Acknowledgements

I would like to thank my supervisors Dr. Selim G. Akl and Dr. David Rappaport for their high expectations, support, guidance, and immeasurable help. Without them the completion of this thesis would have been impossible.

I would also like to thank the entire Parallel and Unconventional Research Group, for their help and support throughout this project.

Thanks to my family and friends, whose contributions have been too many and too great to summarize. In particular, I would like to thank my parents: Kate and Larry Bentley for their assistance and support over all the years of my life.

Finally, I would like to thank Lena Dolman for her love and support throughout the course of my thesis, and also for helping push me to achieve my goals.
Abstract

In this thesis we directly consider an object tracking problem for wireless sensor networks (WSNs), called track persistence. Track persistence temporally extends the problem of object tracking by seeking to store and retrieve the entire history of an object. To provide an initial solution to track persistence, we develop two distinct algorithms. The first algorithm, update to sink, translates track persistence into a centralized problem. The second algorithm, a linked list-like algorithm, builds a dynamic data structure as the object traverses the network, and rebuilds the object history distributively upon demand. We conduct worst case analysis upon both of these algorithms. Finally, we implement a simulation environment and run a number of tests upon both algorithms. Track persistence is a very challenging problem, and this thesis contributes a pair of solutions which stand as a basis for future research.
Table of Contents

Acknowledgements i
Abstract ii
Table of Contents iii
List of Tables vi
List of Figures vii

Chapter 1:
Introduction ......................................................... 1
1.1 WSN General Features ........................................ 1
1.2 Difficulties and Limitations of WSNs ......................... 3
1.3 The WSN Tracking Problem .................................. 3
1.4 The WSN Track Persistence Problem ......................... 4
1.5 Contributions .................................................. 4
1.6 Outline ....................................................... 5

Chapter 2:
Background ......................................................... 6
2.1 Setup Phase .................................................... 7
2.2 Operation Phase ............................................... 13
2.3 Complete Systems ............................................. 19
2.4 Conclusion .................................................... 21
Chapter 3:

**Problem Specification** ........................................... 22

3.1 Definitions ......................................................... 22
3.2 Assumptions ....................................................... 22
3.3 Track Persistence .................................................. 23

Chapter 4:

**Algorithms** .......................................................... 26

4.1 Specification ....................................................... 26
4.2 Update to Sink ..................................................... 28
4.3 Linked List .......................................................... 31

Chapter 5:

**Algorithmic Analysis** ............................................. 42

5.1 Definitions & Preliminaries ....................................... 42
5.2 The Algorithmic Cost ............................................. 46

Chapter 6:

**Experimental Setup** ............................................... 51

6.1 Variables ........................................................... 51
6.2 Assumptions ........................................................ 52
6.3 Simulations .......................................................... 54

Chapter 7:

**Future Work & Open Problems** ................................. 61

7.1 Fault Tolerance & Reliability .................................... 61
7.2 Hierarchy ............................................................. 62
7.3 Mobile Agents ....................................................... 63
7.4 Co-operative Targets .............................................. 64
7.5 Multiple Covering Sets ........................................... 64
7.6 Collaborative Aggregation ......................................... 64
7.7 Co-operative Query Response Balancing on Linked Lists .................. 65
7.8 Dynamic Query Positions .................................................. 65
7.9 Multiple Object Tracking ................................................... 65
7.10 Partial Track Recovery ..................................................... 66

Chapter 8:

Conclusion ................................................................. 67
  8.1 Summary of Results .................................................... 67
  8.2 Contributions .......................................................... 67
  8.3 Finally ................................................................. 68

Appendix A:

Increasing Network Size & Complexity ................................. 69
  A.1 Update Costs ........................................................... 69
  A.2 Query Costs ............................................................ 70

Appendix B:

Increasing Track Length ................................................... 73
  B.1 Update Costs ........................................................... 73
  B.2 Query Costs ............................................................ 74

Bibliography ............................................................... 78
List of Tables

4.1 The state of the linked list at time $t_6$ ........................... 37

5.1 Summary of the algorithm’s analysis ............................... 50
List of Figures

4.1 Small sample sensor network ........................................ 28
4.2 Short example run .................................................. 40
4.3 The state of the linked list at time $t_6$ ................................ 41

5.1 If $v, w$’s sensing regions are overlapping, they can communicate. .......................... 43
5.2 An example of an average case corridor as described in the proof of Lemma 1. ........ 44

6.1 Comparison of query and update costs for networks of increasing size and complexity 56
6.2 Comparison of summed query and update costs for increasing track lengths .......... 58
6.3 Comparison of summed query and update costs for increasing query frequency ...... 60

A.1 Comparison of update costs as network size and complexity increase ................ 70
A.2 Comparison of query costs as network size and complexity increase .................. 71
A.3 Comparison of average update and query cost as network size and complexity increase 71
A.4 Comparison of the number of nodes involved in updating and querying as network size and complexity increase .................................................. 72

B.1 Comparison of update costs as track length increases ................................. 74
B.2 Comparison of query costs as track length increases ..................................... 75
B.3 Comparison of average query and update costs as track length increases ........... 76
B.4 A gradient histogram showing the average update cost of the update to sink algorithm. To read this graph, consider a battery value, track length pair. The shade of the graph at that point describes the quantity of experimental runs that resulted in that battery cost, at that track length .................................................. 76
B.5 Comparison of the number of nodes involved in updating and querying as track length 77
Chapter 1

Introduction

Several advances in hardware technology have led to the increased miniaturization of all computing systems. This miniaturization has prompted the development of ultra-large networks of wireless sensing units called Wireless Sensor Networks (WSNs).

1.1 WSN General Features

More completely, a WSN is a network of sensing nodes arranged in some physical space. Each sensing node (we will use node and sensing node interchangeably) has the capability to sense some feature from its environment, to send and receive messages (with both other sensing nodes, and external systems), to store some information in an internal memory, to process data, and (in some cases) to detect its own location. A WSN has one or more nodes, which are able to communicate directly with an external authority; in the case that only one such node exists, we call it the sink.

WSNs are physically deployed in an environment, and each node is left to detect its neighbouring nodes, and build a network in an ad-hoc manner. There are several potential techniques for deployment of WSNs, which include: individual manual placement of every node, group manual placement (nodes are connected by some material to ensure a grid-like arrangement, and are placed many at a time), group automatic deployment (nodes are dropped from an aerial craft, from boats, or using
some other massive deployment technique), or some combination of these methods (where important nodes are placed manually, but less important nodes are deployed in a less time consuming manner) [9, 6, 2]. More detail on deployment techniques is given in Chapter 2.

A node in a WSN detects its neighbours through whatever communication device is available to it. Typically radios are used for communication, but optical devices also can be used. Usually, communication is assumed to have a circular range of some radius; they can be modeled as unit disc graphs, with a vertex occurring on each sensing node in a plane scaled by the communication radius. This model is a vast simplification of the complexity of the variety of communication systems available. A lot of recent research has been into low power radio systems, both their transmission and reception. Often the communication overhead is the largest cost to sensor networks, but it is unclear how much hardware improvement will occur in this context. Chapter 2 provides more details about communication systems.

There is a lot of variety in the types of sensors used in a WSN. Sensors can detect different types of data: audio data, seismic data, light data, temperature data, and so on (See [2] for a more complete list). The correct type of sensor for a particular application is dependent on the goals, and environment of the network.

We call the region that a sensor can detect its sensing region. This region varies depending on a number of factors: what type of sensing is being done, the power of the sensor, the type of sensor, and more. For example, some sensors detect the directional movement of an object, either towards the sensor or away, and these effectively have a sensing region of the entire deployment area. On the other hand, a microphone-based sensor will have a circular sensing region of some radius around the microphone. There are a variety of sensors, and thus a variety of regions which they may sense.

WSNs have seen applications in health care, defense vehicle monitoring [5, 25], city planning and monitoring, habitat monitoring [15, 41], weather pattern monitoring, and more. There is great potential for WSNs, and they are an area of great research interest. As the quality of low level algorithms and protocols increases, and the miniaturization of hardware continues, we will see WSNs become more and more ubiquitous.
1.2 Difficulties and Limitations of WSNs

WSNs have several limitations, primarily their battery life [2]. In most cases, nodes in a WSN have a limited battery, which is non-replaceable. It is important that the algorithms employed on WSNs be aware of this limitation, and take the following precautions to help limit the battery use.

1. Limit the frequency of high drain activities, chiefly communication.

2. Limit the cost of each communication. The cost of communication is dependent upon two factors, size of the message, and the distance of transmission.

3. Cycle the nodes in and out of sleep modes, so that coverage of the network is maintained, but drain is balanced between nodes, and reduced across the network.

Another difficulty for WSNs are frequent node failures, which can occur in one of two ways. The node can fail in such a way that it ceases to respond to communications. This case occurs if the battery runs out, or the radio fails. If a node stops responding to communications we can detect this easily. However, if a node’s sensor fails (i.e. starts detecting incorrect information, or gets stuck reporting sensing data from some past event) this can be more difficult to detect.

Network reliability, bandwidth overloading, poor sensing quality, and unpredictable physical phenomena, are just a few of the numerous other difficulties and limitations we encounter when working with WSNs. Many techniques have been developed to handle these problems, and there is an active research community working to better solve and cope with these problems.

1.3 The WSN Tracking Problem

To support many of the applications of WSNs, robust, efficient, distributed algorithms for tracking objects (or, synonymously, targets) moving through the physical space of the network have been developed. The advantages of doing tracking in a WSN include cost effectiveness, location flexibility (networks can be deployed in any location, even if transportation to the area is difficult), robustness of tracking, and long-lifetimes of networks. However, achieving these last two goals has proven to be a very challenging problem, which is still being researched and improved upon.
CHAPTER 1. INTRODUCTION

1.4 The WSN Track Persistence Problem

The focus of the literature has been upon tracking an object for an instantaneous query operation, demanding the current location of an object. Distributed data structures have been developed and maintained explicitly for handling these tasks. The focus of this work is a related problem: tracking an object for the sake of a path query operation, demanding the complete historical location information of an object. This is a problem of data aggregation, a balance between centralized and distributed storage. This thesis is the first known document to consider this problem directly.

One motivation for considering a temporal extension of the target tracking problem, is essentially that of documentation. For example, an experiment may be designed to determine the favorite nesting areas of a reclusive animal. This animal is known to adjust its behaviour if it has interaction with human parties. A WSN is therefore deployed in the animal’s habitat, and the network tracks the movements of the animal over the course of several weeks. All of the tracking data is eventually shipped to the external authority, and the animal’s movements are then documented. It is easy to imagine a number of similar experiments that would require such documentation.

This thesis considers only networks which interact with one object at a time. Single object interaction drastically simplifies the problem of tracking the objects, (in Chapter 2 many of the problems associated with multiple object tracking are outlined) however many applications are only possible if we allow multiple object interaction. Track persistence is a difficult problem, to which this thesis is only a first attempt.

1.5 Contributions

The contributions of this thesis are:

1. The identification and definition of a novel WSN problem: Track Persistence. Track persistence is novel, because it extends the traditional object tracking problem and considers the full history of an object, rather than its instantaneous location. It is very likely that groups have been interested in the long term data of tracking algorithms in the past, but this is the first
known work to address the problem directly.

2. A solution, which effectively converts track persistence into a centralized problem. All information about an object is funneled to a central location (a sink) as time passes, allowing for this information to be immediately available whenever it is desired.

3. A second solution, which builds a dynamic, distributed data structure that better balances the cost of track persistence across the network. When the information is desired in this scheme, it is reconstructed by traversing the data structure.

1.6 Outline

In Chapter 2 we will give a thorough background of the research in target tracking using WSN. The primary interest of the chapter will be to identify problems related to object tracking and track persistence, before discussing solutions available from the literature. Chapter 3 formally defines what we mean by a WSN, and also thoroughly outlines track persistence as a cost minimization problem. Chapter 4 define the algorithms proposed in this thesis to solve track persistence. Chapter 5 shows some analysis of these algorithms, in the worst case, and also in a particular case. Chapter 6 gives some details of the simulation environment developed as a part of this thesis as well as showing the results of several experiments that were designed and run on the proposed algorithms. Chapter 7 identifies a number of open problems and future work for the problem of track persistence using WSNs. Finally, Chapter 8 relates the conclusions gained from this work.
Chapter 2

Background

In many disciplines, it is desirable to be able to track objects. Tracking vehicles as they move through diverse environments [37, 25]; tracking objects moving through secure areas [52, 43]; and monitoring animal populations and habitats [2, 15, 41] are a few examples. Typically, tracking is an expensive problem with a variety of limitations upon its use, and this has prohibited its utilization [5]. Tracking an object typically involves establishing observation agents, monitoring these agents, and maintaining the agents. The agents vary among humans, cameras, motion sensors, et cetera. Due to the limited success of computer monitoring systems, monitoring of agents is done manually, either locally or remotely, usually by human parties. Maintenance, too, is done almost exclusively by humans, and is expensive. Since the cost of maintenance in many environments is high, fewer sensors that fail infrequently are preferred to more sensors that fail regularly.

