NETWORK ABSTRACTIONS FOR DESIGNING RELIABLE APPLICATIONS USING WIRELESS SENSOR NETWORKS

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the

Graduate School of The Ohio State University

By

Vinodkrishnan Kulathumani, B.E., M.S.

* * * * *

The Ohio State University

2008

Dissertation Committee:

Anish Arora, Adviser

Prasun Sinha

Tamal Dey

Paul Sivilotti

Approved by

Adviser

Graduate Program in Computer Science and Engineering

ABSTRACT

Applications of wireless sensor networks are moving from simply monitoring based to control based ones and from static network based to pervasive and mobility-centric ones. But while the applications are rising in scale and complexity, the underlying network is still resource-constrained and bandwidth limited, prone to contention and fading. Thus the demands of applications are growing at a faster rate than the resources in the underlying network. This dissertation has addressed the challenge of reliable application design using wireless sensor networks, by the design and implementation of network abstractions that bridge the gap between the application and the network and provide performance guarantees to applications.

The dissertation considers the reliable design of 4 wireless sensor network applications: (1) distributed pursuer evader tracking with requirement of eventual catch, (2) distributed pursuer evader tracking with optimal interception, (3) object classification and track monitoring and (4) distributed control of flexible structures. For each of these applications, we come up with an appropriate design considering limitations of the underlying network and characterize the network abstractions that meet application requirements. The network abstractions are then implemented appropriately sometimes using middleware services running in the form distributed / centralized programs, sometimes by suitably designing the network with the right density, placement of sensors or sometimes using both.

This dissertation is dedicated to my family

ACKNOWLEDGMENTS

It is a pleasure to thank the many people who made this thesis possible.

Foremost, I would like to thank my Ph.D. advisor, Dr. Anish Arora for giving me the opportunity to conduct focused research and for sharing with me a lot of his expertise and research insight. I greatly appreciate his insightful advice on research and presentation skills.

During the DARPA NEST project, I was fortunate to interact with and learn from many great scientists and researchers including Dr. Emre Ertin, Dr. Mohamed Gouda, Dr. Ted Herman, Dr. Sandeep Kulkarni, Dr. William Leal, Dr.Mikhail Nesterenko, and Dr. Prasun Sinha. Their wisdom has greatly enriched my Ph.D. experience.

Being a Ph.D. student in an active research university, I have also enjoyed and appreciated the advice and help from many other professors including Dr. Prasun Sinha, Dr. Tamal Dey, Dr. Paul Sivilotti, Dr. Randy Moses, Dr. Raj Jain, Dr. Wu Chi Feng and Dr. Rama Yedavalli. They have made my stay at The Ohio State University fun and fruitful.

I would like to specially thank Dr. Rajiv Ramnath, who has offered me some great advice on topics in research, life and career directions.

During my Ph.D. study, I have got the chance to work with many other fellow graduate students, Mukundan Sridharan, Sandip Bapat, Vinayak Naik, Hongwei Zhang,

Santosh Kumar, Murat Demirbas and Lifeng Sang, on shared projects and research problems. It was a wonderful experience. I would also like to thank my many other friends for their continued support during my life and study at Ohio State.

I am indebted to my parents Mr. K. K. Mani and Mrs. Lakshmi Mani for their unconditional love, encouragement and support. My parents have always put education as a first priority in my life. They taught me to value honesty, courage, and humility above all other virtues. My parents have always been there for me as an unwavering support. A special thanks to my brother Ram for always being there to cheer me up.

Last, but not least, I would like to thank my wife Gayathri. She has been my source of strength. Her love and support have been enabling me to focus on my research during the days and nights, the weekdays and weekends.

VITA

November 7, 1977	Born - Tenkasi, India
1999	B.E. Computer Engineering
June - September 2000	Intern, Microsoft Corporation
2001	M.S. Computer and Information Science
1999-2001	Graduate Teaching Associate, The Ohio State University
June - November 2001	Network Architect, Nayna Networks
March - June 2007	Research Intern, Los Alamos National Labs
2002-present	Graduate Research Associate, The Ohio State University

PUBLICATIONS

Research Publications

V. Kulathumani and M. Sridharan and R. Ramnath and A. Arora, "Weave: An Architecture for Tailoring Urban Sensing Applications across Multiple Sensor Fabrics", *MODUS, International Workshop on Mobile Devices and Urban Sensing, 2008*

V. Kulathumani and A. Arora, "Distance Sensitive Snapshots in Wireless Sensor Networks", *International Conference on Principles of Distributed Systems (OPODIS)*, 2007.

V. Kulathumani, M. Demirbas, A. Arora, and M. Sridharan, "Trail: A Distance Sensitive Network Service for Distributed Object Tracking", *European Conference on Wireless Sensor Networks (EWSN)*, 2007

M. Demirbas, A. Arora and V. Kulathumani, "Glance: A Lightweight Querying Service for Wireless Sensor Networks", *International Conference on Principles of Distributed Systems (OPODIS)*, 2006

H. Cao, E. Ertin, V. Kulathumani, M. Sridharan and A. Arora, "Differential Games in Large Scale Sensor Actuator Networks", *International Conference on Information Processing in Sensor Networks, IPSN*, 2006

E. Ertin, A. Arora, R. Ramnath, M. Nestkerenko, S. Bapat, V. Naik, V. Kulathumani, M. Sridharan, H. Zhang, H. Cao, "Kansei: A Testbed for Sensing at Scale", *International Conference on Information Processing in Sensor Networks, Special Track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, 2006

V. Kulathumani, P. Shankar, Y. M. Kim, A. Arora, and R. Yedavalli, "Reliable Control System Design Despite Byzantine Actuators", *Fifth International Conference on Multibody Systems, Nonlinear Dynamics and Controls (MSNDC)*, 2005

S. Bapat, V. Kulathumani, and A. Arora, "Analyzing the Yield of ExScal, a Large Scale Wireless Sensor Network Experiment", *International Conference on Networking Protocols (ICNP)*, 2005

S. Bapat, V. Kulathumani, and A. Arora, "Reliable Estimation of Influence Fields for Classification and Tracking in Unreliable Sensor Networks", *IEEE Symposium on Reliable Distributed Systems (SRDS)*, 2005

Y. M. Kim, A. Arora, and V. Kulathumani, "On the Effect of Faults in Vibration Control of Fairing Structures", *Fifth International Conference on Multibody Systems, Nonlinear Dynamics and Controls (MSNDC)*, 2005

A. Arora, E. Ertin, R. Ramnath, P. Sinha, S. Bapat, V. Naik, V. Kulathumani et al., "ExScal: Elements of an Extreme Scale Wireless Sensor Network", *11th IEEE International Conference on Embedded and Real time Computing Systems and Applications, (RTCSA)*, 2005

M. Demirbas, A. Arora, V. Mittal, and V. Kulathumani, "A Fault-Local Self-Stabilizing Clustering Service for Wireless Ad Hoc Networks", *IEEE Transactions on Parallel and Distributed Systems*, 2005

A. Durresi, R. Jain, N. Chandhok, R. Jagannathan, S. Seetharaman, and V. Kulathumani, "IP over WDM Networks", *Proceedings of IEEE Global Telecommunications Conference, (Globecom)*, Volume 4, pp. 2144-2149, 2001

M. Demirbas, A. Arora, V. Mittal, and V. Kulathumani, "A Fault Local Self-Stabilizing Clustering Service for Wireless Adhoc Networks", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 17, No. 9, pp. 912 - 922, 2006

A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V.Mittal, H. Cao, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora and M. Miyashita, "A Line in the Sand: A Wireless Sensor Network for Target Detection, Classification and Tracking", *Computer Networks Special edition*, Vol. 46, No. 5, pp. 605-634, 2004

V. Kulathumani, N. Chandhok, A. Durresi, R. Jain, R. Jagannathan, and S. Seetharaman, "Survivability in IP over WDM Networks", *Journal of High Speed Networks*, Vol. 10, No. 2, pp. 79-90, 2001

FIELDS OF STUDY

Major Field: Computer Science and Engineering

Studies in:

Networking	Prof. Anish Arora
Distributed Computing	Prof. Prasun Sinha
	Prof. Paul Sivilotti
Random Signals Analysis	Prof. Jose Cruz

TABLE OF CONTENTS

Page

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Applications of wireless sensor networks are growing at a faster rate than the resources in the underlying network. In the past ten years, wireless sensor networks have been typically used for many observation-based applications such as habitat monitoring and surveillance [2, 4] where the sensors gather a variety of information and this information is processed centrally or in a distributed manner. But now, applications are changing from being simply observation based to control based where the networks perform actuation and control and from being static network based to mobility-centric ones. The fact that sensor network systems can be really large scale and can be wireless makes them attractive for industrial and process control applications such as illumination control [60] and control of distributed parameter systems [12]. A specific example is the vibration control of a fairing during payload launch using embedded MEMS components based sensor actuator networks. Increasingly, sensors are being integrated with mobile devices like cell-phones and PDAs to support applications like on-line patient health monitoring, disaster relief, social networking, vehicular networking and mobile gaming and thus wireless sensor networks are becoming mobile. Additionally the scale at which the networks are operated have grown from 10s of nodes [2] to several thousands of nodes [4].

