac{3}{2}}^{N-\frac{1}{2}} \frac{N^2}{e_t x R^2} dx + O\left(\frac{N^2}{R^2}\right) \\ &= \frac{N^2 \ln(N)}{e_t R^2} + O\left(\frac{N^2}{R^2}\right)\end{aligned}$$

Therefore in either case

$$\begin{aligned}E &\geq \frac{e_t R^2 n}{N^2 \ln(N)} - O\left(\frac{R^2}{\log^2(n)}\right) \\ &= \frac{e_t R^2}{\ln(n)} - O\left(\frac{R^2}{\log^2(n)}\right)\end{aligned}$$

When $\alpha = 2$ and $R = O(N)$ the simple bound from Equation 4.10 is enough. Using this equation we get the following bound

$$\begin{aligned}E &\geq n \Big/ \sum_{i=1}^{N} \frac{i}{e_0 + e_t(i-1)^2 R^2/N^2} \\ &= n \Big/ \left(\frac{1}{e_0} + \sum_{i=1}^{N-1} \frac{i+1}{e_0 + e_t i^2 R^2/N^2}\right) \\ &\geq n \Big/ \left(\frac{1}{e_0} + \int_1^{\frac{N}{R}} \frac{x}{e_0} dx + \int_{\frac{N}{R}}^{N} \frac{N^2}{e_t x R^2} dx + \int_1^N \frac{N^2}{e_t x^2 R^2} dx\right) \\ &= n \Big/ \left(\frac{N^2}{e_t R^2}\left(\ln(N) - \ln\left(\frac{N}{R}\right)\right) + O\left(\frac{N^2}{R^2}\right)\right) \\ &= \frac{e_t n R^2}{N^2\left(\ln(R) + O\left(1\right)\right)} \\ &= \frac{e_t R^2}{\ln(R^2)} - O\left(\frac{R^2}{\log^2(R)}\right)\end{aligned}$$

4.3.2 2-Dimensional EBP Upper Bound

This subsection analyzes the cost of using the Energy Balanced Protocol (EBP), which provides an upper bound for data propagation, in the two-dimensional network slice model. As in the 1D case each cluster will with some probability send data directly to the base station also called the root. The remaining data is passed along a short hop to another cluster of sensors closer to the root a fixed number of hops away, δ.

As in Subsection 4.2.3, we assume that δ evenly divides N, we only analyze the δ partition of the network which consists of clusters $S_\delta, S_{2\delta}, \ldots, S_N$, and we assume that the exact amount of data generated by each partition is $\frac{n}{\delta}$. As before, we use the data efficiency which is defined as $\eta_i = \frac{d_i}{E}$ where d_i is the amount of data sensors in cluster i send to the root. Under these assumptions the maximum energy usage of any sensor obeys the following bound.

$$E = n \Big/ \delta^2 \sum_{i=1}^{\frac{N}{\delta}} i\eta_{i\delta} \tag{4.12}$$

We can also bound the data efficiency similarly to Subsection 4.2.3 however working with the distance between sensors in different clusters is slightly more difficult. First notice that the distance between the inner edge and the outer edge of a cluster is between $\frac{R}{N}$ and $2\frac{R}{N}$ and the distance from a sensor in S_i to the root is between $\frac{(i-1)R}{N}$ and $\frac{iR}{N}$. Finally, the distance between a sensor in S_i and one of its predecessors in slice S_{i-1} can be generously bounded by $\frac{4R}{N}$. Also the number of sensors in S_i with the same predecessors in slice S_j can be bounded by $\frac{2j}{i}$ and there may be 2 sensors in the predecessor. These bounds give us the following bound on the data efficiency.