WSN may be a better solution for this problem, however. WSNs are inexpensive to deploy, have relatively long potential lifetimes, and require little monitoring. However, tracking objects with WSN presents its own suite of challenges. The lifetime of a WSN is separated into two distinct periods, setup and operation. During setup, a number of sub-tasks may or may not occur: deployment of the network, localization of the nodes, clock synchronization of the nodes, and internal data-structure development. Operation of an object tracking network has the following sub-tasks: target localization, target classification, and target information reporting. WSNs solve these sub-tasks in various ways, balancing quality of service with energy optimizations for maximization of lifetime.
This Chapter will survey some of the solutions to these challenges.

2.1 Setup Phase

The setup phase is made up of deployment, localization, clock synchronization, and internal data-structure development. Not every WSN will do all these steps, and the order of their completion may vary.

2.1.1 Deployment

Deployment of nodes in a WSN is an important problem. WSNs have extensive and often overlapping deployment concerns: connectedness of the network, coverage of the network, approximation of a grid layout, uniform distribution of nodes, uniform distribution of different types of nodes, and line of sight between nodes, are only a few of the potentially important considerations when deploying a WSN.

One of the distinct advantages of WSNs is their capacity for deployment in dangerous or difficult terrain. For example, there are many environments which are accessible during some months of the year, but which become inaccessible during other months; in these environments, a WSN could be deployed during the seasonal months, and retain their functionality through the less seasonal months [41]. Some environments may never be accessible by ground or may be too dangerous for humans to enter, and in these cases, aerial deployment may be possible [21].

- Mainwaring et al. deployed a thirty two node WSN on Great Duck Island [41]. Nodes were deployed manually, and left for a six month lifetime. The authors note that a significant advantage is presented by being able to deploy networks before a sensitive period (breeding seasons for example), and leave them to collect data un-intrusively during this period.

- In 2001, the feasibility of aerial deployment was demonstrated [5]. The nodes were deployed by an unmanned aerial vehicle, to track vehicles traveling on a dirt road. A series of further studies were completed in 2004, where nodes were placed manually in a grid-like arrangement except for along two roads, where all nodes which would have landed on the road were pushed
to the side. This second experiment was repeated a dozen times, and the deployment was intended to simulate aerial deployment.

The cost of building WSNs is still often prohibitively high for many small-scale studies, and good simulation tools are vital for the development and comparison of different algorithms. A simulation tool for WSN deployment called GENSEN attempts to provide a pseudo-realistic deployment simulation [14]. GENSEN is preceded by Topo_gen, which allows for random or clustered deployment, but does not consider real in-field deployment. Camilo et al. define six deployment strategies for WSN: grid, one-by-one, two-by-two, three-by-three, cliff, and propellant. Grid, one-by-one, two-by-two, and three-by-three are all manual deployment strategies. Cliff and propellant are automated deployment strategies, meant to simulate dropping the sensor nodes into a field. Experimental data shows that grid provides the best node placement, but is the most expensive; cliff and propellant are the least expensive, but they tend to group nodes near the center of the network. The others fall somewhere in the middle, highlighting a trade-off between deployment cost, and deployment quality.

Overall, deployment in practice is a problematic area of WSNs. As a result, the majority of WSN projects handle deployment by manually placing nodes [5, 6, 44, 41, 49, 53]. A good overview of a measurement assisted placement schema can be seen in [6].

2.1.2 Localization

Localization is a term borrowed from the robotics literature [12], which refers to the discovery of the physical location of an agent in the world. It is adapted to WSN research to refer to the task of discovering the physical location of all nodes in the network. Conventional localization equipment, like Global Navigation Satellite Systems (GNSSs), are expensive, large, and suffer from placement limitations due to their line-of-sight requirements [8]. More recently, indoor localization techniques have also been proposed [54], however, they require large investment in the environment surrounding the WSN. These limitations make conventional localization inappropriate for all nodes within a WSN. Instead, algorithms which allow for localization of sensor nodes based upon the pre-localization (obtained conventionally) of a subset of nodes, and the distances between nodes have been developed. It is extremely important to know node positions for many applications of WSN, and for tracking applications it is especially important.
Aspnes et al. consider the complexity of computing localization of WSNs in [8]. The main result is that localizing sparse networks, which they model as unit disc graphs is NP-hard, because it can be reduced in polynomial time from the circuit satisfiability problem. This contribution considers centralized algorithms for localizing sparse networks. So it is safe to assume that there is no efficient algorithm for localizing a sparse network. Thankfully, the networks required for object tracking are dense.

Peer-to-peer distance data can be calculated in several different ways [11]. The first technique looks at the received signal strength of messages from a node. Signal strength decreases with distance, and the distance can be calculated according to the propagation loss model. The second is done by measuring the time it takes for a message to pass from one node to another. This time is multiplied by the speed of light, and a distance is the result. Of the two, received signal strength is easier to calculate, but the timing model can be more accurate. A third technique is called time difference of arrival, which requires that a node be able to send two or more types of transmissions at once. The arrival time of the different transmissions are taken, and knowing each one’s propagation speed (the speed of light for radio, or the speed of sound for ultrasound), the distance can be calculated. If high accuracy is a concern, several samples can be taken, and their results averaged.

There are four main categorizations for localization algorithms [23]:

1. **Anchoring** - Anchored localization requires a subset of the nodes to be previously localized. Anchor-free localization proceeds without this.

2. **Propagation** - Incremental localization algorithms localize increasingly bigger and bigger subsets of the network, until the entire network is localized. Concurrent localization algorithms localize all nodes simultaneously, and then increase refinement.

3. **Granularity** - Fine grain localization algorithms localize a node to a point. Coarse grain localization algorithms localize a node to an area in the plane (for example, the intersection of a group of half-planes).

4. **Computation** - Centralized localization algorithms collect the full set of information, and process it centrally before re-transmitting it to the nodes. Distributed localization algorithms localize in-network, with no node knowing complete information.
We feel that there is a fifth categorization that should be added to this list:

5. **Determinism** - High determinism algorithms are those based upon only concrete position and distance information. Low determinism algorithms are those which rely also upon belief values, and pseudo-random sampling techniques.

Determinism is an effective categorization, because it represents a fundamental difference in paradigms between localization algorithms. It is also likely that there will be two fine grain, anchored, concurrently propagated, distributed algorithms, which differ distinctly in their determinism.

- Bulusu et al. propose an anchored, concurrent propagation, coarse-grained, distributed, and deterministic localization method for WSNs in [12]. The group initially attempted distance calculation using received signal strength, however the hardware used made this impossible. Instead, connectivity-based localization was used. Each radio was assumed to have an ideal circular transmission region. Each node with known location beacons its location, and listening non-localized nodes localize themselves to the average of all the locations received. The validity of the method is proven, and they provide some initial results. The primary problem with using this technique is that it requires that a very large number of nodes have known locations. Localization quality degradation is found in scenarios where the nodes cannot be placed in a uniform-grid layout. The simulations done by Bulusu et al. indicate that using this technique causes a 15% quality drop.

- Doherty et al. are able to achieve similar results to a naive beacon system (similar to [12]’s system) with roughly a third of the number of pre-localized nodes, by using convex optimization [20]. This is an anchored, concurrent propagation, coarse-grained, centrally computed, and deterministic algorithm. The results are based upon convex optimizations, which is an emerging mathematical minimization technique related to linear programming, and semi-definite programming. The only distance information used in this system is that of the constraint model’, which says that if two nodes can communicate, they are within the range set by their transmitters. It is shown that allowing the use of dynamically resize-able transmission radii can increase the granularity of the solution set. A disadvantage to this technique is that it only performs well when the anchor nodes are distributed near the borders of the network.
Albowicz et al. adopt a recursive scheme in [3], working on an extremely dense network with only a few pre-localized nodes. This algorithm is anchored, uses incremental propagation, is fine-grained, uses distributed computation, and is deterministic. Nodes predict their position using local information, before iteratively improving the accuracy of their position estimate. The full protocol adopted by Albowicz et al. consists of four phases. Each node initially determines its reference points, the set of points with known position. A node is considered to have known position if its position certainty value is high. A node with a GPS in it has 100% certainty. Once references are established, a node obtains both position estimates and certainty values from all of its reference nodes. The nodes then produce a position estimate of their own position, including a certainty value. The certainty is determined by the certainty of the references used to determine the position estimate. Finally, if a certainty threshold is met by the node, it begins to advertise itself as a potential reference to other nodes. References are ranked based upon their certainty, and when choosing references, those with the highest certainty are chosen. Position estimation is achieved by completing a Taylor Series approximation upon the non-linear system created by combining the distance information, and the references’ positions. It is important to avoid oscillation of this series, to ensure that the approximations do not advertise unless they fall within a certain confidence. Simulation results for this method are encouraging.

Another centralized solution to localization is proposed by Biswas et al in [10], which uses semi-definite programming, and achieves good results with fewer pre-localized nodes than even Doherty et al.’s solution. The main differences between Doherty et al.’s solution and Biswas et al.’s is that the latter solution uses full distance information instead of the constraint model’. Biswas et al. does not use the convex optimization used by Doherty et al., but instead relax the quadratic constraints of the distance information into linear constraints. The performance of this algorithm at the time of publishing is undesirable, however. Significant delay is encountered, even when only using a network of 50 nodes. Optimizations to the algorithm using distributed programming are noted by the authors as future work, which may improve performance significantly, but no details are given. This technique is an anchored, concurrent propagation, fine-grained, centralized, and deterministic localization technique.
• Wu et al. developed a novel localization technique which selects a small subset of nodes, and builds a co-ordinate system around these nodes, based upon their distances, oriented at an arbitrary node. Each other node in the network discovers the location of the landmark nodes (those used in the construction of the co-ordinate system), then selects its position to minimize the error in distances. A technique is presented to discover appropriate landmarks by selecting corner’ nodes in the network. In the main, error in this technique is caused by: distance calculation error, node density, and the number of landmarks. As the number of landmarks increases, the accuracy increases, but so does computational complexity. This algorithm is anchor free, uses concurrent propagation, has fine-granularity, is distributively computed, and deterministic.

• Huang et al. in a very recent paper discuss the use of non-deterministic Monte Carlo approximation [26]. The algorithm uses a mobile beacon, which passes through the network, broadcasting its position. The Monte Carlo approximation is done centrally, off-site, and the calculated locations are re-propagated to the appropriate nodes. The algorithm is anchored, uses concurrent propagation, with fine-granularity, and is centrally computed, non-deterministically. The advantage of using a non-deterministic algorithm is that it naturally handles the inaccurate readings produced by distance calculations. The algorithm increases its accuracy by considering both the beacon data, and the distances to one hop neighbours in its approximations. Results are among the best that can be expected in sparse networks, and improve as density increases. An interesting proposed future work is that of determining better paths for the mobile agent, which currently moves randomly.

• Two novel, biologically inspired localization techniques were recently presented by Kulkarni et al. in [31]. The first technique uses particle swarm theory. Particle swarm optimization involves a population of particles searching some n-dimensional space looking for a global optimum. For a WSN, the global optimum is the node’s actual location, and the fitness of a particle is its distance from this location. The second biologically inspired algorithm is a bacterial foraging algorithm. In this algorithm, a number of bacteria are created, and they can either swim, or tumble. Swimming means that they continue moving in their current direction. Tumbling involves randomly changing their direction. If the bacteria’s current location is better than
its previous location, it swims. If the current location is worse, it tumbles. After a number of rounds, the bacteria in good locations are duplicated, and the bacteria in a bad location are killed off. The fitness function for goodness in a WSN is the same as for particle swarm optimization. Kulkarni et al. propose a hybrid of these two algorithms as a localization algorithm for WSN. Nodes with three or more localized neighbours are localizable, and run the two algorithms to find the location with best fitness. This technique is anchored, has iterative propagation, is fine-grained, distributively processed, and is non-deterministic.

Localization is a challenging problem, to which no best solution exists. There are a number of confounding factors: time limits for the production of a localization, upper limits upon the energy available for localization, required accuracy of localization, ability to provide central processing, and several others. Like so many problems in WSNs, a suite of solutions is appropriate, each solution being best suited for particular requirements. Algorithms which operate locally, like [31], are a likely future for WSN because of their minimal message passing requirements. Novel approaches to beaconing provide several advantages [26]. Centralized algorithms produce results of the highest accuracy, can take good advantage of non-deterministic algorithms like Monte Carlo approximations, but require a high message passing overhead, and high computational cost.

2.2 Operation Phase

There are a number of challenges that become apparent during the operation phase of a WSN. It is vital that operation phase problems be solved well, because they will be run many times, and inefficiencies will vastly decrease the lifetime of the network. The discussion of operation phase problems are broken down into object detection, phenomena reporting, multiple object management, internal representations of state, and communication routing.