But while the applications are growing in scale and complexity, the underlying network still consists of resource constrained devices where energy and computational capabilities are at a premium. The underlying communication can fail in unpredictable ways due to bandwidth limitations, limitations of computational power and environmental effects. As a result, the network is vulnerable to faults such as information loss, delay and data corruption. The challenge in my research has been to design reliable applications despite the unreliability and the energy constraints in the underlying network.

In my research, I have dealt with 3 main classes of application namely classification, tracking and distributed control. I have considered tracking applications in two contexts: (1) observation of tracks of mobile objects at a central location, and a (2) distributed observation and control of tracks of mobile objects. Performance is critical in most of these sensor network applications. For example, (1) high accuracy and low latency are critical in classification and tracking applications that are often deployed in military settings to guard perimeters and valuable assets, and (2) distributed control systems have applications in space missions and nuclear plants where degradation of system performance may even compromise human safety.

> The challenge addressed by this dissertation is the design of sensor network applications that scale in a reliable and energy efficient manner despite the unreliability associated with the underlying network.

1.1 Overview of Approach

In order to address the above challenge, this dissertation proposes specifying the requirements of the application from the network in terms of suitable network abstractions and then implementing these abstractions. The implementation of the

abstractions is performed by either using middleware services that are programs running in the network or by simply designing the network by choosing the right density or placement of nodes. In this way, the network abstractions are wrappers around the network, that conform to certain specifications. The following are two notable aspects of this strategy.

1. The application layer does not have explicit knowledge of the underlying network parameters and is not involved in tuning of any network parameters. Thus this is not a cross layer design of the application and the network layer and the implementation of the application logic is therefore kept simple. The application implementation is shielded from the underlying network in the form of network abstractions.

2. The strategy allows the network abstractions to be implemented in a tunable way such that depending on application parameters, the network can adjust itself to meet the specifications and at the same time the overall system is energy efficient.

The network abstractions can be used to design reliable applications in a couple of usage models. In the first one, the application and the network layers are designed independently and in a de-coupled manner. The application requirements are translated to network specifications and either an implementation of this specification already exists or a network abstraction is implemented on top of the network layer, that satisfies this specification.

However, it may not be always feasible to assume that the specification can be implemented. This is because, there exists a tension between the application requirements and what the network can supply. The application will always demand as much resources from the network as possible, but meeting those may not be possible. For example, let us consider the right logic for classification of objects using a wireless sensor network. Transporting every sensor sample to a centralized base station to perform the classification is not a sensible idea as it would consume lot of resources. Moreover, it would cause lot of contention and therefore end up decreasing the overall system performance and the quality of the application performance. Therefore, a co-design of the application and network is needed. The application logic is designed as per network limitations. Once such an application is designed, the next challenge is to guarantee that the network meets these requirements and this is done using network abstractions.

Note that even in the co-design strategy, the application is still shielded from the low level network details and thus the implementation of the application is therefore kept simple. This dissertation explores this co-design of application and network using network abstractions, to design reliable applications using wireless sensor networks.

1.2 Contributions of the dissertation

The dissertation considers the design of the following applications using wireless sensor networks.

1. Distributed tracking application with *eventual* catch: In this application, one or more pursuers are required to *eventually* catch one or more evaders in a large region. An underlying sensor network is deployed in the region to detect

and track the pursuers and evaders in the network. The tracking application executes on the pursuer object and uses the sensor network to get the desired information about the evader objects.

2. Distributed tracking application with *optimal* catch: Here, we strengthen the requirements of the tracking application. In this application a large sensor network is deployed along the perimeter of a valuable asset. Intruders enter the perimeter with the intention of crossing over to the asset and the objective of the interceptors is to "catch" the intruders as far away from the asset as possible. The tracking application executes on the interceptor object and uses the sensor network to get the desired information about intruder objects.

3. Centralized classification and tracking: The goal of this application is to reliably classify and tracks objects moving through a region covered by a wireless sensor network at a central location.

4. Distributed vibration control: A flexible structure such as a beam or a fairing structure for satellite payload launching is deployed with sensors and actuators. The goal of the application is to detect and control the vibrations in the structure.

In this dissertation, for each of the above applications, we design an application logic given the network constraints. We then state requirements from the network that provide application guarantees and these requirements are translated to network abstractions. The network abstractions are implemented appropriately sometimes as middleware services running in the form distributed / centralized programs, sometimes by suitably designing the network with the right density, placement of sensors

or sometimes as both. In the following chapters, we formally state the application model and problem statement, formally specify the network abstractions and provide details of their implementations. Before that, in the following subsection, we provide a brief overview of the same.

1.2.1 Distributed tracking application with eventual catch

An underlying sensor network is deployed in the region to detect and track multiple pursuers and evaders in a region. The tracking application executes on the pursuer object and uses the sensor network to get the desired information about the evader objects. We assume that every pursuer object is assigned to a particular evader. The goal of the tracking application executing on every pursuer object is to eventually catch the evader that is assigned to that pursuer object.

Application Logic

If the underlying network can provide exact information about object locations continuously and instantaneously (with no delays) and if the pursuer is faster than the evader, then an *eventual* catch is trivial. But getting such perfect information is infeasible using a wireless sensor network. In our analysis we determine sufficient conditions on the resolution, latency and rate of information about the evader being tracked in order to satisfy *eventual* catch. Specifically, we show that *eventual* catch is satisfied if the error in the estimate of distance to the evader decreases linearly with distance between pursuer and the evader, if the rate at which this information is supplied to the evader decreases linearly with distance and if the staleness in the information supplied decreases linearly with distance, where the constants of proportionality depend on the relative speeds of the pursuer and the evader.

Network abstractions and their implementation

The above application imposes a distance sensitivity requirement on the network in terms of latency, rate and resolution. We formally define 3 different network abstractions namely distance sensitive latency, distance sensitive rate and distance sensitive resolution. In order to implement these abstractions, we design a middleware service that periodically delivers the global snapshot of the system to all nodes in the network where the snapshots satisfy the required *distance sensitivity* properties.

Our service is easily adapted to allow snapshots to be delivered only to a subset of nodes as opposed to all nodes. They are memory efficient, and are readily realized in networks with irregular density, networks with arbitrary sized holes, imperfect clustering, and non unit disk radios. We quantify the maximum rate at which information can be generated at each node so that snapshots are periodically delivered across the network, the service can of course be operated at lower rates than these. The service does not require global time synchronization or localization.

Finally, we show our snapshot service is used in the context of the distributed tracking application and results in satisfying the sufficient conditions for *eventual* catch.

1.2.2 Distributed tracking application with optimal catch

In this application a large sensor network is deployed along the perimeter of a valuable asset. We strengthen the requirement of tracking in this application by requiring an *optimal* catch. In other words, intruders enter the perimeter with the intention of crossing over to the asset and the objective of the interceptors is to "catch" the intruders as far away from the asset as possible. The interceptors use

the underlying wireless sensor network to gather information about intruders and intercept the intruders in an optimal manner.

Design of Application Logic

It is known that there exist optimal min-max strategies for the intruder interceptor application when perfect state information is available to the objects. However it is unreasonable to assume in a constrained sensor network that information about intruder objects is instantaneously provided to the interceptor objects. Therefore we study the sampling rate requirements for the interceptor object based on the state of the system to preserve the optimality of the pursuit strategy.