$$\eta_{i\delta} \geq \frac{1 - (e_0 + e_t\delta^\alpha 4^\alpha R^\alpha/N^\alpha)\sum_{j=1}^{i-1}\frac{4j}{i}\eta_{j\delta}}{e_0 + e_t i^\alpha \delta^\alpha R^\alpha/N^\alpha}$$
$$\geq \frac{1 - \frac{4(e_0+e_t\delta^\alpha 4^\alpha R^\alpha/N^\alpha)}{i}\sum_{j=1}^{i-1}\frac{j}{e_0+e_t(j-1)^\alpha\delta^\alpha R^\alpha/N^\alpha}}{e_0 + e_t i^\alpha \delta^\alpha R^\alpha/N^\alpha}$$

This expands Equation 4.12 into the following.

$$E \leq n \Big/ \delta^2 \sum_{i=1}^{\lfloor \frac{N}{\delta} \rfloor} \frac{i - 4\left(e_0 + e_t\delta^\alpha 4^\alpha R^\alpha/N^\alpha\right)\sum_{j=1}^{i-1}\frac{j}{e_0+e_t(j-1)^\alpha\delta^\alpha R^\alpha/N^\alpha}}{e_0 + e_t i^\alpha \delta^\alpha R^\alpha/N^\alpha} \tag{4.13}$$

If $\alpha = 2$ then Equation 4.13 simplifies to the following.

$$E \leq n \Big/ \delta^2 \sum_{i=1}^{\lfloor \frac{N}{\delta} \rfloor} \frac{i - O\left(\log(i)\right)}{e_0 + e_t i^2 \delta^2 R^2/N^2}$$

Now we approximate the sum using integration. However to bound this sum by an integral the terms of the sum must be monotonic. To this end we only consider the terms of the sum after $\frac{i}{e_0+e_t i^2\delta^2R^2/N^2}$ reaches its maximum value at $i = \left(\frac{e_0}{e_t}\right)^{\frac{1}{2}}\frac{N}{\delta R}$. Doing so we get the following bound.

$$E \leq n \Big/ \delta^2 \cdot \int_{\max\left(1, \left(\frac{e_0}{e_t}\right)^{\frac{1}{2}}\frac{N}{\delta R}\right)}^{\lfloor \frac{N}{\delta} \rfloor} \frac{x - O\left(\log(x)\right)}{e_0 + e_t x^2 \delta^2 R^2/N^2} dx$$
$$= nR^2 \Big/ N^2 \left(\ln\left(\frac{x\delta R}{N}\right) + O(1) \Big|_{\max\left(1, \left(\frac{e_0}{e_t}\right)^{\frac{1}{2}}\frac{N}{\delta R}\right)}^{\lfloor \frac{N}{\delta} \rfloor} \right)$$
$$= \frac{R^2}{2\min\left(\ln\left(\frac{N}{\delta}\right), \ln(R)\right) + O(1)} + O\left(\frac{1}{n^2}\right)$$
$$= \frac{R^2}{\ln\left(\min\left(\frac{n}{\delta^2}, R^2\right)\right)} + O\left(\frac{R^2}{\log^2\left(\min\left(\frac{n}{\delta^2}, R^2\right)\right)}\right)$$

Thus if $\alpha = 2$, as long as $\delta = O\left(\min\left(\frac{N}{R},\, 1\right)\right)$

$$E \leq \frac{R^2}{\ln\left(\min\left(n,\, R^2\right)\right)} + O\left(\frac{R^2}{\log^2\left(\min\left(n,\, R^2\right)\right)}\right)$$

On the other hand if $\alpha > 2$ a problem arises when trying to use the same strategy as in Subsection 4.2.3. It is no longer possible to always bound the term subtracted in the data efficiency by a constant less than 1 and for small values of i it may be possible for this term to be greater than 1. This may be the case, for example, if the sensor in the first layer very close to the root and the 2 sensors in the second layer are as far as possible from the root and $\delta = 1$.