2.2.1 Object Location

Object Location is a problem which varies in difficulty and accuracy depending upon the type of sensor available. In a binary sensor network (BSN), it is nearly trivial. When an object first enters the network, entrance detection occurs whenever any sensor falling on the convex hull of the WSN
detects an object. Sometimes an object can become lost in a BSN, either because of node failure or faulty sensing. In this case, the object will eventually enter the sensing region of another sensor, and the location will again be known. For all cases, a closed area which contains the object is the granularity of the object location. This closed area is the intersection of the sensing region of all nodes sensing the object. Although it is common to consider a binary sensor model, other models are frequently used, particularly for the purpose of more accurate object position detection (Shrivastava et al. show in [48] that the ideal granularity of a proximity BSN is inversely proportional to the product of the density of the network, and the sensing radius of the nodes). In these models, object location is more difficult.

- Particle Filtering (sequential Monte Carlo methods) is used by Coates in [17] to derive a position approximation from sensor data. The particle filter is applied to manage the noise incurred by the sensors, and to make use of many sensors’ data to provide accurate location information. Coates proposes a novel distributed particle filtering algorithm to this end, and applies it to audio sensor data.

- Sheng et al. provide several improvements to this algorithm in [45], segmenting the algorithm and reconstructing the results, and reducing the message passing cost. The simulation results yielded a significant reduction in message cost, and produced results as good as or better than Coates’.

- In [7], Aslam et al. show that for BSN using a particle filter both a range of possible positions of where the object may be, and a direction that the object is going can be found. The algorithm is centralized, but the results from [17, 45] were not available for these authors, and decentralization might be possible.

- Liu et al. use a novel non-deterministic sequential Bayesian filtering approach, which approximates the position of the object in [37].

2.2.2 Reporting of Phenomena

Another challenge in object tracking is to decide when and under what stimuli tracking information should be reported. In [52], Tilak et al. proposed a classification of data delivery into four
types: continuous (every t seconds, the sensors provide a report), event-driven (upon noticing a phenomenon, the sensors provide a report), observer-initiated (an external authority requests a report, which the sensors then provide), and hybrid. Considering object tracking as an application, a set of example reporting schema which epitomize each of these respectively is: every t seconds, the WSN builds a report of the current state of the objects in the network, and passes it to an authority; when an object enters (or exits) the network, the WSN reports the location of entry (or exit), and passes this information to an authority; when an authority requests an update, the network generates a report of the current state of the objects in the network, and returns it; if an object enters (or exits), a report is generated, if an authority requests one, a report is generated, if neither of these have occurred in t seconds, a report to this effect is generated. Liu et al. discuss a comb type push-pull hybrid system in [39]. This framework involves active nodes (those which are detecting a phenomenon) pushing their knowledge to locally close nodes. External agents query (pull) a comb-shaped sample of nodes that it is likely to overlap those nodes that are aware of a detection.

Reporting information at the right time is important to maintenance of battery life in WSN. The typical quality of service versus network lifetime trade off will eventually define the correct balance.

2.2.3 Managing Multiple Objects

Managing multiple objects presents a host of challenges for WSN, many of which are exacerbated by the minimal information available in a BSN. The biggest such problem is that of identity fidelity. When objects interact within the network, particularly when they come within a small distance of each other, maintaining the identity of the objects becomes difficult if not impossible. Busnel et al. discuss a formalism of proximity BSNs in [13], which shows that in a general proximity BSN, it is impossible to maintain identity fidelity. However, in networks which do not contain certain structures (mainly loops of small length), and with interactivity limits (two objects cannot simultaneously enter into one node’s sensing region), identity fidelity can be maintained. In general these assumptions are unrealistic, and identity fidelity is not maintainable.

When not considering BSNs, classification can be attempted to maintain identity fidelity.

• In [34], a collaborative signal processing approach to object tracking is taken. Li et al. propose signal classification based on a priori target signal types. That is, if multiple objects operate in
close proximity within a network, their movement is analyzed, and their identity is determined by trying to fit their movement into known patterns.

- In [38], multiple object tracking is considered as a state-space reduction problem. Distant objects are tracked individually, as though single tracking were occurring at the local level. As objects move towards one another, and their influence begins to interfere with the tracking quality, they become jointly estimated. Identity maintenance and location estimation are handled separately, which reduces the accuracy, but increases the efficiency and communication expense. Location tracking is done using a distributed particle filter, similar to [17].

- Shin et al. propose a novel representational framework for doing state-space reduction which is called Identity-Mass Flow [47]. There are two main contributions to this technique. The first is a doubly-stochastic matrix, which allows the maintenance of the possible identities of targets using only local information. The second is an algorithm for updating the matrix in $O(N^2)$ time, which is a significant improvement over some of the past algorithms.

All of these examples are non-deterministic algorithms based upon signal classification techniques that are similar to Sequential Monte Carlo approximations. In all cases, the only object identity fidelity that can be done is between objects that produce different sensing profiles. That is to say that a network can maintain the fidelity of a car compared to a person, but not between two cars. This is a limitation that WSN are not expected to overcome. In the future Radio Frequency Identification (RFID) chips might become ubiquitous enough that identity could be distinguished through their use.

### 2.2.4 Internal Representation of Object Location

- Introduced by Dolev et al. is the concept of treating the network as a database [22]. The network database tree should be expected to handle update and search requests. Update requests occur whenever an object moves from one position to another. Search requests occur when prompted by an external authority. The challenge then becomes discovering an internal representation which efficiently handles both of those operations.

- The contribution provided by Kung et al. in [32], is the Scalable Tracking Using Networked
Sensors (STUN) architecture, which consists of an internal state maintenance tree, an algorithm for updating the data structure, and an algorithm for efficient routing of information upon receipt of a query. STUN also specifies the construction of the internal state tree. A limitation of STUN is that it requires previous knowledge of high traffic locations of the network.

- In [35], a new data structure called a deviance-avoidance tree is proposed, which improves upon the STUN data-tree. Lin et al. also provide algorithms for updating and querying the tree. The main contribution in Lin et al.’s paper is a better data structure construction algorithm, while the principle of having a static routing data structure which is constructed once and used for the lifetime of the network is the same. The construction of the deviance-avoidance tree also relies upon knowledge of expected traffic patterns. Chapter 4.3 gives a more thorough investigation into these papers.

- Kulathumani et al. recently proposed an alternative, but similar type of internal representation algorithm, which maintains an active track to the object in [30]. At each node in this track, a pointer towards the object is available. Trail provides update and find operations, much as [32] and [35] did. One main difference in technique, in this paper, is that the data structure itself is dynamic, and updates itself on the fly. When an update operation occurs, the data structure itself changes, rather than just the information stored at the nodes of the data structure. Update operations are distance sensitive, meaning that their cost depends only upon the distance the object has traveled since the last update.

The future of internal representations is very important, because it defines the efficiency with which useful data can leave the network, and get into the hands of an external authority. Kuluthami et al.’s work is very new, and novel, and it is likely that more algorithms using similar techniques to this, or providing refinements to it, will become available. These works were the primary inspiration for the dynamic data structures used in this thesis.

### 2.2.5 Communication Routing

Tracing an optimal route between two nodes is difficult using localized algorithms and information. Also, it might be desired that multiple sub-optimal routes be rotated between to avoid heavy strain.
on any single pathways if many communications occur. When considering the task of object tracking, we can reduce the complexity of this problem considerably. The concern for tracking problems is in most cases communication with a predetermined node (the sink). However, WSNs generally require more complex routing protocol. Macker et al. outlined a series of desirable qualities in a routing algorithm in [40]. The qualities outlined include: distributed nature, loop-free behavior, demand based operation, “sleeping” operability, and minimization of hop count and delivery rate. These requirements should be met by all algorithms proposing to solve the communication routing problem.

- The simplest routing models are: network flooding, gossiping, and direct transmission [1]. In network flooding, every node in the network passes the message to each of its neighbouring nodes in the network. Coverage of the network is guaranteed, but there is a high energy cost overhead as every node is involved. In gossiping, a node chooses a random neighbour to pass the message on to, and this continues until the destination node is reached. While gossiping usually outperforms flooding, it is often inefficient, does not guarantee loop-free behaviour, and does not guarantee that the destination node will be reached. Direct transmission is the simplest of the three, as the broadcasting node transmits the message directly to the destination node. Energy cost is typically exponential in the distance transmitted, so for most cases, multi-hop transmission is more efficient. However, in cases where the destination is close to the source, direct transmission can be the most efficient [24].

- Stojmenovic et al. considered the routing problem in [51]. Eight goals were identified: energy optimization, loop-freedom, task quantity maximization, low communication cost overhead, past-behavioural independence, localized-nature, single-path routing preference, and maximization of delivery likelihood. Stojmenovic et al. also consider the use of mobile agents, but were able to separate concerns between the problem of routing in a WSN, and between the complexity of mobile agents. Three new power saving routing algorithms are reported. First, a localized version of Dijkstra’s single source shortest weighted path algorithm is proposed. The second algorithm considers the remaining lifetime of the nodes as their weight in the graph, thus nodes with little remaining life are avoided in favour of nodes with longer remaining life. The final algorithm proposed by Sojmenovic et al. is a hybrid algorithm that combines the two
above weightings in one of two ways. The first way is taking the product of the two types of weightings, and the second method is a sum of the two types of weightings (with a coefficient added to each term for tweaking the influences).

- In [24], Heinzelman et al. critique protocols similar to Stojmenovic’s, since nodes close to a sink would drain quickly and be replaced by nodes a bit further from the sink, which would then also drain quickly, until large sections of the WSN would no longer be sensing. Heinzelman et al. propose instead a cluster-head lead protocol, where nodes dynamically volunteer themselves to be a leader, and all those nodes closest to the leader route their communication through this node. The cluster heads periodically change, when a new node volunteers. The drain of routing messages is rotated among different nodes at different times. This drain balancing routing is shown to vastly improve the overall lifetime of the network, and also the lifetime before a single node failure due to battery drain.

- Heinzelman et al.’s work is improved upon in [36] by adding an ordered ‘turn-taking’ for cluster heads. Instead of nodes nominating themselves, an order is established, and the nodes take turns being the cluster head. The order is established randomly, thus the leader in each round is in a random position in the cluster. Lindsey et al. also add a data fusion method to Heinzelman et al.’s work. At each transmission, the nodes near to the leader collaborate to aggregate their data, producing a digest of the data collected before transmission. This improved algorithm increases the lifetime still further.

### 2.3 Complete Systems

Theoretical considerations of the WSN tracking problem have yielded enough results that complete systems have been designed. Some of these complete systems were implemented on WSN hardware, and tested in the field, others were not.

- Arora et al. designed and implemented a dense WSN for surveillance in [5]. Their solution uses locally similar node collaboration, and comparison against previous data. The solution also provides multiple object identity fidelity through a novel classification technique using
influence fields. The idea of influence fields is similar to that in Li et al. in [34], but is technically very different. This implementation is based around a binary sensing model.

• De La Parra produced a complete system description of a WSN for tracking a single object in [19]. The system was hierarchical, providing two tiers of nodes, those which route data, and those which sense. There is no concrete implementation of this system.

• Kim et al. implemented their system of path-based target tracking using binary sensors in [29]. The model uses acoustic sensors to detect object proximity. The path-based tracking system was not implemented as a field network, however data was collected from a field of sensors, and then fed into a system which simulated the implementation. This study made several simulation assumptions, including an assumption that the targets in the WSN have a constant velocity.

• Sharp et al. produced a complex complete tracking system, which not only tracks an object, but also assists autonomous agents in the interception of tracked objects in the network [44]. The system is also multi-tiered, the WSN being the low level tier, and the robot agents providing the high level. This system provides a concrete implementation, and several problems with faulty hardware were encountered. To this end, it was required to ignore several nodes in every run. The system used magnometers for sensing, and problems were encountered involving the magnetic region surrounding the sensor becoming warped by the battery, the antenna, and the base of the node, which reduced the quality of sensing significantly.

• Arora et al. produced another implementation called ExScal in 2005 [6], which is a low density WSN, for border monitoring. The goal of this WSN is to detect border crossings, and was deployed in a 1km by 300m area. ExScal was a hierarchical WSN with low level nodes which sense and handle small distance messages, middle level nodes which handle longer distance communication, and a high level base station node. The nodes sense a range of 7m in the worst case, so given the network size, and only 1,000 nodes, coverage was impossible. Instead of coverage, Arora et al. focused upon network boundaries; if any target passes through a boundary, at least five nodes would sense it. To achieve this boundary coverage, accurate placement was necessitated, and 552 man hours were invested in the marking and placement
of nodes. A second paper [9] outlining the results of the experiment is optimistic. A number of lessons learned and possible improvements were given.