We prove that latency with which the *exact* state of the intruder is provided to the requesting interceptor should decrease proportionally with decreasing distance between interceptor and intruder in order to to guarantee that the evader does not have an incentive to deviate from its strategy to move directly to the predicted intercept point.

Network abstraction

The above requirement by the application imposes a distance sensitivity network abstraction that guarantees that *complete* information about nearby objects will be available at a lower latency than farther objects.

We implement the distance sensitivity abstraction using network protocol *Trail*. We use geometric ideas to design an energy-efficient, fault-tolerant and hierarchy-free WSN service, *Trail*, that supports tracking-based WSN applications. The specification of *Trail* is to return the location of a particular object in response to an in-network

subscriber issuing a *find* query regarding that object. *Trail* has a *find* cost that is linear ($O(d_f)$) in terms of the distance (d_f) of the subscriber from the object. To this end, *Trail* maintains a tracking data structure by propagating mobile object information only locally, and satisfying the distance sensitivity requirement for the track updates. The amortized cost of updating a track when an object moves a distance d_m is $O(d_m * log(d_m))$.

A basic *Trail* protocol can be refined by tuning certain parameters, thus resulting in a family of *Trail* protocols. Appropriate refinements of the basic *Trail* protocol are well suited for different network sizes and *find*/*update* frequency settings: One refinement is to tighten its tracks by progressively increasing the rate at which the tracking structure is updated; while this results in updating a large part of the tracking structure per unit move, which is for large networks still *update* distance sensitive, it significantly lowers the *find* costs for objects at larger distances. Another refinement increases the number of points along a track, i.e., progressively loosens the tracking structure in order to decrease the *find* costs and be more *find-centric* when object *updates* are less frequent or objects are static. Moreover, *Trail* scales well to networks of all sizes. As network size decreases, *Trail* gradually eschews local explorations and updates and thus increasingly centralizes *update* and *find*.

We evaluate the performance of *Trail* by simulations in a 90×90 sensor network and experiments on 105 Mica2 nodes in the Kansei testbed[3]. This implementation has been used to support a distributed intruder interceptor tracking application where the goal of the interceptor is to catch the intruders as far away from an asset as possible.

1.2.3 Classification of intruders and monitoring their tracks

The goal of this application is to classify and track objects such as humans, soldiers (humans with metallic objects), small cars, SUVs and army vehicles at a centralized base station. The network is equipped with sensors such as magnetometers, microphones and motion sensors.

Application Logic

Techniques such as transmitting every sample sensed by individual nodes to the base station or complex signal processing at nodes to perform classification may not be feasible to implement in resource constrained sensor networks. Therefore we use a technique by which each node only has to send a binary information upon 'detection' in order to perform classification and tracking.

The *influence field* of an object j with respect to a given sensing modality is the region surrounding j where it can be "detected" by a sensor of that type [10]. This region thus depends on both the characteristics of the object, such as its size and shape, as well as the sensing modality being used. Differences in the area and/or shape of the influence fields of different objects can be used to distinguish between them. The influence field feature is thus useful in sensor network applications for surveillance, where typical tasks include detection, classification, and tracking of various types of objects. To estimate the influence field, each node merely has to detect a binary presence of an object; network-based aggregation of these bits yields the influence field without requiring substantial or complex node operation. The influence field feature is thus well suited for wireless sensor network applications where individual nodes are constrained due to limited processing, sensing and communication capabilities.

Estimation consists of calculating the area and the shape of the influence field. If we assume sensor node density ρ exceeds some lower bound that depends on the targets at hand, the estimation of the area A of the influence field of j is effectively reduced to counting the number of nodes that detect j. The key challenge in realizing the influence field is the unreliability of wireless sensor networks. We deal with both node and network faults when estimating the influence fields. The requirement to separate influence fields of different objects and track them, gives rise to the following network abstractions.

We consider node faults of two types: false positives and false negatives can be generated by a node regarding a particular detection due to many factors such as environmental variations or node failures. We also consider two types of network faults: channel fading and channel contention. We model the channel fading effect as a constant probability of message loss over each hop. We model the channel contention probability as a function of the number of nodes trying to send a message simultaneously and the number of slots available.

Network abstractions and implementation

1. *False report insensitivity*: The net effect of a node fault is modeled as a node missing a detection of an object, i.e. a false negative or a node asserting a detection when there is no object, i.e. a false positive. Based on our experimental observations for node faults across a wireless sensor network deployment, we assume that failures at nodes occur independent of each other and of the objects. This leads to the following abstraction *False report insensitivity*.

False report insensitivity implies that the network guarantees separation between estimated influence fields of objects despite false negatives and false positives.

Implementation: Insensitivity to false reports are implemented by identifying a minimum density of deployment that ensures separation between influence fields with high probability enough to satisfy application accuracy. In scenarios where density cannot be met, we use *temporal aggregation* technique to ensure separation between influence fields.

2. *Distance insensitivity:* Recall that the channel fading effect is modeled as a constant probability of message loss over each hop. Therefore messages from nodes closer to the aggregator have lower probability of failure. Hence the estimated influence field of a small object close to the aggregator can overlap with that of a large object far away. We therefore need to compensate for the effect of distance of an object from the aggregator during its estimation.

   Distance insensitivity implies that the network guarantees that the influence field of an object estimated at the classifier is invariant with distance of the object from the classifier.

   Implementation: In order to achieve distance insensitivity, we use a probabilistic reporting technique by which nodes report a detection with probability directly proportional to their distance from the central classifier.

3. *Contention insensitivity:* As the number of nodes that report a detection increases, there is increased channel contention leading to message losses. As a result, after a point fewer detections are reported for larger objects compared

with smaller objects. We therefore need to compensate for the effect of influence field size on the correct estimation.

Contention insensitivity implies that the network guarantees that the separation of influence fields of objects is independent of the size of the influence fields of the objects.

Implementation: In order to achieve contention insensitivity, we use the following techniques. (1) Influence field suppression: In this technique, the influence fields of all objects are suppressed by each node reporting only a fraction of the actual reporting nodes. By doing this, the maximum influence field may be decreased such that contention does not result in inversion. (2) Temporal segregation: The influence field suppression may fail to avoid inversion if the difference between the influence field of the smallest and largest object under a given modality is so large that the influence field of the smallest object may not be decreased any more. In such cases, we use temporal segregation by which the reporting of detections are separated in time to avoid contention.

We then show how the different network abstractions can be composed to guarantee separation of influence fields under a combination of faults. We also show how the network abstractions can be used to achieve classification in the presence of multiple objects. We experimentally validate the reliable estimation of influence fields by using the implementation of these abstractions to accurately classify and track persons, soldiers, and vehicles in *A Line in The Sand* [2].

1.2.4 Distributed vibration control

The goal of this application is to sense and control vibrations in structures such as a beam or a fairing structure used for launching of satellite payloads. The requirement is to provide mission critical stability despite unreliabilities in the underlying network.

Application Logic

The system is modeled as a marginally stable linear time-invariant multi-variable system S with m sensor-actuator pairs as shown below.

$$\dot{x} = Ax + Bu \tag{1.1}$$

$$y = Cx \tag{1.2}$$

where x is an n-dimensional state vector $[x_1, x_2, \cdots, x_n]^T$, u is an m-dimensional actuator vector, B is an $n \times m$ dimensional matrix and the individual sensor-actuator pairs are collocated. Since S is marginally stable, A has eigenvalues on the imaginary axis. Since the individual pairs of sensors and actuators are collocated, we have the following condition.

$$B = C^T \tag{1.3}$$

Starting at any state, without any control being applied the system maintains its energy as it is marginally stable.

We apply the following local on-off output feedback control law to stabilize the system.

$$u_i = \alpha \times sign(y_i), \quad i = 1....m \tag{1.4}$$

where α is less than zero. Further u_i equals zero when y_i is zero. Thus a correct actuator can have 3 possible control values 0, $-\alpha$ and α. We choose $|\alpha|$, the magnitude of the actuator force, to be the maximum force that an actuator can apply and assume that this is the same across all actuators.