To overcome this challenge we artificially limit $\eta_{i\delta}$ when i is small. That is, we assume the first say $1,000$ layers only send the data they generate to the root and do not receive any additional data from following layers. We define the data efficiency of a sensor in the ith layer for this modified strategy to be η'_i. Notice that $\sum_{i=1}^{N} i\eta_i \geq \sum_{i=1}^{N} i\eta'_i$. Doing so we can bound the data efficiency with the following.

$$\begin{aligned}
\eta'_{i\delta} &\geq \frac{1 - \left(e_0 + e_t\delta^\alpha 4^\alpha R^\alpha/N^\alpha\right) \sum_{j=1,000}^{i-1} \frac{4j}{i}\eta_{j\delta}}{e_0 + e_t i^\alpha \delta^\alpha R^\alpha/N^\alpha} \\
&\geq \frac{1 - \frac{4(e_0 + e_t\delta^\alpha 4^\alpha R^\alpha/N^\alpha)}{i} \sum_{j=1,000}^{i-1} \frac{j}{e_0 + e_t(j-1)^\alpha \delta^\alpha R^\alpha/N^\alpha}}{e_0 + e_t i^\alpha \delta^\alpha R^\alpha/N^\alpha}
\end{aligned}$$

We let $\delta = \left\lceil \left(\frac{e_0}{e_t}\right)^{\frac{1}{\alpha}} \frac{N}{R} \right\rceil$ and get the following generous bound.

$$\eta'_{i\delta} \geq \frac{1 - \frac{4(1+2^\alpha 4^\alpha)}{i} \sum_{j=1,000}^{i-1} \frac{j}{1+(j-1)^\alpha}}{e_0\left(1 + 2^\alpha i^\alpha\right)}$$

Then if say $j \geq 1,000$ then $\frac{j\left(4+2^{3\alpha+2}\right)}{1+(j-1)^{\alpha}} < \frac{1}{2}$ and thus $\eta'_{i\delta} \geq \frac{1}{2e_0(1+i^{\alpha})}$. Additionally, notice for sufficiently large N, $\sum_{i=1}^{\frac{N}{\delta}} \frac{1}{2e_0(1+i^{\alpha})} \leq c \cdot \sum_{i=1,000}^{\frac{N}{\delta}} \frac{1}{2e_0(1+i^{\alpha})}$ for some constant c. Thus following the same steps as in Subsection 4.2.3 we get the same lower bound as before with a larger leading coefficient and e_t replaced by $\frac{e_t R^{\alpha}}{n^{\frac{\alpha}{2}}}$. If $e_0 \leq \frac{e_t R^{\alpha}}{n^{\frac{\alpha}{2}}}$ then for some constant c

$$E \leq c \cdot (\alpha - 2) \frac{e_t R^{\alpha}}{n^{\frac{\alpha}{2}-1}}$$

On the other hand if $e_0 \geq \frac{e_t R^{\alpha}}{n^{\frac{\alpha}{2}}}$ then for some constant c

$$E \leq c \cdot (\alpha - 2) e_0^{1-\frac{2}{\alpha}} e_t^{\frac{2}{\alpha}} R^2 + O\left(R^{4-\alpha}\right)$$

This completes the proofs of Theorems 32 and 34.

Chapter V

Final Remarks

Energy dissipation is a primary concern in wireless sensor networks. Algorithms and protocols for wireless sensor networks should not only minimize runtime and average energy usage but also the maximum energy usage of any sensor. Sensors conserve energy by going into a low-energy sleep state in which they cannot communicate with other sensors or do meaningful computations. Coordinating networks of sensors that are asleep for the vast majority of the time provides unique challenges. Because of the spatial nature of sensor networks geometry problems are of particular interest. Sensor networks often have a high-energy base station and sensors will often propagate the data generated at each sensor to the base station.

In this dissertation we presented research on energy efficient algorithms for low-energy wireless sensor networks and data propagation. In Chapter II we focus on algorithms in single-hop networks. We then use single-hop networks as a building block for creating multi-hop algorithms in Chapter III. Finally, in Chapter IV we analyze the energy usage for optimally propagating data to the base station.