2.4 Conclusion

Object tracking in WSN is non-trivial, and the best algorithms depend upon the specifications of the sensors, the density of the network, the available battery life, and a number of other factors both internal and external to the network. The challenges facing object tracking in WSN are interrelated, requiring optimizations for battery life, minimization of communication frequency, message size, transmission distance, avoidance of nodes with low remaining battery life, tolerance to frequent node failures, and lack of known node location, among others. In this survey, a handful of localization algorithms were discussed, and some of their pros and cons were outlined. Object detection as a goal was related, and its differing requirements depending upon sensor assumptions were given. A set of data delivery schema was outlined, and an example of their application in the WSN tracking problem was given. Several of the difficulties regarding multiple object tracking were defined, and a couple of proposed solutions were shown. The idea of treating WSN object tracking operations as database operations were described, and two architectures implementing this idea were outlined. Finally a discussion of the problems relating to communication routing in light of energy awareness was explained, and four potential solutions were given. Object tracking using WSN is attractive because of its low cost, and its flexibility, but this chapter has focused in majority upon the challenges that are encountered. In light of these challenges, object tracking using WSN is being explored in research, and several recent papers have created solutions which successfully track objects in deployed networks.

This chapter has focused on object tracking, because the tracking problems motivated the work in this thesis. Track persistence has to deal with all of the same problems as object tracking, before it can handle its own unique problems.
Chapter 3

Problem Specification

3.1 Definitions

Definition 1. WSN Diameter
The diameter of a WSN $W$ is the diameter of the convex hull of the sensors in $W$.
The diameter of $W$ is written as $diam(W)$.

Definition 2. Covering Set (CS)
A CS is a set of sensing nodes $S$, such that every point within the convex hull of the nodes is sensed by one or more nodes.

Definition 3. Minimum Covering Set (MCS)
An MSC is a smallest subset of $S$ that is a covering set.

3.2 Assumptions

For the remainder of this thesis, we only consider WSN which satisfy these assumptions.
A Wireless Sensor Network $W$ is deployed.

- $W$ is a dense, connected network.
- $W$ is distributed, so that the nodes in $W$ make up a covering set.
• For every sensor $\gamma$:
  
  – $\gamma$ has a circular sensing range.
  
  – $\gamma$ is location aware.

• There will be at most one object in the network at any time.

• All nodes in the WSN have the same sensing radius, and sampling rate.

• There is one node, called the sink, which can directly communicate with an external authority, and which has effectively infinite battery life.

### 3.3 Track Persistence

The fundamental problem addressed by this thesis is maintaining historical movement information of objects moving through a network over time. The approach used here, successfully applied in [32, 35, 30], is to think of the WSN as a database. The four fundamental operations of a database are insertion, deletion, updating, and querying.

1. Insertion occurs whenever an object enters the network for the first time. Its presence is noticed by a node in the network, and tracking begins. Insertion is a special case of updating.

2. Deletion occurs whenever an object leaves the network. Its exit is noticed by a periphery node, and tracking is halted. Deletion is a special case of updating.

3. An update occurs whenever an object moves between two nodes in the network. The network makes some structural or content-based change.

4. Querying occurs whenever requested by an external party. The sink receives the request for query information, and the network produces the tracking information from in network, at which point the sink relays the information back to the external party. Tracking is maintained.

For simplification of the problem of analysis, insertion and deletion are treated identically to updating operation.

In formalizing this problem, we must develop distributed frameworks for performing the update and the query operations.
CHAPTER 3. PROBLEM SPECIFICATION

Problem 1. Update

Updating the network needs to be done whenever an object enters the network, moves from one nodes’ sensing region to another, or when an object leaves the network through a periphery node. The goal in updating the network is parallel to the problem of updating a database: changing the internal structure and content of the system to match the real data set, in this case the physical location of the object. This is however only half of the problem. Since message passing is the largest overhead for the lifetime of the network, updating the network should be done with as few message passes as possible. It may be necessary to sacrifice the quality of the model for lower battery cost, as in a real database uniqueness is sometimes violated for the sake of efficiency. The exact information being encoded and the data structures used to encode it can be varied to better suit the goal of minimizing communication cost; the only requirement is that an object’s path information is retrievable.

Problem 2. Query

Querying the network can be requested by the central authority at any time. The signal to query is received by $\sigma$, and some part (or all) of the network takes part delivering the complete history of the movement of the object $o$ currently in the network. At the time of a query, the object $o$ is not assumed to still be physically inside the network though it may be. The task of minimizing the cost of querying is related to the updating task; the data structures used to store the data, the locations in the network that the data is stored and their physical distance from the sink all effect the cost of querying.

Individually the Query and Update problems may not be difficult to solve. The problem proposed by this thesis however is different: we consider the two problems in tandem, building a framework such that, given a query frequency and an update frequency, we can maximize Equation (3.1).

$$\text{network lifetime} = \frac{1}{\text{update frequency} \cdot \text{cost(update)} + \text{query frequency} \cdot \text{cost(query)}}, \quad (3.1)$$
The equation is in four unknowns, two of which are dependent upon the algorithm \textit{cost(update)}, \textit{cost(query)}, and the other two are independent of the algorithm. The \textit{update\_frequency} is independent because it is related only to the objects which interact with the network. If the objects are very fast moving, then the \textit{update\_frequency} will be high. The \textit{query\_frequency} is independent because it is activated by an authority outside of the network.

In some cases, an external authority will be interested in the track persistence information of every object that interacts with the network. In these cases, ignoring compression techniques, the chosen algorithm will not affect the total quantity of information that must be transmitted - all the data must eventually make its way to the sink. The chosen algorithm does however still have an effect on the network’s lifetime - an algorithm which sends small messages frequently will have a larger overall cost than an algorithm which sends larger messages less frequently. The cost of transmitting a message is determined by the size of the message, however there is a startup cost associated with a transmission. Because of this startup cost, it is less expensive to send one message of \( n \) bits, than it is to send \( n \) messages of one bit.

The remainder of this thesis will address the task of finding appropriate algorithms to solve the problem described in this chapter.
Chapter 4

Algorithms

4.1 Specification

4.1.1 Motivation

The intention of this thesis is to provide a suite of algorithms for handling path persistence in WSN. Since this is among the first works considering this problem, there are no algorithms with which to compare the proposed algorithms. To this end, a naive algorithm is provided, and used as baseline comparisons against the other algorithm. This subsection will detail the algorithms, and make some comments about possible indicators for when to use which techniques. It is important that all algorithms be treated as equals within this thesis. Similar effort will be spent on optimization and documentation for all algorithms.

4.1.2 Notes

Communication Routing

The problem of routing an update from a node to the sink is obviously non-trivial, and a number of solutions have been presented. The three naive routing techniques: gossiping, flooding and direct transmission, are inappropriate for use because they don’t scale, don’t necessarily guarantee coverage, and don’t balance the cost. The primary requirements for communication routing in
the algorithms proposed below are: distributed nature, load balancing, demand based operation, “sleeping” operability, minimization of cost, avoidance of low energy nodes, and maximization of reliability. Sophisticated algorithms are available in the literature, which solve each of these problems in novel ways. Generally, the selection of an algorithm for routing is a problem which should be handled on a per-implementation basis; in Chapters 5 and 6 particular routing assumptions are made for the sake of analysis.

Messages

It is assumed that each node has a message box, and whenever a message is received, it is stored in this message box until the node has a chance to deal with it. Only messages which are intended to terminate at this node are placed in the message box, all other nodes are routed as required. Messages are stored in the box ordered from least recent to most recent, in the order of a FIFO queue.

Overcoming Clock Synchronization

There have been many recent improvements in clock synchronizations for WSN; [50, 33] are two recent studies, which achieve errors as low as hundreds of µs. Clock Synchronization is an area of constant study, and of advanced algorithms, which will continue to improve as the hardware improves. However, maintaining high quality synchronization requires an energy cost, and it is beneficial if we don’t use clock synchronization unless necessary. The algorithms used below avoid the use of this fine-grained clock synchronization, by working under some coarse-grained synchronization assumptions:

1. Every actively sensing node in the network senses the network at least once in a \( \tau \) second period.

2. Given any two actively sensing nodes, the drift between their two clocks is less than \( \tau \) seconds.

Essentially, the sampling rate (granularity of sensing, \( \tau \)) is a maximum on the clock granularity. In Chapter 6 we will use \( \tau = 0.1s \), which is well within the possible granularity of synchronization.
CHAPTER 4. ALGORITHMS

4.1.3 Memory

For the sake of these algorithms, we consider memory to be finite but unbounded. Typically, most nodes will have very little memory requirements, but as network lifetime passes and many events are observed, it is possible that any presupposed finite memory could be overwhelmed. In these cases, it is probably safe to cull the oldest data, and only store the more recent events.

4.1.4 Sample Run

For explanatory purposes, a small sample run is defined here. As each potential solution is introduced, a tracing of each nodes’ actions during each time step will be given. Hopefully, this will clarify any unclear or difficult cases. Figure 4.1 provides the legend for the sample run, and figure 4.2 shows the movement of the object in the sample run.

![Legend of nodes in sample run](a) Legend of nodes in sample run

![Graph representation of WSN](b) Graph representation of WSN

Figure 4.1: Small sample sensor network

4.2 Update to Sink

4.2.1 Specification

The simplest solution to handling track persistence is to turn it into a centralized problem by passing a report message to the sink at each update. We call this algorithm *update to sink*. The benefits of updating the sink immediately are:
• Querying operations are free.

• Maximum message size is small.

• Simplicity: implementation is nearly trivial.

The negative side effects of immediate updating are:

• High update cost especially for nodes distant from the sink.

• Fails to take advantage of the distributed computation and memory of a WSN.

The update to sink solution is particularly useful when updates are infrequent and queries are very frequent. More analysis of the update to sink algorithm can be seen in Chapter 5.

### 4.2.2 Algorithm Details

**Update**

Performing an update operation involves notifying the sink of whatever update is perceived. Typically this involves sending a message to the sink, however there is a special case. If two nodes \( n_1, n_2 \) notice a target in their sensing region at the same time \( t \), then these two messages should be fused into one message before being sent to the sink. This case generalizes to any number of nodes. There are two types of updates: *entrance* updates, and *exit* updates. Both updates need to be passed to the sink. If high update latency is acceptable to the sink, it will make sense not to report an *entrance* update until the *exit* is also perceived.

**Query**

Performing a query operation is simple. The sink receives the query, looks into its memory, and can immediately return with the track data already available to it.

### 4.2.3 Sample Run

The following is a time-step by time-step description of the sample run seen in figure 4.2. Note that the update message format is \(<\text{node identifier}>, <\text{message type}>, <\text{round}>\).
1. $n_6$ sends a message to the sink: (6, “Object Seen”, 1).

2. All nodes do nothing.


5. $n_3$ sends a message to the sink (3, “Object Seen”, 5).

After some amount of time, $n_5, n_4, n_3$ all note that they are not sensing $O$ any longer, and send similar messages (identifier, “Object Lost”, $t$) to the sink. The sink can then reconstruct the path from this list of messages.

- **From:** $n_6$, “Object Seen”, 1
- **From:** $n_5$, “Object Seen”, 3
- **From:** $n_6$, “Object Lost”, 3
- **From:** $n_4$, “Object Seen”, 4
- **From:** $n_3$, “Object Seen”, 5
- **From:** $n_5$, “Object Lost”, 6+
- **From:** $n_4$, “Object Lost”, 6+
- **From:** $n_3$, “Object Lost”, 6+

4.2.4 Conclusions

The update to sink algorithm is very straightforward, and simplifies track persistence into a centralized algorithm. It is a naive algorithm, which is intended for use in edge cases, particularly when queries are frequent.
4.3 Linked List

4.3.1 Specification

The idea behind the second algorithm, which we call linked list, is to maintain a dynamic linked list-like data structure throughout the WSN. Each node that the object passes through becomes a node in the linked list. The information that a node stores is the time of entrance, and time of exit of the object from this node, and the node identifiers for the previous and next node. An update operation is local, and involves a small number of nodes updating the node identifiers of their links. It is possible in this scheme that a node may occur multiple times within the linked list. It may even be possible that an object may enter into a loop, traveling over the same nodes again and again.

It is important then, that we be able to differentiate the $n^{th}$ pass through this node by time data alone.

Some advantages of using the linked list path persistence data structure are:

- It is a fully distributed data structure, which maintains the path by taking advantage of distributed storage.

- Each update operation is a local operation only, with small constant size messages.

- Provides a mechanism for retrieving partial paths at a lower cost than their complete counterparts.

Some disadvantages of this data structure are:

- It is brittle. If one node in the path is damaged, or becomes irreparable, the whole path is disrupted (although it is possible to rebuild the track despite this. Karp and Kung proposed the use of a right-hand rule, to loop around the damaged area, until a node that is aware of the track is found, in \cite{28}.)