Sensor actuator components being unreliable, the various component failures can manifest themselves in the form of arbitrary actuator behavior in which case their effect on the underlying systems can be severe. One of the methodologies for the design of fault-tolerant control systems involves real-time fault detection, isolation and control system reconfiguration [11, 39, 61, 44, 13]. An appropriate action is taken after the diagnosis of the faults. This method still leaves the following challenges. The hardware itself can be faulty causing the actuators to fail-stop and offer no control or debond from their surface causing them to offer incorrect control. It is sometimes also not feasible to integrate the fault detection, diagnosis and reconfiguration in dynamical systems particularly when the available reaction time is limited. The underlying fault detection service is itself vulnerable to faults in the middleware and software services. This leads us to consider a Byzantine model for the actuator faults. A Byzantine actuator can produce an arbitrary control input to the plant at all times.

Byzantine insensitivity network abstraction

The question then arises as to how can the control system be guaranteed to be stable when a fraction of the actuators in the network are Byzantine. *Byzantine insensitivity* implies that stability of the system is guaranteed despite a fraction of the actuators being Byzantine.

Implementation of Byzantine insensitivity

Given a maximum of k Byzantine actuators, we first determine a necessary number of actuators to guarantee asymptotic stability. Then for a 2 dimensional system, we determine a sufficient number of redundant actuators and determine conditions on placement of the actuators that will guarantee asymptotic stability of the control system. We demonstrate our approach using a beam vibration control application as a case study.

1.3 Organization of this Thesis

The rest of this thesis is organized as follows.

In Chapter 2, we describe a pursuer evader tracking application with requirement for eventual catch, and describe a distributed snapshot service for wireless sensor networks, that supports this application. In Chapter 3, we tighten the requirement of the pursuer evader tracking application to satisfy an *optimal* catch and describe a distributed sensor network based tracking service, *Trail*, that supports this application. In Chapter 4, we design a surveillance application that classifies and monitors the tracks of objects in a region. In Chapter 5, we describe the design of a reliable vibration control application that guarantees asymptotic stability despite Byzantine actuators. We discuss conclusions and future work in Chapter 6.

CHAPTER 2

DISTRIBUTED PURSUER EVADER TRACKING APPLICATION WITH EVENTUAL CATCH

Wireless sensor networks have enabled new surveillance systems, where sensor nodes equipped with processing and communication capabilities can collaboratively detect, classify and track targets of interest over a large area. These surveillance systems make it viable to use the state information collected through the sensor network to guide mobile agents to achieve surveillance goals such as target capture and asset protection. A sensor network surveillance system has the advantage of giving the mobile agents access to the global information so that they can optimize their motion for pursuit tasks, as opposed to resource-intensive search and map building tasks. That said, using sensor networks to implement "active" surveillance strategies introduces new challenges as well. Target track information obtained by local processing of sensor information needs to be routed to mobile agents through multi-hop communication links, which results in delays, message losses and random arrival times of the packets carrying track information. In addition, as the network is scaled in size, high throughput rates for all pursuers cannot be sustained at all times, which necessitates a network communication strategy that adapts to pursuer information requirements.

In this chapter, we consider a pursuer evader tracking scenario where multiple pursuers and evaders exist in a bounded region. We design a pursuer control strategy and associated network support that guarantees an *eventual* catch despite network effects. We assume target information has been established through local fusion of sensor data. This target information is communicated through the multi-hop wireless network infrastructure to a pursuer object, which calculates the next move based on evader and its own state. In summary, we show that *eventual* catch can be achieved if network delays, update periods and error in the estimation of the target location scale linearly with distance between the pursuer and the evader and describe a wireless sensor network service that satisfies these conditions.

In Section 2.1, we describe the system model. In Section 2.2, we derive the sufficient conditions for successful pursuit (that result in eventual catch) and translate these to network abstractions. In Section 2.3, we describe a snapshot service for wireless sensor networks that implements these network abstractions. In Section 2.4, we show how the snapshot service can be used to satisfy successful pursuit requirements. In Section 2.5 we discuss work related to our snapshot service and pursuer evader tracking applications. In Section 2.6, we present a summary.

2.1 System Model

In our system model one or more pursuer objects are interested in *eventually* catching all evader objects spread over a bounded region. The pursuer objects are assisted by a wireless sensor network that provides the state (location) of evader objects to the pursuer objects. We assume that every pursuer is assigned to track at most one evader at a time. We assume that all the objects in the network are

uniquely identifiable. A catch is said to occur when when the distance d_{pe} between a pursuer p and an evader e assigned to p is smaller than ϵ where ϵ approaches 0.

In our analysis we consider the case where one pursuer p has been assigned to an evader e and this assignment holds until the evader is caught. Note that an eventual catch in this scenario is sufficient to show that all evaders will be eventually caught. Let v_p and v_e denote the pursuer and evader speeds respectively. The strategy of pursuer p is as follows: given a location x_e for the evader, follow the straight line to x_e.

2.2 Sufficient Conditions for Eventual Catch

If the network can provide the exact location of the evader e to pursuer p at all times with no delay in transmission, then it is trivial that $v_p > v_e$ is sufficient to obtain eventual catch. In this section, we characterize tighter sufficient conditions for an eventual catch.

Let $z(t)$ denote the error in the location of evader at any time t. Let $d_{pe}(t)$ denote the distance between the pursuer and evader at time t. Let δt denote the staleness in the state of e supplied to p at time t. Let $I(t)$ denote the maximum interval after time t at which location of e can be provided to p. Let $\alpha = \frac{v_p}{v_e}$ and $\alpha > 1$.

Theorem 2.2.1. *Evader e will be eventually caught by pursuer p if there exists constant* $k > \frac{\alpha+1}{\alpha-1}$ *and time* T_o *such that the following conditions hold at all* $t > T_o$*:*

1. $z(t) < \frac{d_{pe}(t)}{k}$
2. $\delta(t) < (\frac{d_{pe}(t)}{v_e}) * (1 - \frac{\alpha+k+1}{\alpha*k})$
3. $I(t) < (\frac{d_{pe}(t)}{v_p}) * (\frac{k+1}{k})$

Proof. Let l_p denote the location of pursuer p at time t such that $t > T_o$. Let $\hat{l_e}$ be the location of evader supplied to p at time t. If the maximum error and staleness in this location satisfy conditions 1 and 2 stated above, the actual location l_e of evader e at time $t - \delta(t)$ is within a ball of radius $\frac{d_{pe}(t)}{k}$ around $\hat{l_e}$. At time t, evader e is within a ball of radius $\frac{d_{pe}(t)}{k} + \delta(t) * v_e$ around $\hat{l_e}$.

At time t, the action of the pursuer is to move towards $\hat{l_e}$. Let us assume that next information about the evader is available to p only after reaching $\hat{l_e}$ (or the pursuer may even choose not to consume any information in this interval).

The maximum distance between l_p and $\hat{l_e}$ is bounded by the following inequality.

$$dist(l_p, \hat{l_e}) < d_{pe}(t) * (\frac{k+1}{k}) \tag{2.1}$$

It follows using Eq. 2.1 that the maximum time $I(t)$ to reach $\hat{l_e}$ is bounded by the following inequality.

$$I(t) < (\frac{d_{pe}(t)}{v_p}) * (\frac{k+1}{k}) \tag{2.2}$$

Let $d_{pe}(t + t')$ denote the distance between p and e at time $t + t'$. Using Eq. 2.2, we have the following inequality.

$$d_{pe}(t + I(t)) < (\frac{d_{pe}(t)}{v_p}) * (\frac{k+1}{k}) * v_e + \frac{d_{pe}(t)}{k} + \delta(t) * v_e \tag{2.3}$$

In order for eventual catch, we require that $d_{pe}(t+t') < d_{pe}(t)$. Using this, we get the following inequality.

$$\delta(t) < (\frac{d_{pe}(t)}{v_e}) * (1 - \frac{\alpha + k + 1}{\alpha * k}) \tag{2.4}$$

Note that for $\delta(t) > 0$, we require that $\alpha + k + 1 > \alpha * k$. Using this we get the following condition:

$$k > \frac{\alpha + 1}{\alpha - 1} \tag{2.5}$$

Note that $\alpha > 1$.

□

Using the above theorem, we showed that error, latency and update periods that decrease linearly with distance are sufficient to achieve eventual catch in pursuer tracking application. We now define 3 network abstractions that capture these requirements and describe a network service that implements these abstractions.