In Section 2.4 we discuss a problem that arises when executing randomized recursive algorithms in a single-hop sensor network. To overcome this problem we present the technique *breadth first recursion*. We use this technique in Section 2.5 to create a

sorting algorithm that is optimal within a constant factor both in terms of time and energy. We then analyze simulations of our sorting algorithm to show that it is much faster than previous algorithms and is competitive with previous algorithms in terms of energy usage. Then in Section 2.6 we use breadth first recursion to compute the 2-dimensional convex hull with an algorithm that is often sublinear depending upon the number of points in the convex hull and analyze simulations of this algorithm under different distributions.

In Section 3.2 we introduce *consolidation algorithms* as a method for executing algorithms for multi-hop networks using single-hop algorithms as a building block. Then in sections 3.3, 3.4, and 3.5 we create consolidation algorithms for all points nearest neighbor, coverage boundary, and the Voronoi diagram respectively.

In Section 4.2 we analyze data propagation in a simplified 1D model and show that short hop long hop approaches like the Energy Balanced Protocol (EBP) are close to optimal and show empirical results that compute the coefficient of the high order terms. We additionally show that only using short hops is only close to optimal if sending long distances is exceptionally expensive. In Section 4.3 we generalize some of these results to a 2D model and show that EBP is still close to optimal.

Several open questions relating to this work still remain.

- **Fault Tolerance** Because sensor networks consist of small low energy processors, faults in individual sensors or communication are relatively common [59]. Fault tolerance is already a well studied problem in the literature [19, 34, 42]. Techniques for fault tolerance are considered at every level of the stack from hardware, to communication protocols, to algorithms. To apply the techniques presented in Chapters II and III in realistic scenarios one must anticipate sensor failures. Is it possible to create fault tolerant versions of the breadth first

recursion framework and consolidation algorithms?

- **Data Propagation** There many open problems relating to analysis of the data propagation problem discussed in Chapter IV. In Section 4.3 we assumed that the sensors were placed at evenly distributed locations. Do these results also apply to the case where sensor locations are instead drawn from the uniform distribution? In this case there are logarithmic sized gaps in the network [23]. What results are achievable when sensors' locations are distributed in other manners, such as a Gaussian distribution? Here there are more sensors near the root, so data propagation should be more efficient. When the sensors' locations are chosen from some distribution, how does the distribution affect their energy usage?

Appendix

Extreme Statistics

We introduce the following definition to help describe random variables with properties that will be useful when examining extreme statistics.

Definition 35. *If $X(n) \in \mathbb{Z}^+$ is a random variable, $X(n)$ is said to have an **exponentially small tail** if there exist constants $k > 0$ and $c > 1$ such that for all $j > 0$ we have $P(X(n) = k\mathbb{E}[X(n)] + j) \leq c^{-j}$.*

In particular, negative binomial (and geometric) distributions have exponentially small tails. These random variables have the following property that will be useful for analyzing the behavior of the maximum values of random variables. This lemma is referenced throughout the dissertation and we note that the random variables in the lemma need not be independent.

Lemma 36. *If $X_1(n), \ldots, X_m(n) \in \mathbb{Z}^+$ are random variables with exponentially small tails, then*

$$\mathbb{E}[\max_i (X_i(n))] = O(\max_i (\mathbb{E}[\, X_i(n)\,]) + \log m)$$

with exponentially small tail.

Proof. Let $X_{\max}(n) = \max_i (\mathbb{E}[\, X_i(n)\,])$ and let $k > 0$ and $c > 1$ be the constants such

that for all $1 \leq i \leq m$ and $j > 0$ we have $P(X_i(n) = k\mathbb{E}[X_i(n)] + j) \leq c^{-j}$.