- Query operations are expensive. Querying becomes a three part problem. First the linked list must be discovered, then the list must be followed, iteratively building the track at each node. After a full traversal of the list, the full track is transmitted back to the sink.
4.3.2 Algorithm Details

Update

Performing an update operation involves modifying the internal representation of the linked list. Typically this involves modifying next and previous pointers. For a node \( n \) in the track, two sets of pointers are maintained - those which point to the previous nodes, and those which point to the following nodes. There are two types of updates: \textit{entrance} updates, and \textit{exit} updates. When a node \( n \) senses an object \( p \) in its radius, \( n \) sends the message “ENTRANCE” to all nodes in \( N(n) \), and similarly when \( n \) ceases to sense \( p \), \( n \) sends the message “EXIT”. Each of these messages are sent to all neighbours. When a node \( n \) receives an “ENTRANCE” message from node \( n_e \), \( n \) adds \( n_e \) to a list of potential previous nodes. Similarly, if \( n \) receives an “EXIT” message from node \( n_e \), \( n \) removes \( n_e \) from it’s list of potential previous nodes; node \( n_e \) must already exist on this list, since we assume lossless communication. At any time if \( n \) senses \( p \), its list of potential previous nodes becomes its list of actual previous nodes, those nodes that were sensing the object \( p \) before \( n \) was.

Determining the previous pointers is straightforward, and would be sufficient if the list were only to be accessed from one end, but querying may require that it be traversable in either direction. Therefore, we must also determine the list of pointers to nodes which the object next enters. For determining the list of next nodes, the \textit{entrance} updates must also broadcast the round in which the entrance detection occurred. When a node \( n \) senses an object \( p \), \( n \) notes the round \( r_{seen} \) and broadcasts “ENTRANCE \( r_{seen} \)” to all nodes in \( N(n) \). When a node \( n \) receives an “ENTRANCE \( r_{received} \)” message from node \( n_e \), if \( r_{received} > r_{seen} \), then \( n_e \) is added to the list of next nodes, and \( r_{received} \) is stored - note that \( r_{seen} \) for this node is unset until \( p \) is seen for the first time. If any future “ENTRANCE” messages are received, unless their round was also \( r_{received} \), they are not added to the list. Constructing the list of nexts in this way determines the node which \( p \) enters at the closest moment in the future, but no more; in the case that there is a tie, all tying nodes are stored.

Update operations for the linked list algorithm are fully localized, with messages remaining only within the 1-hop neighbourhood of any node. A set of previous nodes, and a set of next nodes is
maintained, so that the list can be traversed in either direction.

Query

Performing a query operation consists of three steps. First, either the head or the tail of the list must be found. Second the list must be traversed iteratively building the track. Finally, the track must be transmitted back to the sink.

Discovering an end of the list can be achieved in a number of ways:

1. The simplest option is to have the sink flood the network with a search request. In this case, each node would report only if it is the head of the list.

2. The sink can send a search request, so that the network will search in increasingly large concentric circles, until eventually a node on the list is found.

3. A different approach is to maintain some form of data structure to facilitate finding an end. A selection of these approaches follows:

   - In [32], Kung et al. proposed an object tracking architecture that uses a tree-like data structure to provide efficient querying of an object’s location in a network. The architecture proposed, called Scalable Tracking Using Networked Sensors (STUN), tries to maintain the position of the object by updating as little of the network as possible, that is, limiting updates to those nodes which are near the leaves of the internal tree structure. The source of a query is the root of the tree. The tree contains two types of nodes: communication nodes, which make up the non-leaf nodes, and sensing nodes, which make up the leaf nodes. The presence of any objects is stored in the communication nodes; in particular, each communication node knows all of the objects being sensed by all of its descended sensors. These lists of sensed objects are kept up to date by passing detection information up the tree towards the root - if a node knows that it won’t modify the tree any further up, then the communication halts. When a query arrives to the root of this tree then, it can be deterministically traversed to determine the location of the object very quickly. The idea of such a tree is not the contribution of this paper, and is used in
cellular telephone systems [42]. Of primary importance to the usefulness of such trees is
their construction: if an object travels between two nodes $a$ and $b$ with high frequency,
then nodes $a$ and $b$ should have a low common ancestor (causing as little of the tree to
be updated as possible), whereas if an object must move a very large distance (or move
in a way that it is unlikely to) to reach node $b$ from node $a$, there is no need for them
to have a low common ancestor. Thus, a priori knowledge of the expected movement of
objects is required for construction of this data structure. The primary contribution from
Kung et al., in this paper is the drain-and-balance tree construction algorithm, which
involves first bunching a subset of the sensors into balanced subtrees, and then iteratively
merging these subtrees. The technique takes advantage of edge weights in the graph,
which represent the expected frequency of movement between nodes $a$ and $b$. Kung et
al.’s attempt appears to be the first use of such internal structures for improving the effi-
ciency of querying in WSNs, but the drain-and-balance construction suffers from certain
drawbacks: edges did not necessarily reflect direct communication connectedness, so while
the edge weight might be high, the euclidean distance might also be high; also, the tree
constructed by drain-and-balance could have a large depth, resulting in high query costs.
Lin et al. address these concerns in [35], by proposing a different construction method,
which ensures that no path in the original graph between nodes $a$ and $b$ is shorter than the
path between $a$ and $b$ in the constructed tree. They also propose a post-construction re-
finement algorithm which also reduces the query cost. Lin et al. also operate under Kung
et al.’s assumption that some knowledge of the expected high traffic areas is known, and
they combine this with the physical distance (number of communication hops) to produce
the edge weights.

- In [30], a totally novel, hierarchy free, distance-sensitive, energy efficient tracking solution
called Trail is proposed. Trail allows much more flexibility than the previously described
algorithms: no a priori knowledge of the traffic in the network is required, no static data
structures are constructed, and update operations are always distance-sensitive (their cost
is related to the distance the object moves). Trail works by maintaining a track from an
object to the center of the network. These tracks are maintained in such a way that they
are at longest 1.2 times the distance from the object to the center of the network. A track consists of a set of segments, with the longest segment touching the center of the network, and exponentially smaller segments towards the object. When the track is first built, or when it is reset (which occurs only when the object has moved a long enough distance), the track consists of only one segment, and the number of segments increases as the object moves. Update operations modify the small segments close to the object, and the larger segments are only modified after a large amount of movement. This is a nice property, because longer segments are more prone to failure, and cost more to modify.

4. It is always possible to detect a new object entering the network. At this time, an end of the list is known. So, another option is to simply record this pointer to the end of the list, at the time of object detection.

We will proceed assuming option 4, but any of the above can be successfully used for querying. One potential benefit of options 1 through 3 is that they can report the front of the list, while option 4 can only report the end. A further analysis of the benefits and costs of these options may be necessary for anyone implementing this algorithm.

Once the end of the list is found, the track information will need to be reconstructed. This is a straightforward task. Three lists must be maintained: a list of futures (those nodes which must be visited), a list of pasts (those nodes which have already been visited), and the list of track data collected so far. The list of pasts contains duplicate data to track data collected, and is only thought of separately as a convenience. If a node receives a reconstruction message, they respond in the following way:

1. Extract the three lists from the received message.

2. Add this node’s track data to the data list, and add its identifier to the list of pasts.

3. Look at this node’s internal list of nexts, which were built as the object traversed the network, and add all of these nexts to the end of the list of futures.

4. Remove any items from the list of futures if they exist in the list of pasts.

5. If the list of futures is empty,
(a) this is the last node in the track; send a message, containing just the collected data, to the sink

(b) else, select the first item on the list of futures as the destination, and remove it from the list, then reconstruct the message, and pass it on to the destination.

It should be noted, that this is the algorithm for reconstructing if one begins at the entrance point of the object. Working in the other direction is a trivial change.

4.3.3 Sample Run

The following is a time-step by time-step description of the updates performed during the sample run seen in figure 4.2.

Note that the update message format is ($\langle$node identifier$\rangle$, $\langle$message type$\rangle$, $\langle$round$\rangle$).

1. $n_6$ notes the time, and broadcasts a message: (6, “Object Seen”, 1).

2. No other nodes noticed the node at $t_1$, so $n_6$ sends a message to the sink of type “Track Starts Here”. At each node in the path from $n_6$ to the sink, the node identifier is added to the message, and this message forms a pointer for the query request to follow. Also, the neighbours of $n_6$ ($N(n_6) = \{n_3, n_4, n_5, n_7\}$) notice the first message, and add $n_6$ to their list of potential previous nodes.

3. $n_5$ notes the time, and broadcasts a message: (5, “Object Seen”, 3); the set of potential previous nodes for $n_5$, i.e. $\{n_6\}$ becomes the final set of actual previous nodes. $n_6$ broadcasts a message: (6, “Object Lost”).

4. All the nodes in $N(n_6)$ remove $n_6$ from their list of potential previous nodes. Since $n_5$ has already locked $n_6$ in as an actual previous node, it is unaffected. Nodes in $N(n_5) = \{n_3, n_4, n_6\}$ add $n_5$ to their list of potential previous nodes if they have not yet seen the object, as is the case for $n_3$ and $n_4$. $n_6$ has seen the object already, so $n_5$ is added to its list of nexts, and since no other “Object Seen” messages are received from round 3, $n_6$’s final next set is $\{n_5\}$. Finally, $n_4$ notes the time, and broadcasts the message: (4, “Object Seen”, 4); the set of potential previous nodes for $n_4$, i.e. $\{n_5\}$ becomes the final set of actual previous nodes.
5. Nodes in $N(n_4) = \{n_1, n_2, n_3, n_5, n_6, n_7\}$ add $n_4$ to their list of potential previous nodes if they have not yet seen the object, as is the case for $n_1, n_2, n_3$, and $n_7$. $n_6$ has seen an object, but its list of nexts’ is set, so it is unchanged. $n_5$ adds $n_4$ to its list of nexts, and since no other “Object Seen” messages are received from round 4, $n_5$’s final next set is $\{n_4\}$ . Finally, $n_3$ notes the time, and broadcasts the message: (3, “Object Seen”, 5); the set of potential previous nodes for $n_3$, i.e. $\{n_4, n_5\}$ becomes the final set of actual previous nodes.

6. Nodes in $N(n_3) = \{n_1, n_2, n_4, n_5, n_6, n_7\}$ add $n_4$ to their list of potential previous nodes if they have not yet seen the object, as is the case for $n_1, n_2$, and $n_7$. $n_5$ and $n_6$ have seen the object, but their list of nexts’ is set, so they are unchanged. $n_4$ adds $n_3$ to its list of nexts, and since no other “Object Seen” messages are received from round 5, $n_4$’s final next set is $\{n_3\}$.

At this point, the state of each node in the network is summarized in Figure 4.3.

<table>
<thead>
<tr>
<th>Node</th>
<th>Potential Prevs</th>
<th>Prevs</th>
<th>Nexts</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1$</td>
<td>${n_3, n_4}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$n_2$</td>
<td>${n_3, n_4}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$n_3$</td>
<td>$\emptyset$</td>
<td>${n_4, n_5}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$n_4$</td>
<td>$\emptyset$</td>
<td>${n_5}$</td>
<td>${n_3}$</td>
</tr>
<tr>
<td>$n_5$</td>
<td>$\emptyset$</td>
<td>${n_6}$</td>
<td>${n_4}$</td>
</tr>
<tr>
<td>$n_6$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>${n_5}$</td>
</tr>
<tr>
<td>$n_7$</td>
<td>${n_3, n_4}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

Table 4.1: The state of the linked list at time $t_6$

At some time after $t_6$, the object may leave the network, and if this happens, the potential futures described in Figure 4.3 will become the empty set, but the remainder of the table will be accurate. Say that a query request arrives at the sink after this point. A brief outline of the query is:

- The pointer, which was built up by the messages sent at time $t_2$, is followed in reverse, bringing the query request to node $n_6$.

- $n_6$ builds the three lists described above:
  - futures: $\{n_5\}$
  - posts: $\{n_6\}$
- track data: \{[n_6 > (\text{Enter} @ t_1, \text{Exit} @ t_3)]\}

\(n_6\) sends these three lists to the first node in the list of futures (i.e. \(n_5\), and removes this node from the list.

- \(n_5\) receives a message containing:
  - futures: \(\emptyset\)
  - pasts: \{\(n_6\)\}
  - track data: \{[n_6 > (\text{Enter} @ t_1, \text{Exit} @ t_3)]\}

\(n_5\) adds its own data to get the lists, it is removed from futures, and it is sent a message containing the three lists:

  - futures: \{\(n_4\)\}
  - pasts: \{\(n_6, n_5\)\}
  - track data: \{[n_6 > (\text{Enter} @ t_1, \text{Exit} @ t_3)], [n_5 > (\text{Enter} @ t_3, \text{Exit} @ t_{6+})]\}\)

\(n_4\) is the next node on the list.