2.3 Snapshot service for wireless sensor networks

In this section, we systematically design wireless sensor network algorithms that periodically deliver distance sensitive snapshots to all nodes in the network. Our algorithms are easily adapted to allow snapshots to be delivered only to a subset of nodes as opposed to all nodes. They are memory efficient, requiring only $O(3^f * log(N^{\frac{1}{f}}) * m)$ bits per node. They are readily realized in networks with irregular density, networks with arbitrary sized holes, imperfect clustering, and non unit disk radios. We quantify the maximum rate at which information can be generated at each node so that snapshots are periodically delivered across the network, the algorithms can of course be operated at lower rates than these. For our services, global time synchronization is not required; a local notion of time however is needed to ensure fair scheduling of transmission of nodes.

Overview of algorithms and main results

Consider an ideal network where nodes are embedded in a virtual 2-d grid such that there is exactly one node at each grid location and that each grid node can reliably reach each of its neighbors in the grid and no others. We first describe an algorithm to deliver snapshots with distance sensitive resolution and latency at all nodes at the maximum rate. To obtain distance sensitive resolution, instead of dispersing the individual state of each node, we map the state of nodes into aggregate values of non-overlapping regions. We then deliver snapshots across the network such in a snapshot delivered to a node j, the size of a region into which a node i is mapped increases as $dist(i, j)$ increases. Thus, the resolution with which i is represented in the snapshot decreases as $dis(i, j)$ increases. To achieve this kind of snapshot delivery, we refine the clustering of nodes into a hierarchical one with a logarithmic number of levels as the network size. The basic idea is that a clusterhead at each level compresses data from all nodes in that level into m bits. Thus, the data aggregated at each level is represented by the same number of bits. At higher levels, the data is summarized with a coarser resolution as these levels contain more nodes.

In order to ensure uniform distance sensitive latency, we regulate the flow of information in all directions by proceeding in rounds. Intuitively, a round is a unit of time when information is exchanged between any level 1 cluster and all its neighboring clusters. Our scheduling and other protocol actions at each step are such that information is propagated across the network in a pipelined manner; by this, new information can be generated at a node as soon as previous information has been dispersed only to its local neighborhood as opposed to the entire network.

> In the first algorithm, in a snapshot S of the network delivered to node j the resolution of the state of a node i in S decreases as $O(d^f)$, the staleness of the state of a node i in S is $O(3^{2f} * m * log(n) * d)$ and the average network communication cost for N samples is $O(3^f * log(n) * N * m)$.

To achieve distance sensitive rate, we schedule the delivery of aggregated information at each level such that information of higher levels is delivered over a larger interval as opposed to lower levels. To do this, we allocate an exponentially increasing number of bits per message to lower level aggregates so that they are delivered at a faster rate.

> In this second algorithm, the average communication cost per N samples (one from each node) is $O(3^f * N * (m + log(n/m)))$, the maximum staleness in the state of a node i received by a snapshot at node j is $O((m + log(n/m)) * d)$ where $d = dist(i, j)$ and the maximum interval between when a node j receives the state of node i is $O((m + log(n/m) * d)$.

Our algorithms allow for a user-pluggable aggregation function. We require only that the function, say *f*, be idempotent and satisfy the following *decomposability* property: $\forall a, b,\ f(a \cup b) = f(f(a) \cup f(b))$. Examples of such functions are average, max, min, count and wavelet functions.

We then relax our regularity assumptions and describe how our algorithms handle the cases of non uniform density, non uniform radio range and holes of arbitrary sizes in the network. The case of over density is modeled as certain virtual grid locations containing more than 1 node. In the case of holes in the network, we show that our algorithms achieve distance sensitivity in terms of the shortest communication path between any two nodes as opposed to the physical distance.

2.3.1 Network model and problem statement

In this section, we present the system model and a generalization of the concept of snapshots based on distance sensitive properties.

Network model

The sensor network consists of N nodes that are embedded in an f-dimensional space. We let n abbreviate $N^{\frac{1}{f}}$. The nodes induce a connected network where each communicate at W bits per second. Nodes are synchronized in time. Each node j periodically generates m bits of (sensor) information, and maintains a data structure comprising the most recent state of nodes (or partitions of nodes) and a timestamp associated with that state. We let $j.X_i$ and $j.T_i$ (respectively, $j.X_p$ and $j.T_p$) refer to the state of node i (respectively, partition p) and the timestamp of that state at node j.

For ease of exposition, we restrict our attention to sensor networks that form a 2 dimensional grid with a node at every grid location. We further assume that node communication follows an idealized disk model: specifically, each node can communicate reliably with all its neighbors in the grid and with no others. We define the neighbors of node j to be the ones to its north, east, west, and south and also to its northeast, northwest, southeast, or southwest that exist in the grid; we denote these (up to 8) neighbors as $j.n$, $j.e$, $j.w$, j, s, $j.ne$, $j.nw$, $j.se$ and $j.sw$ respectively. In Subsection 2.3.4, we remove each of these restrictive assumptions.

Generalized snapshots

Let's begin by considering global snapshots where state is maintained for each node.

Definition 1 (Snapshot S)**.** A snapshot S is a mapping from each node in the network to a state value and a timestamp associated with that state value.

A consistent snapshot is one where the timestamps associated with each state value are all the same. The *staleness* of a state value in S is the time elapsed between its timestamp and the current time. We now consider a generalization where state values do not necessarily correspond to the same instant of time but their staleness enjoys a distance sensitive property.

Definition 2 (Snapshots with distance sensitive latency)**.** A snapshot S received by a node j has *distance sensitive latency* if the staleness in the state of each node i in S decreases as $dist(i, j)$ decreases.

We now further generalize the notion of snapshots so that state is associated with partitions p of the network as opposed to individual nodes. Let P be a partitioning of the network.

Definition 3 (Snapshot S of P)**.** A snapshot S of P is a mapping from each partition p in P to a state value and a timestamp associated with that state value.

The generalized definition is useful even if P is not a total but a partial partition, i.e., not all nodes are represented in the snapshot. More to the point, the state and timestamp of each p in S intuitively represent the aggregate state of all nodes in the partition and the aggregate timestamp. We assume that the timestamp of recoding the state of all nodes in any partition p is the same, and refer to this common value as the aggregate timestamp. Note that the aggregate timestamp of different partitions may be different.

As there may not exist a mapping from the aggregate state of a partition to the exact state of individual nodes that was recorded for the purpose of computing the aggregate, the latter may be estimated using some function of the state of the partition. The *resolution* of the state of a node in a snapshot is an inverse measure of the error between the state of the node that was recorded and the aggregate state of the partition p that it belongs to.

We are interested in snapshots where the increase in the error in the state of a node is bounded by the size of the partition p to which it belongs and thus the decrease in resolution of the state of a node is bounded by the size of p. This leads us to consider a generalization where the resolution of the state of a node increases as distance decreases.

Definition 4 (Snapshots with distance sensitive resolution)**.** A snapshot S of P received by a node j has *distance sensitive resolution* if the resolution of the state of each node i covered by P increases as $dist(i, j)$ decreases.

Informally speaking, the size of the partition to which the state of node i is mapped into in a snapshot received at j increases as $dist(i, j)$ increases. Therefore the resolution with which i is represented in S decreases with distance. In other words, j has a *myopic* or *fisheye* view of the network.

Finally, we consider a generalization where the rate at which state of the nodes is reported to a node decreases as the distance of the nodes increase.

Definition 5 (Snapshots with distance sensitive rate)**.** A node j receives snapshots of P with *distance sensitive rate* if the rate at which the state of each node i covered by P is updated in snapshots received by j increases as $dist(i, j)$ decreases.

Note that the concept of distance sensitive rate is orthogonal to that distance sensitive latency. In the latter staleness in the state received decreases with distance but fresh information arrives at the same rate at all nodes, where as in the former, the state of nearby nodes is reported more often than farther nodes.

2.3.2 Distance sensitive resolution and latency

We partition the network into a hierarchical one with a logarithmic number of levels, which are numbered $0..(log_3 n)$. A 3 by 3 set of 9 level r clusters form a cluster at level $r+1$, as illustrated in Fig. 2.1. Each node belongs to one cluster at each level, and each cluster has a clusterhead which is the center node of that cluster. A clusterhead at level r is also a clusterhead at levels $0..r-1$.