Then

$$\begin{aligned}
&P(\max_i(X_i(n)) = kX_{\max}(n) + j) \\
&\leq P(\exists i \text{ s.t. } X_i(n) \geq kX_{\max}(n) + j) \\
&\leq \sum_{i=1}^{m} P(X_i(n) \geq kX_{\max}(n) + j) \\
&\leq \sum_{i=1}^{m} \sum_{l=j}^{\infty} (1/c)^l \\
&= \frac{m}{1 - 1/c}(1/c)^j \\
&= (1/c)^{j - \log_c\left(\frac{m}{1-1/c}\right)}
\end{aligned}$$

and

$$P\left(\max_i(X_i(n)) = kX_{\max}(n) + \log_c\left(\tfrac{m}{1-1/c}\right) + j\right) \leq (1/c)^j$$

From this probability bound it directly follows that

$$\mathbb{E}[\max_i(X_i(n))] = O\left(X_{\max}(n) + \log(m)\right)$$

which completes the proof. □

Bibliography

[1] N. Ab Aziz, A. Mohemmed, and M. Alias, "A wireless sensor network coverage optimization algorithm based on particle swarm optimization and Voronoi diagram," *IEEE international conference on networking, sensing and control* (2009), pp. 602–607.

[2] A. Abbasi, M. Younis, and U. Baroudi, "Recovering from a node failure in wireless sensor-actor networks with minimal topology changes," *IEEE Transactions on vehicular technology* 62.1 (2012), pp. 256-271.

[3] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. Yap, "Parallel computational geometry," *Algorithmica* (1988), pp. 293–327.

[4] K. Akkaya and M. Younis. "A survey on routing protocols for wireless sensor networks," *Ad Hoc Networks 3.3*(2005), pp. 325–349.

[5] J. Al-Karaki and A. Kamal, "Routing techniques in wireless sensor networks: a survey," *IEEE Wireless Communications* (2004), pp. 6–28.

[6] W. Alsalih, K. Islam, Y Núñez-Rodríguez, and H. Xiao, "Distributed Voronoi diagram computation in wireless sensor networks," *SPAA* (2008), pp. 364–364.

[7] J. Bentley, "Multidimensional divide-and-conquer," *Communications of the ACM* 23.4 (1980), pp. 214–229.

[8] O. Boyinbode, H. Le, A. Mbogho, M. Takizawa, and R. Poliah, "A survey on clustering algorithms for wireless sensor networks," *IEEE International conference on network-based information systems* 13 (2010), pp. 358-364.

[9] Y. Chang, T. Kopelowitz, S. Pettie, R. Wang, and W. Zhan, "Exponential separations in the energy complexity of leader election," *ACM SIGACT Symposium on Theory of Computing* 49 (2017), pp. 771-783.

[10] I. Chatzigiannakis, T. Dimitriou, S. Nikoletseas, and P. Spirakis, "A probabilistic algorithm for efficient and robust data propagation in wireless sensor networks," *Ad Hoc Networks* 4.5 (2006), pp. 621–635.

[11] X. Chen, K. Makki, K. Yen, and N. Pissinou, "Sensor network security: A survey," *IEEE Communications Surveys and Tutorials* 11.2 (2009), pp. 52-73.

[12] T. Chowdhury, C. Elkin, V. Devabhaktuni, D. Rawat, and J. Oluoch, "Advances on localization techniques for wireless sensor networks: A survey," Computer Networks 110 (2016), pp. 284-305.

[13] N. Dadoun and D. Kirkpatrick, "Parallel processing for efficient subdivision search," *Proceedings of the third annual symposium on computational Geometry* (1987), pp. 205–214.

[14] A. Dementyev, S. Hodges, S. Taylor, and J. Smith, "Power consumption analysis of Bluetooth Low Energy, ZigBee and ANT sensor nodes in a cyclic sleep scenario," *Intl. Wireless Symp.* (2013), pp. 1–4.

[15] C. Efthymiou, S. Nikoletseas, and J. Rolim, "Energy balanced data propagation in wireless sensor networks," *Wireless Networks* 12.6 (2006), pp. 691–707.

[16] D. Evans, "On parallel computation of Voronoi diagrams," *Parallel Computing* 12.1 (1989), pp. 121–125.

[17] M. Farooq, A. Dogar, and G. Shah, "MR-LEACH: multi-hop routing with low energy adaptive clustering hierarchy," *IEEE International Conference on Sensor Technologies and Applications* 4 (2010), pp. 262–268.