- \(n_4\) receives a message containing:
  - futures: \(\emptyset\)
  - pasts: \{\(n_6, n_5\)\}
  - track data: \{[n_6 > (\text{Enter} @ t_1, \text{Exit} @ t_3)], [n_5 > (\text{Enter} @ t_3, \text{Exit} @ t_{6+})]\}\)

\(n_4\) adds its own data to get the lists:

  - futures: \{\(n_3\)\}
  - pasts: \{\(n_6, n_5, n_4\)\}
  - track data: \{[n_6 > (\text{Enter} @ t_1, \text{Exit} @ t_3)], [n_5 > (\text{Enter} @ t_3, \text{Exit} @ t_{6+})],
          [n_4 > (\text{Enter} @ t_4, \text{Exit} @ t_{6+})]\}\}

\(n_3\) is the next node on the list.
• \( n_3 \) receives a message containing:

- futures: \( \emptyset \)
- pasts: \( \{n_6, n_5, n_4\} \)
- track data: \[ [n_6 > (Enter@t_1, Exit@t_3)], [n_5 > (Enter@t_3, Exit@t_{6+})], [n_4 > (Enter@t_4, Exit@t_{6+})]] \]

\( n_3 \) adds its own data to get the lists:

- futures: \( \emptyset \)
- pasts: \( \{n_6, n_5, n_4\} \)
- track data: \[ [n_6 > (Enter@t_1, Exit@t_3)], [n_5 > (Enter@t_3, Exit@t_{6+})], [n_4 > (Enter@t_4, Exit@t_{6+})], [n_3 > (Enter@t_5, Exit@t_{6+})]] \]

And the list of futures is empty, so we have reached the end of the list.

• \( n_3 \) sends a message to the sink containing the most recent track data list generated.

### 4.3.4 Conclusion

The Linked List algorithm takes advantage of distributed storage by building a dynamic distributed data structure as the object traverses the network. When a query arrives, a reconstruction pass rebuilds the track from the data structure, and transmits it back to the sink. This chapter has defined the way this algorithm works. The remainder of this thesis serves to compare the costs of the algorithms described in the previous chapters, and to assess how they will solve the problem defined in Chapter 3.
Figure 4.2: Short example run
Figure 4.3: The state of the linked list at time $t_6$
Chapter 5

Algorithmic Analysis

As discussed in Chapter 3, algorithmic analysis of algorithms on WSNs can be very difficult for a number of reasons. For example, a large number of metrics can be considered, including: The number of nodes involved in the algorithm, the total battery cost across the network, the average node battery cost, the time until first node failure, and the locality of the algorithm. In this chapter, we will present some traditional analysis of the proposed algorithms.

5.1 Definitions & Preliminaries

We will proceed under the additional assumption that the communication radius is at least double the sensing radius for any WSN.

Note 1. For convenience we will use the following notation.

- $sr$ will refer to the sensing radius of a specific network, and unless multiple networks are in consideration, it will be used unqualified.

- $cr$ will refer to the communication radius of a network, with the same caveat as above.

- When discussing a graph of a network, we will notationally treat the vertices as their graph-theoretic meaning, as the names of sensing nodes, and as the Cartesian co-ordinate of a sensing node interchangeably. The meaning intended will be obvious through context.
• \( N \) will refer to the number of nodes in the network.

**Definition 4.** For two arbitrary vertices \( v, w \) in a graph \( G = (V, E) \), let \( P(v, w) \) denote a shortest path from \( v \) to \( w \) in \( G \).

Let \(|P(v, w)|\) be the number of hops in a path, and let a path of \( k \) hops be written as a sequence \( P(v, w) = (v = p_0, p_1, p_2, p_3, \ldots, p_{k-1}, p_k = w) \).

**Definition 5.** For two arbitrary vertices \( v, w \) in a graph \( G = (V, E) \), let \( \text{dist}_e(v, w) \) be the Euclidean distance between the nodes, which \( v \) and \( w \) represent.

![Diagram](image_url)

(a) two nodes close enough to communicate directly (b) two nodes which require an intermediary to communicate

**Figure 5.1:** If \( v, w \)'s sensing regions are overlapping, they can communicate.

**Lemma 1.** For two arbitrary vertices \( v, w \) in a graph \( G = (V, E) \), if \(|P(v, w)| = k \), then

\[
\text{dist}_e(v, w) \geq \begin{cases} 
\frac{\pi}{8} \cdot k \cdot sr & \text{if } k \text{ even,} \\
\frac{\pi}{8} \cdot (k - 1) \cdot sr & \text{if } k \text{ odd}
\end{cases} 
\tag{5.1}
\]

**Proof.** Consider a path \( \overline{P} \) similar to \( P \), except that it has the following properties.

1. For every point \( \overline{p} \in \overline{P} \), \( \overline{p} \)'s sensing region overlaps with the line segment \( l(v, w) \), with endpoints \( v \) and \( w \).

2. Of all the paths satisfying property 1, \( \overline{P} \) is the shortest such path.
We are guaranteed that there will be a path $\mathcal{P}$ by the assumption that $cr > 2 \cdot sr$, and because $W$ is covered.

Let $|\mathcal{P}| = l$, and note that $l \geq k$.

As above, $\overline{P^*}$ is the sub-path, obtained by taking every second item in $\overline{P}$. $|\overline{P^*}| = \lfloor l/2 \rfloor$.

Also, all sensing regions of the nodes $p^* \in \overline{P^*}$ are non-overlapping.

Since we know the area which all vertices in $\overline{P^*}$ fall into, and we know that the sensing regions of all nodes in $\overline{P^*}$ are non-overlapping, we can determine the minimum Euclidean distance between $v$ and $w$, by parameterizing it as a packing problem. That is, the minimum Euclidean distance between $v$ and $w$ is the length of rectangle required to pack $\lfloor l/2 \rfloor$ half circles into a corridor of width $2 \cdot sr$. An example packing of a corridor can be seen in Figure 5.2.

In the worst case, the number of half circles that you can pack into the corridor is given by equating the two areas.

$$2 \cdot sr \cdot dist_e(v, w) \geq \frac{\lfloor l/2 \rfloor \cdot \pi \cdot sr^2}{2} \tag{5.2}$$

Solving for Euclidean distance:

$$dist_e(v, w) \geq \frac{\lfloor l/2 \rfloor \cdot \pi \cdot sr}{4} \tag{5.3}$$

Expanding the floor:

$$dist_e(v, w) \geq \begin{cases} 
(\pi/8) \cdot l \cdot sr & \text{if } l \text{ even}, \\
(\pi/8) \cdot l - 1 \cdot sr & \text{if } l \text{ odd} 
\end{cases} \tag{5.4}$$
Since \( l \geq k \), it is also true that:

\[
dist_e(v, w) \geq \begin{cases} 
(\pi/8) \cdot k \cdot sr & \text{if } k \text{ even,} \\
(\pi/8) \cdot k - 1 \cdot sr & \text{if } k \text{ odd}
\end{cases} \quad (5.5)
\]

**Lemma 2.** For a network of diameter \( D \), the number of communication hops for any two nodes \( v, w \) in the network is \(|P(v, w)| \leq \lceil \frac{8D}{\pi sr} \rceil\). For convenience, let \( \lceil \frac{8D}{\pi sr} \rceil = \Delta \).

**Proof.** Proceed by contradiction:
Assume \( D = (\pi/8)k \cdot sr \), and that \( \exists v, w \) such that \( P(v, w) = k' \), and finally assume \( k' > k \).
From Lemma 1, \( dist_e(v, w) \geq (\pi/8) \cdot k' \cdot sr \).
But then \( \exists v, w \) such that \( dist_e(v, w) > D \), which is a contradiction.
Thus it must be true that \( k' \leq k = \lceil \frac{8D}{\pi sr} \rceil \).

**Note 2.** The number of neighbours of an arbitrary node \( n \) in a WSN \( W \) is in \( \Theta(N) \). This is obvious, since all nodes can fall within the communication radius of \( n \).

**Note 3.** It is obvious that the diameter of a network is bounded by: \( D \in \Theta(N) \). In the worst case, all nodes in the network can fall on a straight line that would be of length \( N \cdot sr \).

**Lemma 3.** For any arbitrary nodes \( v, w \), and any shortest path \( P = \{v = p_1, p_2, \ldots, p_n = w\} \) between them, it is true that:

\[
\sum_{p \in P}(|N(p)|) \leq 3 \cdot N \quad (5.6)
\]

**Proof.** Let \( v, w \) be nodes in a WSN graph \( G = (V, E) \), and let \( P = \{v = p_1, p_2, \ldots, p_n = w\} \) be a shortest path between \( v, w \). Consider the set of \( N - |P| \) nodes in \( G \) not in \( P \), call this set \( P_{out} \).
Consider any three consecutive nodes in \( P, p_{l-1}, p_l, p_{l+1} \). \( \neg \exists p \in P_{out} \) such that \( dist_e(p_{l-1}, p) \leq cr \) and \( dist_e(p_{l+1}, p) \leq cr \), and \( \exists p_x \in P\setminus\{p_{l-1}, p, p_{l+1}\} \), such that \( dist_e(p_x, p) \leq cr \). If such a node \( p_x \) existed, then \( P \) would not be a shortest path.
In plain English, any node not in the path may be the neighbour of at most three nodes in the path. Now, consider any two nodes \( p_l, p_k \in P \); if \( \text{dist}_{p_l, p_k} \leq cr \), then \( p_l \) and \( p_k \) must be consecutive nodes on the path, as well as neighbours. Thus, a path node may be neighbours with up to two nodes, its next node, and its previous node in the path, but no more.

So:

\[
\sum_{p \in P} (|N(p)|) \leq 3 \cdot (N - |P|) + 2 \cdot |P| \leq 3 \cdot N
\] (5.7)

**Note 4.** We can bound the cost of sending a message between two arbitrary nodes \( v \) and \( w \) by:

\[
\text{cost to send} \leq \Delta \cdot \text{cost to transmit} + 3 \cdot N \cdot \text{cost to receive}
\] (5.8)

Recall that \( \Delta \) is the longest possible shortest path, and \( 3 \cdot N \) is the maximum number of receiving nodes along this path.

### 5.2 The Algorithmic Cost

In this section, the algorithmic cost of running different operations will be discussed. As always, we consider only the battery cost of communication operations.

**Note 5.** Throughout this section, it will be useful to refer to an object entering or exiting the sensing region of a node, and we will call either of these incidents an event.

#### 5.2.1 The Total Network Cost

The total network cost is a metric which evaluates the cost of an algorithm by summing its battery cost to each individual node, to determine the overall cost. This is a useful metric, but it is unable to identify whether algorithms balance their cost across the network.

In any case, we can consider the total network cost to the algorithms proposed in Chapters 4 - 4.3.

**Definition 6.** The cost of performing a query or an entire update depends upon the number of events in that update, since the number of events is directly related to track length. Let \( \Upsilon \) be the
number of events for a given operation.

We are not able to put an upper limit on \( \Upsilon \) because it is possible that a target may loop indefinitely inside a network before leaving.

### 5.2.2 Update To Sink

**Update**

First let us consider the cost of doing an update operation on a single event. The cost of performing an update on a single event is:

\[
\text{cost of event} \leq \Delta \cdot \text{transmission cost} + 3 \cdot N \cdot \text{reception cost} \\
\leq N \cdot \text{transmission cost} + 3 \cdot N \cdot \text{reception cost} \leq O(N + N) \in O(N)
\]  

\[\text{(5.9)}\]

To perform an event, a message must be passed between the node at which the event occurs and the sink. The distance between these two nodes is bounded by the diameter of the network. Note 4 provides a bound for sending a message between arbitrary nodes.

\[
\text{cost of update} = \Upsilon \cdot \text{cost of event} \\
\in O(\Upsilon \cdot N))
\]

\[\text{(5.10)}\]

**Query**

Performing a query in the update to sink model is trivial, and is characterized by:

\[
\text{cost of query} \in O(1)
\]

\[\text{(5.11)}\]

This is obvious. Since all track information is available at the sink, querying a sink node does not require any communication - it is a free operation.
5.2.3 Linked List

Update

First let us consider the cost of doing an update operation on a single event:

\[\text{cost of event} \leq 1 \cdot \text{transmission cost} + N \cdot \text{reception cost}\]
\[\in O(N)\]  

(5.12)

It is reasonably obvious that an event causes a node to warn each of its neighbours of the event, which requires one transmission, and a reception cost from all neighbours. In the worst case, there are \(N\) such neighbours. It is important to note that this is the point at which worst case analysis becomes misleading: given a uniform distribution with a constant density, which is the case for well deployed networks, the number of neighbours for a node is a constant. Thus, in this case, the cost of updating a linked list is constant:

\[\text{cost of event} \leq 1 \cdot \text{transmission cost} + C \cdot \text{reception cost}\]
\[\in O(1)\]  

(5.13)

However, there is a minor complication in calculating this cost due to the first entrance event having some additional cost. One of the nodes participating in the event must send a pointer message to the sink. More details regarding the reasons for this message are given in Chapter 4.3. There is an election process which requires no additional messages, so the cost of this first event is:

\[\text{cost of first event} \leq \Delta \cdot \text{transmission cost} + 3 \cdot N \cdot \text{reception cost} + \text{transmission cost} + N \cdot \text{reception cost}\]
\[\in O(N + N) \in O(N)\]  

(5.14)

So, the cost of an update is:

\[\text{cost of update} \leq \text{cost of first event} + (\Upsilon - 1) \cdot \text{cost of event}\]
\[\in O(N + \Upsilon \cdot N)\]  

(5.15)

\[\in O(\Upsilon \cdot N)\]
However, if we consider the constant density case of this cost, it is much better:

\[
\text{cost of update} \leq \text{cost of first event} + (\Upsilon - 1) \cdot \text{cost of event} \\
\leq N + \Upsilon \cdot C \\
\in O(N + \Upsilon)
\]  (5.16)

**Query**

Querying a linked list-structured WSN is a three step process; first the endpoint of the track is found. This is done via a series of messages of decreasing size.

\[
\text{cost of find} \in O(N)
\]  (5.17)

This step is the reverse operation to the first step of the update operation. The derivation of its cost is identical.