Overview of algorithm

The basic idea is that a clusterhead at each level compresses data from all nodes in that level into m bits. Thus, aggregated data at each level is represented by the same number of bits. At higher levels, data is summarized into a coarser resolution as the levels contain more nodes. The aggregated data is then dispersed to all nodes at that level. This solution suffers from a multi-level boundary problem however: two nodes could be neighbors but belong to a common cluster only at level $r \gg 1$. Thus despite being neighbors, both nodes get a summary of the other at a much coarser resolution than desired. The multi-level boundary problem is illustrated in Fig. 2.1, where nodes j and k are neighbors at level 0 but belong to a common cluster only at level 3. To avoid this problem, we disperse a summary computed at level r not only to nodes in level r cluster, but also to nodes in all neighboring level r clusters. This algorithm can be implemented in multiple phases; aggregates at each level are

computed and dispersed sequentially. When implemented sequentially, however, new samples can be generated at each node only after data is dispersed at all levels. To avoid this constraint, we use the following pipelined implementation described next.

Notations

Let $j.L$ be the highest level for which j is clusterhead. Note that there are at most 8 neighbors at each level for each node in the grid topology. We implement virtual trees along the structure at each level. To describe these trees, we will need the following definitions.

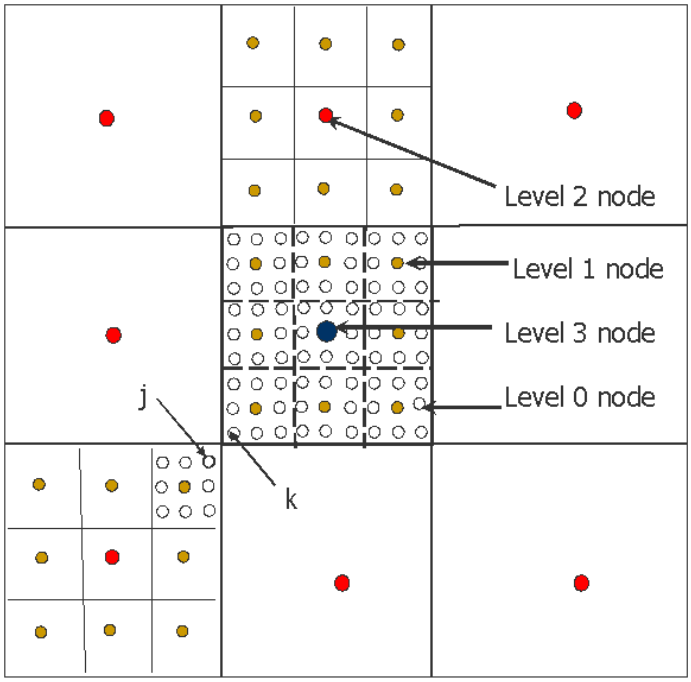


Figure 2.1: Hierarchical clustering

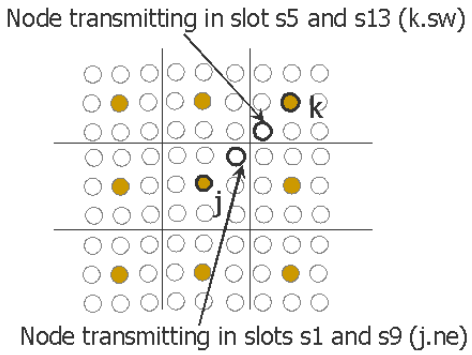


Figure 2.2: Neighboring level 1 clusters

Definition 6 ($tree(k, j)$)**.** $tree(k, j)$, where j is a level k clusterhead, is a level k tree formed with j as root and spanning all nodes in the level k cluster of j and all level k clusters that are its neighbors.

Definition 7 ($j.in(k, y)$)**.** For each $tree(k, y)$ that j belongs to, $j.in(k, y)$ is j's parent towards root y.

Definition 8 ($j.out(k, y)$)**.** For each $tree(k, y)$ that j belongs to, $j.out(k, y)$ is the set of j's descendants on the tree.

Definition 9 ($M(k, y)$)**.** $M(k, y)$ is the level k summary computed by a level k clusterhead y.

In Fig. 2.3, a level 1 tree rooted at j is shown as an illustration. The level 1 tree extends up to all level 0 nodes in its own cluster and level 0 nodes in the 8 neighboring level 1 clusters. The trees represent the distance up to which an aggregate at any level is propagated.

Schedule

We schedule the nodes to transmit in rounds. A round is a unit of time in which information is exchanged between a level 1 clusterhead and all of its 8 neighboring

level 1 clusterheads. Each round is divided into multiple slots. In the first slot, all level 1 clusterheads transmit. In the remaining slots, all level 0 nodes in each cluster transmit twice. The second transmission by a node within a round takes place after all its 8 neighbors have transmitted at least once. Intuitively speaking, during the first turn for a node, information is communicated outwards from the clusterhead. In the next turn for the node, information is communicated to the level 1 clusterhead that the node belongs to. A simple non-interference schedule that satisfies these constraints is one where all level 0 nodes take turns in say a clockwise direction. For example, level 0 nodes transmit in a clockwise direction starting from $j.ne$, where j is a level 1 node. Each round thus consists of 17 slots.

Local storage

Each node i stores the most recent value of $M(x, y)$ received by i for each $tree(x, y)$ that i belongs to. The state of any node j is obtained as a function of $M(x', y')$ where x' is the smallest level that contains information about j. Recall that the resolution of the state of j decreases as the number of nodes in the aggregate $M(x', y')$ increases.

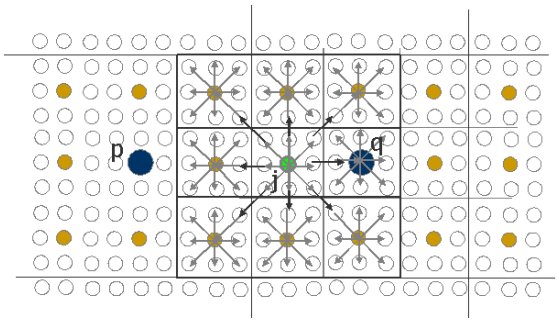


Figure 2.3: Illustrating level 1 tree rooted at j

Algorithm S1

We describe the actions executed by the nodes in three parts: (1) send / compute actions for nodes with $j.L > 0$, (2) send / compute actions for nodes with $j.L = 0$ and (3) receive / update actions for all nodes.

- In slot 0 of each round nodes with $j.L > 0$ compute the summary $M(r, j)$ for each level $1 \leq r \leq j.L$ that they are a clusterhead of based on the corresponding lower level information received in the previous round. The computed summary at each level is transmitted to the children on the respective tree rooted at j. Thus $M(r, j)$ is sent to $j.out(r, j)$ for $1 \leq r \leq j.L$. In addition, for each $tree(x, y)$ that $j.L$ belongs to but is not a root of, transmit $M(x, y)$ as heard in slot 0 from $j.in(x, y)$ to $j.out(x, y)$.

- To explain the actions of level 0 nodes, without loss of generality, consider level 1 nodes j and k and level 0 nodes $j.ne$ and $k.sw$ as shown in Fig. 2.2.

  - In slot 1, for each $tree(x, y)$ that $j.ne$ belongs to but is not a leaf of, transmit $M(x, y)$ as heard in slot 0 from $j.in(x, y)$ to $j.out(x, y)$. Also, transmit its own information $M(0, j.ne)$ to children in the level 0 tree rooted at $j.ne$.

  - In slot 9, for each $tree(x, y)$ that $j.ne$ belongs to but not a leaf of, transmit $M(x, y)$ as heard in slots 2 to 8 from $j.in(x, y)$ to $j.out(x, y)$.

- The action at any node j upon receiving a message from i is as follows: for each $tree(x, y)$ that j belongs to, store $M(x, y)$ if $i = j.in(x, y)$. □

In summary, aggregates computed at each level are copied only going downwards along a tree. This is sufficient for a level r node to compute aggregates from level $r-1$ nodes, because a tree at level $r-1$ extends up to all level 0 nodes in neighboring level $r-1$ clusters. And one of the neighboring level $r-1$ node is a level r node. Thus, when a computed aggregate by any node is being dispersed to nodes in its own cluster and the neighboring clusters, it is also being sent *in* to a higher level node to compute an aggregate. In Fig. 2.3, nodes p and q are level 2 clusterheads. Note that the level 1 tree rooted at j reaches the level 2 clusterhead q that j belongs to. Since a level r node is equidistant from all level $r-1$ nodes, the computed summaries are synchronous. We now analyze the latency and communication cost of algorithm $S2$.