[18] A. Fawzy, H. Mokhtar, O. Hegazy, "Outliers detection and classification in wireless sensor networks," *Egyptian Informatics Journal 14.2*, 2013, pp. 157-164.

[19] G. Gupta and M. Younis, "Fault-tolerant clustering of wireless sensor networks," *Wireless Communications and Networking* 3 (2003), pp. 1579–1584.

[20] J. Hart and K. Martinez, " Environmental sensor networks," *IEEE Comput.* 37.8 (2004), pp. 50–56.

[21] C. He, Z. Feng, and Z. Ren, "Distributed Algorithm for Voronoi Partition of Wireless Sensor Networks with a Limited Sensing Range," Sensors 18.2 (2018), pp. 446-466.

[22] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," *Proceedings of the 33rd annual Hawaii international conference on IEEE* (2000), pp. 3005–3014.

[23] L. Holst, "On the lengths of the pieces of a stick broken at random," *Journal of Applied Probability* 17.3 (1980), pp. 623–634.

[24] C. Huang and Y. Tseng, "The coverage problem in a wireless sensor network," *Mobi. Netw. & Appl.* 10.4 (2005), pp. 519–528.

[25] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for Sensor Network," *MOBICOM* (2000), pp. 56–67.

[26] A. Jarry, P. Leone, S. Nikoletseas, and J. Rolim, "Optimal data gathering paths and energy-balance mechanisms in wireless networks," *Ad Hoc Netw.* 9.6 (2011), pp. 1036–1048.

[27] E. Júnior, P. Câmara, L. Vieira, and M. Vieira, "3DVS: Node scheduling in underwater sensor networks using 3D voronoi diagrams," *Computer Networks* 159 (2019), pp. 73-83.

[28] J. Kahn, R. Katz, and K. Pister, "Next century challenges: mobile networking for Smart Dust," *Mobi. Comput. & Netw.* 5 (1999), pp. 271–278.

[29] R. Kershner, "The number of circles covering a set," *American Journal of Mathematics* 61.3 (1939), pp. 665–671.

[30] M. Kik, "Merging and merge-sort in a single hop radio network," *Theory & Pract. of Comp. Sci.* (2006), pp. 341–349.

[31] D. Kirkpatrick, "Optimal search in planar subdivisions," *SIAM Journal on Computing* 12.1 (1983), pp. 28–35.

[32] D. Kirpatrick and R. Seidel, "The ultimate planar convex hull algorithm?", *SIAM J. Comput.* 15.2 (1986), pp. 287–299.

[33] E. Knox and R. Ng, "Algorithms for mining distance-based outliers in large datasets," *Proceedings of the international conference on very large data bases* (1998), pp. 392–403.

[34] B. Krishnamachar and S. Iyengar, "Distributed Bayesian algorithms for fault-tolerant event region detection in wireless sensor networks," *IEEE Transactions on Computers* 53.3 (2004), pp. 241–250.

[35] Q. Li, M. De Rosa, and D. Rus, "Distributed algorithms for guiding navigation across a sensor network," *Mobi. Comput. & Netw.* 9 (2003), pp. 313–325.

[36] H. Lingxuan and D. Evans, "Localization for mobile sensor networks," *Mobi. Comput. & Netw.* 10 (2004), pp. 45–57.

[37] P. MacKenzie and Q. Stout, "Ultra-fast expected time parallel algorithms," *J. Algorithms* 26 (1998), pp. 1–33.

[38] C. Martel, "Maximum finding on a multiple access broadcast network," *Information Processing Letters* 52.1 (1994), pp. 7–13.

[39] J. Miranda, R. Abrishambaf, T. Gomes, P. Gonçalves, J. Cabral, A. Tavares, and J. Monteiro,. "Path loss exponent analysis in wireless sensor networks: Experimental evaluation," *IEEE International Conference on Industrial Informatics* 11 (2013), pp. 54-58.

[40] L. Mottola and G. P. Picco, "Programming wireless sensor networks: Fundamental concepts and state of the art," *ACM Comput. Surveys* 43.3 (2011) pp. 19.