Once the end of the track has been found, the query must be built by traversing the entire linked list.

\[
\text{cost to rebuild} \leq \sum_{i=1}^{\Upsilon} [\text{transmission cost} + N \cdot \text{reception cost}] \\
\leq \Upsilon + C \cdot \Upsilon \in O(\Upsilon \cdot N)
\]  (5.18)

The track reconstruction requires traversing the track until the last event is reached. Note that because the track may not be a shortest path, we cannot apply Lemma 3 to reduce the number of neighbours in the worst case. As with the \text{cost of event} above, assuming constant density, the number of neighbours is constant. In this case, the cost is different:

\[
\text{cost to rebuild} \leq \sum_{i=1}^{\Upsilon} [\text{transmission cost} + C \cdot \text{reception cost}] \\
\leq \Upsilon + C \cdot \Upsilon \in O(\Upsilon)
\]  (5.19)

The final step in querying is to send the rebuilt list back to the sink:

\[
\text{cost to send} \leq \Delta \cdot \text{transmission cost} + 3 \cdot N \cdot \text{reception cost} \\
\in O(N + N) \in O(N)
\]  (5.20)
This requires the passing of a message containing the event descriptions along the shortest path to the sink.

The full cost of querying is the summation of the above:

\[
\text{cost}_{\text{to query}} \in O(\text{cost of find} + \text{cost to rebuild} + \text{cost to send})
\]

\[\in O(N + \Upsilon \cdot N + N)\]

\[\in O(\Upsilon \cdot N)\] (5.21)

Or, for the case of uniform distribution:

\[
\text{cost}_{\text{to query}} \in O(\text{cost of find} + \text{cost to rebuild} + \text{cost to send})
\]

\[\in O(N + \Upsilon + N)\]

\[\in O(\Upsilon + N)\] (5.22)

Table 7.1 summarizes the algorithmic costs detailed above. If we consider the problem initially set

Table 5.1: Summary of the algorithm’s analysis

in Chapter 3, the goal was initially to minimize equation 5.23

\[
\text{network lifetime} \cdot ([\text{update frequency} \cdot \text{cost(update)} + \text{query frequency} \cdot \text{cost(query)}])
\]

(5.23)

In the worst case, the update to sink algorithm can perform better than the linked list algorithm. However, if we consider nodes to have a constant number of neighbours, then the linked list algorithm’s cost drops dramatically. Given these results, we expect the linked list algorithm to be superior to the update to sink algorithm in situations where query frequency is low.
Chapter 6

Experimental Setup

A simulation environment was constructed, and a number of tests were performed to check how the two algorithms compared under varying network conditions. The simulation environment was written in C++, and was designed to measure only the radio transmission cost of the network. The two algorithms were implemented, and the results are given here. This chapter will provide insight into the design of the experiments, and will showcase the results.

6.1 Variables

There are a number of variables which can affect the performance of the algorithms. A non-exhaustive list includes:

- Network Complexity - Total number of nodes in the network.
- Network Size and Shape - The area in which the network is deployed.
- Deployment Technique - Square grid, hexagonal grid, uniform random distribution, . . . .
- Sink Location - Near the center, near a corner, or a random node.
- Sensing Radius of Nodes.
- Communication Radius of Nodes.
• Sampling Rate of Nodes.

• Object Movement - Can be almost anything.

• Communication Routing Protocols - Lots of options.

Some variables (Network Complexity, Sensing Radius, etc.) directly affect the functioning of the network, while others (object movement, sampling rate, etc.) do not. The experimental setups proposed below are intended to show how some of these variables affect the algorithms.

6.2 Assumptions

For the simulation implementation, a few additional assumptions were required; these are outlined below.

6.2.1 Routing

The routing algorithm used is a shortest path routing algorithm. There are a variety of routing protocols, which fall into different categorizations [1]: Data-centric (gossiping, SPIN, directed diffusion, ...), Hierarchical (LEACH, PEGASIS, APTEEN, ...), Location-based (SMECN, GAF, GEAR, ...).

It is unclear which of these categorizations would be most favourable for track persistence. For this reason, the routing for all of these experiments will be done with as few hops as possible - that is, in a minimally hopped manner. Since all edge weights are positive, Dijkstra’s algorithm was used as an efficient technique for calculating the single-source shortest path problem [18].

6.2.2 The Radio Model

The simulation environment uses a radio model proposed by [46], and re-iterated in [2]. In this model, the cost of transmission can be expressed as:

\[
\text{cost}_{\text{to transmit}}(k) = P_{tx} \cdot (T_{\text{on-}tx}(k) + T_{st})
\]

where \( k \) is the number of bits to transmit, \( P_{tx} = 81\, mW \) is the power consumption of the transmitter, \( T_{\text{on-}tx}(k) = k \, b/1\, Mbps \) is the number of bits to transmit over the transmission rate, and
$T_{st} = 450 \mu s$ is the startup time.

The cost of reception is modeled as:

$$cost_{to\_receive}(k) = P_{rc} \cdot (T_{on\_rx}(k) + T_{st}),$$

where $k$ is the number of bits to receive, $P_{rc} = 81 \text{mW}$ is the power consumption of the receiver, $T_{on\_rx}(k) = k \text{ b/1Mbps}$ is the number of bits to receive over the reception rate, and $T_{st} = 450 \mu s$ is the startup time of the receiver.

Notice that in this model, reception cost is the same as the transmission cost. This is a minor modification to the model in [46], where the cost to receive a transmission is 2 - 3 times larger than the cost to transmit. The relative cost is hardware specific, and varies from less than equal to two- or three-times larger.

### 6.2.3 Communication

The problem of handling interfering bandwidth is very challenging, but in most cases, randomly assigning time segments for communication based upon node locality should avoid this problem, so for our simulations, we ignored interfering messages.

In these simulations, we wished to establish the quality of these algorithms, but not their fault tolerance, so communication was assumed to be reliable in all cases.

### 6.2.4 Sensor Nodes

As with communication, these simulations assumed a perfect sensing model. Sensors neither yield false positives nor negatives. Also, sensor nodes’ memory is not capped in these simulations. A low granularity time synchrony is also assumed. For the simulation, we assumed that each sensor senses once per tenth of a second. If less granularity in reporting is required, this value can be safely lowered. This assumption proceeds without loss of generality - it essentially notes that the sensing granularity is required to be no better than the level of time synchrony of the nodes.
6.3 Simulations

The proposed simulations are intended to compare the quality of tracking persistence algorithms. Each of these setups has been run against the proposed track persistence algorithms. For each value of the independent variable we are considering, the simulator was run 100 times.

6.3.1 Increasing Network Size

Specification

This experiment was built to show how the algorithms perform as the network size and complexity grows, while keeping the track length fixed. A rectangular grid deployment was assumed, and the minimum required number of nodes were used to provide coverage, so that while network complexity grew with network size, it grew at a proportional rate to network size. This experiment was run 100 times. A breakdown of the variables considered:

Dependent Variables

- Sink Location: Random
- Sensing Radius: 1.0 m
- Communication Radius: 2.0 m
- Sample Rate: 0.1 s
- Track Length: 10 m
- Network Density: 1 node per meter$^2$
- Querying Schema: A query arrives after the object halts

Independent Variables

- Network Size & Network Complexity
  - 10 m · 10 m using 100 nodes
  - 10 m · 20 m using 200 nodes
– 15 m · 20 m using 300 nodes
– ...
– 70 m · 70 m using 4,900 nodes
– 50 m · 100 m using 5,000 nodes

Note that the network was kept as close to square as possible.

Results

Figure 6.1 shows the results of comparing the combined update and query costs incurred for the two algorithms. The plotted values here represent the mean observations, and standard error bars are shown. The standard error bars show the standard error on the costs across the observations. The standard error on the results is large, and this makes sense - the sink location is decided randomly, the object position is also random, and the distance between these two highly influences the cost of the algorithm. As we can see, as the network size grows, the mean distance to the sink also increases, and the update to sink algorithm suffers the cost of sending repeated messages over this long distance. The linked list algorithm does not have the same overhead in message sending, so it outperforms the update to sink algorithm. More details showing the results of this experiment are given in Appendix A. This simple experiment highlights exactly how much improvement the use of a dynamic data structure can provide over a naive centralized approach.

6.3.2 Track Length

Specification

This experiment was built to show how the algorithms perform as the track length grows, while keeping the rest of the network fixed. As before, a rectangular grid deployment was assumed, and the minimum required number of nodes were used to provide coverage. A large network of 10,000 nodes was simulated for this experiment. A breakdown of the variables considered:

Dependent Variables

• Deployment Technique: Square grid
Figure 6.1: Comparison of query and update costs for networks of increasing size and complexity

- Sink Location: Random
- Sensing Radius: 1.0 m
- Communication Radius: 2.0 m
- Sample Rate: 0.1 s
- Number of Nodes: 10,000 nodes
- Network Size: 100m \times 100m
- Network Density: 1 node per meter$^2$
- Querying Schema: A query arrives after the object halts

Independent Variables
• Track Length: The object was simulated moving a certain distance before halting. The exact number of events that this incurs depends upon the actual path of the object, which was randomized, but the range of the number of events is quoted below.

− 10 m (20 - 40 events)
− 20 m (40 - 80 events)
− ... 
− 100 m (200 - 400 events)

Results

Figure 6.2 shows the results of comparing the combined update and query costs incurred for the two algorithms. As track length increases, the linked list algorithm significantly outperforms the update to sink algorithm (by an order of magnitude). This is the expected result, as the longer the list becomes, the more events there are to report, and the larger the overhead of constantly shipping data back to the sink. The linked list algorithm does not have as much overhead, so while its cost grows, it is at a much slower rate. An analysis of the update and query operations independently and more analysis of this experiment are given in Appendix B.

6.3.3 Query Frequency

Specification

The final experiment was designed to show how varying the query frequency affects the algorithms. As above, a rectangular grid deployment was assumed. A network of 2,500 nodes was used for this experiment. A breakdown of the variables considered:

Dependent Variables

• Deployment Technique: Square grid
• Sink Location: Random
• Sensing Radius: 1.0 m
• Communication Radius: 2.0 m
Figure 6.2: Comparison of summed query and update costs for increasing track lengths

- Sample Rate: 0.1 s
- Number of Nodes: 2,500 nodes
- Network Size: 50m \times 50m
- Network Density: 1 node per meter\(^2\)
- Track Length: 50 m

**Independent Variables**

- Query Frequency:
  - once per object lifetime
  - twice per object lifetime
... twenty times per object lifetime

The queries were spaced as evenly as possible, so that the last query occurred at the end of the objects life - if two queries occurred, the first would arrive after the object had moved 25 meters, and the second query would arrive after 50 meters.

Results

Figure 6.3 shows the results of this experiment upon the two algorithms. Again, the dots represent mean values, and the bars show the standard deviation of the multiple runs evaluated. The update to sink algorithm is predictably constant; query cost is negligible. The linked list algorithm however is forced to repeatedly query the dynamic data structure, and it after around six queries in an object lifetime, it becomes more expensive to use than the update to sink algorithm. This result confirms our expectations, in situations where query rates are high, the update to sink algorithm will excel.
Figure 6.3: Comparison of summed query and update costs for increasing query frequency
Chapter 7

Future Work & Open Problems

This thesis makes a first attempt at solving track persistence using WSNs, however there are a lot of places where further improvement is possible. In this chapter, we will make a number of suggestions of different algorithms or enhancements to algorithms that may better solve track persistence.

7.1 Fault Tolerance & Reliability

- This thesis made many assumptions regarding fault tolerance, and a thorough examination of the effects of the removal of these assumptions is a very important open problem. Node failure is frequent in WSNs, and network density is intended to provide reliability in spite of this; however for the algorithms in this thesis, some modification/extension would be required to meet fault tolerance goals.

- Another area that would be of interest for future research is into handling networks with non-ideal communication reliability. Assuming a probability model for all communications would drive up the communication costs overall, as acknowledgements or other checks would need to be inserted, and the cost of doing this is an open problem.
7.2 Hierarchy

There is a large amount of research that has addressed using hierarchical setups in WSN for the purposes of extending the life of systems, for improving scalability of algorithms, for data aggregations, etc. This thesis has not proposed any algorithms that would use these techniques, but there are a number of ways that they could be used.