Analysis

Lemma 2.3.1. *In $S1$, the maximum message length needed per slot is $(9 * log(n) - 7) * m$ bits.*

Proof. Consider a node j at any level r, where $0 \leq r \leq log(n)$. Let $1 \leq l \leq log(n)-1$. A level l tree rooted in the same level l cluster as that of j can pass through j. A level l tree rooted in neighboring level l clusters as that of j can also pass through j, but no other level l tree can pass through j. Thus, at most 9 trees at levels $1..log(n)-1$ can pass through each node. There is only one level $logn$ tree. Also j belongs to only one level 0 tree for which it is not a leaf. The maximum message length needed per slot in algorithm $S3$ is $(9 * log(n) - 7) * m$ bits. □

It follows that the slot width s_w needed in algorithm $S1$ is $\frac{(9*log(n)-7)*m}{W}$ bits per second.

Lemma 2.3.2. *In S1, the maximum staleness in the state of a node i received by a snapshot at node j is* $O(log(n) * m * d)$ *where* $d = dist(i, j)$.

Proof. Consider a node p at level r. To compute a summary at level r, level $r-1$ summaries are needed. $dist(p, q) = 3^{r-1}$, where q is any node in the set $p.nbr(r)$. Thus, a level r summary is computed based on a level $r-1$ summary that was generated $(17/3) * 3^{r-1} * s_w$ time ago, since latency between each pair of level 1 nodes is 17 slots. A level $r-1$ summary is computed based a level $r-2$ summary, and so on until level 0. Upon summation, we see that the staleness of a level 0 (individual node) state information in a level r summary is equal to $(17/2) * 3^{r-1} * s_w$. The maximum distance traveled by a level r summary is $(3/2) * 3^r$. The maximum latency involved is $(17/2) * 3^r * s_w$. The minimum distance between j and i for which a level r summary is the smallest level that contains information about j is 3^{r-1}.

The maximum staleness in the state of a node i at node j is given by the following equation:

$$
\begin{aligned}
staleness(i, j) &= (L1 + L2) && (2.6)\\
&= \frac{(L1 + L2)}{3^{r-1}} \times 3^{r-1} && (2.7)\\
&= O(34 * s_w * d) && (2.8)\\
&= O(log(n) * m * d) && (2.9)
\end{aligned}
$$

The result follows.

□

Lemma 2.3.3. *In S1, the resolution of state of a node i in a snapshot received at node j is* $\Omega(\frac{1}{d^2})$ *where* $d = dist(i, j)$.

Proof. In a level r summary, the state of 9^r nodes is compressed into m bits. We thus regard the error in the state of each node in that summary to be $O(9^r)$. The minimum distance between i and j at which j gets a level r summary of i but not a level $r-1$ summary of i is 3^{r-1}. Thus, the error in the state of i in a snapshot received at j is $O(d^2)$ and the resolution of state of i in a snapshot received at j is $\Omega(\frac{1}{d^2})$, where $d = dist(i,j)$. □

Lemma 2.3.4. *In S1, the average communication cost in the network to deliver a snapshot of one sample from each node to all nodes is $O(N * log(n) * m)$.*

Proof. To deliver a snapshot with a sample from each node, every node communicates $O(m*log(n))$ bits n times. And to deliver a snapshot with y samples from each node, every node communicates $O((n+y)*(m*log(n)))$ bits, since all the y samples are pipelined. Hence, if y is large and $y = \Omega(n)$, the average communication cost at each node to deliver a snapshot of a sample from each node to all nodes is $O(m*log(n))$. The average communication cost over N nodes is $O(N*(m*log(n))$.

Note that if y is small, for instance, if there is only one sample from each node, then the communication cost is $O(N*n*(m+log(n/m))$. Pipelining the delivery of snapshots improves the average communication cost to $O(N*(m*log(n))$. □

Lemma 2.3.5. *In S1, the memory requirement per node is $O(log(n)*m)$ bits.*

Proof. Recall that the data structure maintained at each node is the most recent value of $M(x,y)$ received by i for each $tree(x,y)$ that i belongs to. Nodes do not buffer information to be forwarded over multiple rounds. The maximum number of trees through any node is $O(log(n))$, with m bits of information flowing along each tree. The result follows. □

Extending to other dimensions

Consider an f dimensional structure. In this structure, nodes are divided into clusters with 3^f nodes per cluster. Thus there are $3^f - 1$ level 0 nodes per cluster. Each round consists of $2 * 3^f - 1$ slots. The number of slots per round increases proportional to 3^f.

Using these, we generalize Lemma 2.3.2 and Lemma 2.3.4 as follows:

Lemma 2.3.6. *In $S1$, the maximum staleness in the state of a node i received by a snapshot at node j in a network of f dimensions is $O(3^{2f} * log(n) * m * d)$ where $d = dist(i, j)$.* □

Lemma 2.3.7. *In $S1$, the average communication cost in the network to deliver a snapshot of one sample from each node to all nodes is $O(N * 3^f * log(n) * m)$.* □

Note also that in an f dimensional structure, the state of 3^{f*r} nodes is compressed in m bits. Following the structure of proof for Lemma 2.3.3, we get the following result.

Lemma 2.3.8. *In $S1$, the resolution of state of a node i in a snapshot received at node j in an f dimensional network is $\Omega(\frac{1}{d^f})$ where $d = dist(i, j)$.* □

2.3.3 Distance sensitive rate

In this subsection, we describe two algorithms in which nodes receive snapshots that are distance sensitive in latency, resolution and also distance sensitive in rate.

We partition the network hierarchically into clusters and schedule nodes to transmit in rounds exactly as we did in algorithm $S2$. The snapshot at each level is aggregated into m bits. However, instead of transmitting m bits for each level of data

in every round, we allocate the number of bits hierarchically. Accordingly, a message transmitted by a node in any given round consists of m bits for each level 0 information, $m/3$ bits for each level 1 information, and 1 bit for each level from $log(m)$ to $log(n)$. The actions executed by every node in their corresponding slots remain the same as in algorithm $S2$ except that every level r summary is now transmitted over $min(3^r, m)$ rounds, as opposed to in each round containing a new level r summary.

Algorithm $S3a$

By way of refining algorithm $S2$, consider a level 0 node with $j.L = r$. A level r summary is computed by this node once every 3^r rounds based on the most recent level $r-1$ summaries it receives. This summary $M(r, j)$, which consists of m bits, is transmitted in slot 0 of each round with $max(1, \frac{m}{3^r})$ bits per round. Thus, a level r summary is sent over $min(3^r, m)$ rounds. The actions for forwarding nodes remain the same except for the change that each node now only receives a fraction of $M(x, y)$ in every round for each $tree(x, y)$ that it belongs to, and it forwards only that fraction in the next round. We now analyze the latency and communication cost of algorithm $S3a$.

Analysis

Lemma 2.3.9. *In $S3a$, the maximum message length needed per slot in algorithm is $11 * \frac{m}{2} + 9 * log(\frac{n}{m})$ bits.*

Proof. Recall that at most 9 trees at levels $1..log(n) - 1$ can pass through each node, and there is only one level $logn$ tree. Also, j belongs to only one level 0 tree for which it is not a leaf. Moreover, $\frac{m}{3^r}$ bits are allocated for each level $0 \leq r \leq log(m)$ and 1

bit for each level z where $log(m) < z \leq log(n)$. The maximum message length (ML) needed is obtained by the following equation:

$$\begin{aligned} ML &= \frac{m}{3^0} + \sum_{i=1}^{i=log(m)} (9 * \frac{m}{3^i}) + 9 * (log(n/m)) && (2.10) \\ &= m + m * \frac{9}{2} * (1 - \frac{1}{m}) + 9 * log(\frac{n}{m}) && (2.11) \\ &< m + m * \frac{9}{2} + 9 * log(\frac{n}{m}) && (2.12) \end{aligned}$$

The result follows from these facts. □

It follows that the slot width s_w needed in algorithm $S3a$ is $\frac{m*\frac{11}{2}+9*log(\frac{n}{m})}{W}$ bits per second.