[41] C. Otto, A. Milenkovic, C. Sanders, and E. Jovanov, "System architecture of a wireless body area sensor network for ubiquitous health monitoring," *J. Mobi. Multi.* 1.4 (2006), pp. 307–326.

[42] L. Paradis and Q. Han, "A survey of fault management in wireless sensor networks," *Journal of Network and Systems Management* 15.2 (2007), pp. 171–190.

[43] O. Powell, L. Pierre, and J. Rolim, "Energy optimal data propagation in wireless sensor networks," *Journal of Parallel and Distributed Computing* 67.3 (2007), pp. 302–317.

[44] F. Preparata and M. Shamos, "Computational geometry, an introduction," *Springer, New York* (1985).

[45] M. Rabbat and R. Nowak, "Distributed optimization in sensor networks," *International Symposium on Information Processing in Sensor Networks* 3 (2004), pp.20-27.

[46] H. Raynaud, "Sur l'envelope convexe des nauges des points aléatoires dans $\mathbb{R}^n$," *Intl. J. Appl. Prob.* 7.1 (1970), pp. 35–48.

[47] B. Reed, "The height of a random binary search tree," *J. ACM* 50.3 (2003), pp. 306–332.

[48] M. Serna, A. Bermudez, R. Casado, and P. Kulakoski, "A convex hull-based approximation of forest fire shape with distributed wireless sensor networks," *Intl. Sens., Sens. Netw.& Info. Proc.* (2011), pp. 419–424.

[49] M. Singh and V. Prasanna, "Energy-optimal and energy-balanced sorting in a single-hop wireless sensor network," *Perv. Comput. and Comm.* (2003).

[50] A. So and Y. Ye, "On solving coverage problems in a wireless sensor network using Voronoi diagrams", *WINE* (2005), pp. 584–593.

[51] A. So and Y. Ye, "Theory of semidefinite programming for sensor network localization," *Math. Prog.* 109.2–3 (2007), pp. 367–384.

[52] B. Sundararaman, U. Buy, and A. Kshemkalyani, "Clock synchronization for wireless sensor networks: a survey," *Ad Hoc Networks* 3.3 (2005), pp. 281–323.

[53] B. White, A. Tsourdos, I. Ashokaraj, S. Subchan, and R. Zbikowski "Contaminant cloud boundary monitoring using network of UAV sensors," *IEEE Sensors Journal* 8.10 (2008), pp. 1681-1692.

[54] M. Winkler, K. D. Tuchs, K. Hughes, and G. Barclay, "Theoretical and practical aspects of military wireless sensor networks," *J. Telecom. & Inf. Tec.* (2008), pp. 37–45.

[55] F. Xiangning and S. Yulin, "Improvement on LEACH protocol of wireless sensor network," *SensorComm* (2007), pp. 260–264.

[56] R. Xu, H. Da, F. Wang, and Z. Jia, "A convex hull based optimization to reduce the data delivery latency of the mobile elements in wireless sensor networks," *IEEE Embedded & Ubiq. Comput.* 10 (2013), pp. 2245–2252.

[57] W. Ye, J. Heidemann, and D. Estrin, "Medium access control with coordinated adaptive sleeping for wireless sensor networks," *IEEE/ACM Transactions on Networking* 12.3 (2004), pp. 493-506.

[58] C. Zhang, Y. Zhang, and Y. Fang, "Localized algorithms for coverage boundary detection in wireless sensor networks," *Wireless Networks* 15.1 (2009), pp. 3–20.

[59] J. Zhao, and R. Govindan, "Understanding packet delivery performance in dense wireless sensor networks," *Proceedings of the 1st international conference on Embedded networked sensor systems* (2003), pp. 1–13.

[60] G. Zhou, T. He, J. Stankovic, and T. Abdelzaher, "RID: radio interference detection in wireless sensor networks," *IEEE INFOCOM* 24.2 (2005), pp. 891–901.