- Powerful routing hierarchies could be used, as recommended in [24], to reduce the drain on nodes close to the sink. The radios in nodes for this model need to be more flexible than we have assumed in this thesis, as they should be able to transmit to different distances (probably up to an upper limit). This use of clustering nodes would require cluster heads to transmit messages directly to the sink, without using intermediate nodes. A more thorough examination of these techniques should improve time until first failure, and load balancing.

- Chen et al. used Voronoi neighbourhoods to dynamically build clusters [16]. These have the advantage of being event triggered. This could be applied to track persistence by dynamically building a cluster when an object enters a node. A nearby node would volunteer itself as cluster head, and nodes would send their messages to the cluster head. After the target leaves the cluster, it could dynamically de-cluster, and the cluster head could pass its message to the sink. A more thorough examination of this could prove a great improvement to the standard update to sink algorithm.

- Islam and Akl have proposed a technique for locally generating a set of connected dominating sets in [27]. It should be possible to modify the linked list algorithm to store the linked list only on nodes of the connected dominating set. Since Islam and Akl’s algorithm provides a set of connecting dominating sets, they can be rotated between to balance cost across many nodes in the network. There are two fundamental challenges in doing this: a connecting dominating set may not provide coverage, so nodes need to be able to efficiently route their events to the closest dominating node; and shifting the linked list data from one connected dominating set to another without incurring a large cost. If these problems could be overcome, then update operations would not be fully local, but would still be inexpensive and query costs would be reduced, as fewer nodes would need to be traversed for reconstruction.
Hierarchical algorithms have a few problems in general: if they are not carefully constructed, they can cause the nodes higher in the hierarchy to drain more quickly than the rest of the nodes in the network; they can also require additional hardware, which complicates analysis, among other problems. However, if they are carefully constructed, they may well be able to solve track persistence elegantly.

7.3 Mobile Agents

One of the fundamental problems with both the update to sink and the linked list algorithms proposed in this thesis is that nodes close to the sink will suffer greater drain than nodes further from the sink. The main reason for this is that nodes close to the sink have to relay messages to the sink from all over the network; the same subset of nodes must handle funneling messages from all over the network. One possible solution to this problem was given above - allow nodes to have variable communication distances, and sometimes use further nodes to send messages instead of the close ones. Another possible solution is to use mobile agents.

Assume, for argument’s sake, that the query response time isn’t important; when a query arrives, we are willing to wait indefinitely for the network to respond. Now assume that there is a mobile agent that moves around the network on a schedule, and collects all the aggregated data from the network, and at the end of its patrol can transmit this information to the sink.

The above scenario is contrived, but it shows how high energy mobile agents may solve the problem of communicating from distant parts of the network to the sink without placing additional cost on nodes near the sink. In [4] Alsalih et al. use mobile agents to manage transferring data from underwater nodes to an above-water sink; the mobile agents improve the lifetime of the network. This practical example shows that there may be significant promise in using such techniques for track persistence, and this could be an interesting future work.
7.4 Co-operative Targets

It may be possible in some cases to outfit the targets which we are tracking with a small amount of circuitry, which would allow the target to carry the data with it. In this scheme, the target would have a receiver, and some memory. As the target passes by a sensing node, the node transmits to the target that it is being sensed, and the target records this in its memory. At some point later, the memory of the target could be read to retrieve all of its historical track information. Adding circuits to vehicles for this purpose is readily possible, and would be an elegant solution to the problem. This solution could be favourable to using a GNSS like the global positioning system (GPS) because it could have a higher level of position granularity, it would not be affected by line-of-sight limitations, and it would not require communication to systems external to the WSN.

7.5 Multiple Covering Sets

An interesting implicit assumption of this thesis is that we always hope to have the highest level of granularity of path data possible. This may however not always be the case; in some cases, a low granularity path may be ‘good enough’, and these cases have some special interesting properties. In the situation where low granularity path data is good enough, then any nodes beyond the minimum nodes required to provide coverage should not be involved in normal operation. It is certainly possible to build a set of minimum covering sets (see Definition 3 - that is sets which cover all area of the network in close to as few nodes as possible). Using these subsets would allow us to extend the lifetime of very dense networks by using rotating sleep cycles.

7.6 Collaborative Aggregation

A possible improvement to the update to sink algorithm is to add a delay after observing an event, and reporting it. Adding a delay after observing the event would allow other events which occur nearby this event to be heard, and a digest to be created. In this way, updates would only be shipped to the sink when most of the updates nearby have been collected into one message. This could provide large savings to the algorithm, as well as reducing the number of messages which
nodes close to the sink would need to handle.

7.7 Co-operative Query Response Balancing on Linked Lists

In the linked list algorithm, one problem is that the node at either end of the track may suffer from significant drain, since query operations will result in a large message (containing the entire path) being transmitted to the sink. This enhancement intends to decrease this drain by breaking up the query message at various points along the linked list. There are a number of schemes for deciding when to send a message to the sink: at a fixed message size, a percentage of the way through the list, at significant time points (i.e., every hour - this is different than fixed message size since different objects may travel at different velocities), at locations in the network locally close to the sink, or at locations in the network with large remaining battery life.

7.8 Dynamic Query Positions

In [30], Kulathumani et al. define an extremely high quality object tracking algorithm, which has the feature that a query can arrive at any node in the network, not just the sink. This is achieved by establishing the central node of the network as a sort of “hub”; no matter where the query arrives, the distance to the center of the network is the furthest that a query must search. It might be possible to do something similar to this for track persistence, where the “hub” is treated as a sink, and queries that arrive dynamically make their way to the data through it. Such a modification of the problem specification may have a number of advantages for the future applications of track persistence in WSNs.

7.9 Multiple Object Tracking

This thesis has assumed only one object will be in the network at a time, and that query operations always relate to the most recently observed object. An open problem is how a network can handle multiple object identity fidelity, and Chapter 2 gives a discussion of the state of this problem in WSNs in general. An open problem for track persistence is to extend the handling of multiple
object identities to track persistence problems. References [32, 35, 30] all handle this problem well, and could be a good starting point for work in this area.

7.10 Partial Track Recovery

One possible problem that has not been handled in this thesis is that of partial track recovery. It may be that a query could only be interested in where the target has been in the last $l$ hours. It should be possible to perform this query at a lower cost than the full path reconstruction. With update to sink, cost is constant, so no improvement can be seen. Using a linked list type algorithm however, this could be achieved by only following part of the track.

Track persistence is a very interesting problem, and there is room for lots of development and growth in the algorithms used to solve it. In this chapter we have proposed several possible future works and open problems.
Chapter 8

Conclusion

8.1 Summary of Results

This thesis introduced and made an initial attempt to solve the novel WSN target tracking problem: track persistence. The goal was to provide not only an interesting problem, but also some algorithms focused on solving the problem as efficiently as possible. The primary result is that given a network with uniform node distribution and a low query frequency, the linked list algorithm outperforms the update to sink algorithm. If query frequency increases, then the update to sink algorithm is the preferable algorithm.

8.2 Contributions

This thesis has provided several contributions from original research. First, we introduced and defined a novel target tracking problem for WSN. Tracking the instantaneous position of an object in a WSN inspired the extension of the problem temporally, considering not only the instantaneous, but the historical position information of the target in the network. We made a first attempt at solving the problem, defined a baseline simplistic algorithm, and a more sophisticated algorithm which takes advantage of the distributed computing and memory available in WSNs. In addition
to the definition of these algorithms, we analyzed them both using worst case analysis, and using a simulation environment to show their performance in some particular cases. Finally, we applied the work of previous research in standard WSN target tracking to divide the track persistence into the two related problems: maintaining an internal representation of the track, and retrieving the internal representation of the track, which aided in both the definition of the problem, as well as in the analysis of our solutions.

8.3 Finally

Through this thesis, we have seen the possible improvement to be reaped by using distributed algorithms to solve problems in WSNs. Chapter 7 provides a great deal of future work for WSNs in track persistence algorithms. WSNs are gaining in research popularity all the time, and track persistence is a problem that could become well solved for many classes of networks. We hope that the future of WSN applications will take advantage of the contributions of this thesis.
Appendix A

Increasing Network Size & Complexity

In order to avoid disrupting the flow of the results chapter, a large amount of the analysis of the simulation data has been moved here.

A.1 Update Costs

The cost of performing only the required updates on the object moving can be seen in figure A.1. Considering the update cost separately from the query cost shows us two things that were expected. First, it confirms that the whole cost of the update to sink algorithm comes from updation, as query cost is a constant. Second, it confirms that the cost growth for the update to sink algorithm is about an order of magnitude higher than the linked list algorithm. To establish how well the algorithms balance their cost, we have plotted the average cost to nodes, which can be seen in figure A.3. We can see that for updating, the update to sink algorithm steadily grows in average node cost, while the linked list algorithm slowly decreases in average cost. If we consider figure A.4, we can see why linked list’s average cost is decreasing: as network size grows, the total cost isn’t growing as quickly as the number of nodes involved.
Figure A.1: Comparison of update costs as network size and complexity increase

A.2 Query Costs

The cost of just performing a query on the network can be seen in figure A.2. The plotted query cost is also roughly what is expected. The query cost for the update to sink algorithm is constant. The cost for the linked list algorithm grows steadily, and we can see from figures A.3 and A.4 that while the total cost for linked list grows, it balances the cost, and the number of nodes involved grows fast enough that the average cost to each node does not grow.
### APPENDIX A. INCREASING NETWORK SIZE & COMPLEXITY

**Query Costs vs. Increasing Network Size**

<table>
<thead>
<tr>
<th>Network Complexity (nodes)</th>
<th>Energy Cost (nJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>2000</td>
<td>2000</td>
</tr>
<tr>
<td>3000</td>
<td>3000</td>
</tr>
<tr>
<td>4000</td>
<td>4000</td>
</tr>
</tbody>
</table>

![Figure A.2: Comparison of query costs as network size and complexity increase](image)

**Average Update Costs vs. Increasing Network Size**

<table>
<thead>
<tr>
<th>Network Complexity (nodes)</th>
<th>Energy Cost per Node (nJ/node)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>2000</td>
<td>2000</td>
</tr>
<tr>
<td>3000</td>
<td>3000</td>
</tr>
<tr>
<td>4000</td>
<td>4000</td>
</tr>
</tbody>
</table>

![Figure A.3: Comparison of average update and query cost as network size and complexity increase](image)
Figure A.4: Comparison of the number of nodes involved in updating and querying as network size and complexity increase
Appendix B

Increasing Track Length

As in Appendix A, most of the details of analysis for the increasing track length experiment are given here.

B.1 Update Costs

Figure B.1 shows the mean, of all the runs, of the update costs for different track lengths for the two basic algorithms. As always, the update comparison is where the cost of using the update to sink algorithm becomes overwhelming. At the scale of Figure B.1, it is difficult to see, but the growth rate or the linked list algorithm’s cost is roughly an order of magnitude in cost slower than the update to sink algorithm’s cost. Figure B.3 shows the average cost incurred by a node, of performing an update operation for the two algorithms. Unsurprisingly, the average cost to a node under the update to sink algorithm is much higher than for the linked list algorithm. It is difficult to interpret the growth of the update to sink average cost in this plot, however. So for clarity, we have provided Figure B.4. Figure B.4 shows the distribution of samples taken from the simulator for the update to sink average node cost. Figure B.4 shows that the average cost seems to be growing very slowly, if at all, so we can say that the growth of the linked list average cost is growing faster than the update to sink average cost. Figure B.5 shows the rate of growth of the number of nodes used in an update operation. The number of nodes used for update to sink grows more quickly than the number used.
by the linked list algorithm.

![Update Costs vs. Increasing Track Length](image)

Figure B.1: Comparison of update costs as track length increases

### B.2 Query Costs

Figure B.2 shows the mean query cost for different track lengths. Predictably, the update to sink algorithm is plotted as a horizontal, with no growth; querying the sink is a constant cost operation. Figure B.3 shows that the average cost per node increases, and that the number of nodes involved increases quickly as well. This implies that the cost balances somewhat well across a number of nodes in the network, but that the cost grows more quickly than the number of nodes being utilized grows. We can note from this appendix, and Appendix A, that while the linked list algorithm well outperforms the update to sink algorithm in both cases, the linked list algorithm balances its cost much better to increasing network sizes than it does to increasing track lengths - there is a possibility
for enhancement using a scheme like a co-operative query response balancing linked list as described in Chapter 4.3.

![Query Costs vs. Increasing Track Length](image)

Figure B.2: Comparison of query costs as track length increases
Figure B.3: Comparison of average query and update costs as track length increases.

Figure B.4: A gradient histogram showing the average update cost of the update to sink algorithm. To read this graph, consider a battery value, track length pair. The shade of the graph at that point describes the quantity of experimental runs that resulted in that battery cost, at that track length.
Figure B.5: Comparison of the number of nodes involved in updating and querying as track length
Bibliography