Lemma 2.3.10. *In $S3a$, the maximum staleness in the state of a node i received by a snapshot at node j is $O((m + log(n/m)) * d)$ where $d = dist(i, j)$.*

Proof. Consider a node p at level r where $r \leq log(m)$. To compute a summary at level r, level $r-1$ summaries are needed. $dist(p, q) = 3^{r-1}$ where q is any node in the set $p.nbr(r)$. The latency between each pair of level 1 nodes is 17 slots. Thus the latency to travel from level $r-1$ node to level r node is given by $(17/3) * 3^{r-1} * s_w$.

But in this algorithm, a level $r-1$ summary is actually transmitted in $3^(r-1)$ rounds by dividing it into 3^{r-1} parts. Thus, a level r summary is computed based on level $r-1$ summary that was generated $2 * (17/3) * 3^{r-1} * s_w$ time ago. A level $r-1$ summary is computed based a level $r-2$ summary and so on until level 0. Upon summation, we see that the staleness of a level 0 state information in a level r summary is $(17) * 3^{r-1} * s_w$.

Note that a complete level $r-1$ snapshot is sent every 3^{r-1} rounds in a pipelined manner. Thus every 3^{r-1} rounds, a level $r-1$ snapshot is received by a node. On

the other hand a level r snapshot is computed only every 3^r rounds. Thus a fresher level $r-1$ snapshot is always available to compute a new level r snapshot.

The maximum distance traveled by a level r summary is $(3/2) * 3^r$. However, this summary is sent in 3^r rounds. The maximum latency involved is $2 * (17/2) * 3^r * s_w$.

Recall that in order to update the local data structure of j, the state of a node i is updated using summary $M(x', y')$ where x' is the lowest level which contains information about k. Now the minimum distance between j and i for which a level r summary is the smallest level that contains information about j is 3^{r-1}.

The maximum staleness in the state of a node i at node j is given by the following equation:

$$\begin{aligned} staleness(i,j) &= (L1 + L2) && (2.13)\\ &= \frac{(L1 + L2)}{3^{r-1}} \times 3^{r-1} && (2.14)\\ &= O(68 * s_w * d) && (2.15)\\ &= O(m * d + log(n/m) * d) && (2.16) \end{aligned}$$

Note that at levels $r > log(m)$, 1 bit is allocated per round. Thus, for all these levels, a summary can be sent out in less than 3^r rounds. The maximum staleness for those nodes whose state is obtained using summaries greater than level r is less than that derived in the above equation.

The maximum staleness in the state of a node i received by a snapshot at node j is thus $O((m + log(n/m)) * d)$, where $d = dist(i, j)$. □

Lemma 2.3.11. *In S3a, the maximum interval between when a node j receives the state of node i is $O((m + log(n/m) * d)$, where $d = dist(i, j)$.*

Proof. Consider levels $0 \leq r \leq log(m)$. Note that a complete level r snapshot is sent every 3^r rounds in a pipelined manner. Thus every 3^r rounds, a level r snapshot is received by a node. The time corresponding to 3^r rounds is $(17/3) * 3^r * s_w$.

Recall that in order to update the local data structure of j, the state of a node i is updated using summary $M(x', y')$ where x' is the lowest level which contains information about k. Now the minimum distance between j and i for which a level r summary is the smallest level that contains information about j is 3^{r-1}.

The maximum interval between when a node j receives the state of node i is given by the following equation:

$$
\begin{aligned}
interval(i,j) &= \frac{(17/3) * 3^r * s_w}{3^{r-1}} \times 3^{r-1} && (2.17)\\
&= O(17 * s_w * d) && (2.18)\\
&= O(m * d + log(n/m) * d) && (2.19)
\end{aligned}
$$

Note that for levels $r > log(m)$, 1 bit is allocated per round. Thus, for all these levels, a summary can be sent out in less than 3^r rounds. The maximum interval between receiving two successive state information for those nodes whose state is obtained using summaries greater than level r is less than that derived in the above equation. The maximum interval between when a j receives the state of i is thus $O((m + log(n/m) * d)$. □

Lemma 2.3.12. *In S3a, the average communication cost to deliver a snapshot of one sample from each node to all nodes is* $O(N * (m + log(n/m))$. □

Proof. To deliver a snapshot corresponding to 1 sample from each node, every node communicates $O(m+log(n/m))$ bits n times. And to deliver a snapshot corresponding

to y samples from each node, every node communicates $O((n+y)*(m+log(n/m)))$ bits since all the y samples are pipelined. Hence if y is large and $y = \omega(n)$, the average communication cost at each node to deliver a snapshot of one sample from each node to all nodes is $O(m+log(n/m))$. The average communication cost over N nodes is $O(N*(m+log(n/m))$. □

Lemma 2.3.13. *In $S3a$, the memory requirement per node is $O(log(n)*m)$.* □

Proof. The maximum number of trees passing through any node is $O(log(n))$. The storage at each node is the most recent $M(x,y)$ for each $tree(x,y)$ that a node belongs to. In algorithm $S3a$, the difference is that this information $((M(x,y))$ arrives at a node over multiple rounds. The memory requirement is still $O(log(n)*m)$. We note that although information received by a node in a given round is buffered for certain number of rounds in schemes $S3a$, fresh information about a level is stored only after existing information about a level is disseminated. The maximum information that can be buffered by any node for a given level is still $log(n)*m$. Hence we have the result. □

Algorithm $S3a$ can be generalized to f dimensions just as algorithm $S2$ is. We summarize our results in Fig. 2.4, which shows the bounds on staleness and resolution in the state of i at nodes that are distance d away, the bound on interval at which state updates of i are received, and the average communication cost for delivering a snapshot of N samples (one from each node) across the network.

Discussion

We note that apart from decreasing the average communication cost per sample from each node, solution $S3a$ also offers flexibility in terms of the size of each sample.

Algorithm	Staleness	Communication cost	Resolution	Interval	Memory
$S2$	$O(3^{2f} * log(n) * m * d$	$O(3^f * N * m * log(n))$	$\Omega(\frac{1}{d})$	independent of d	$3^f * log(n) * m$
$S3a$	$O(3^{2f} * (m + log(n/m)) * d)$	$O(3^f * N * (m + log(n/m)))$	$\Omega(\frac{1}{d})$	$O(3^f * (m + log(n/m)) * d)$	$3^f * log(n) * m$

Figure 2.4: Summary of results for snapshot algorithms

When $m << log(n)$, the factor $log(n)$ dominates and the average communication cost is $O(N * log(n))$ and the staleness is bounded by $O(log(n) * d)$. In algorithm $S2$ the corresponding costs are $O(N * m * log(n))$ and $O(m * log(n) * d)$ respectively. But when the sample size increases to as large as order n, the average communication cost is $O(N * n)$ and the staleness is bounded by $O(n * d)$, where as in algorithm $S2$ the corresponding costs are $O(N * n * log(n))$ and $O(n * log(n) * d)$ respectively.

2.3.4 Irregular Networks

In this subsection, we relax several assumptions we have made in designing algorithms $S1 - S2$. We show how the algorithms continue to yield distance sensitive snapshots and quantify the impact on the performance in the following cases: non uniform density, holes of arbitrary sizes within the connected network, non unit disk radios and imperfect clustering.

Clustering Model

We assume the existence of a clustering layer that partitions the general but connected network, as modeled in Section 2, into hierarchical clusters such that every network node belongs to one cluster at each level. As perfect (i.e., regular and symmetric) clustering may no longer be possible, we may not assume, for example, that all level 0 nodes are within 1 hop range of the level 1 clusterhead. We weaken that assumption to: each level 1 cluster includes all nodes that are 1 hop away but may

also include nodes that are up to some bounded number of hops, z, from it. Likewise, all higher level clusterheads also have the same radius range as opposed to a uniform radius. Moreover, the paths between neighboring clusters also need not the shortest ones, unlike what we assumed in the previous sections.

More formally, our clustering assumption is stated as follows (with distance standing for communication *hop* distances):

- $(C1)$ All nodes within distance $\frac{3^k-1}{2}$ from a level k clusterhead belong to that cluster.
- $(C2)$ The maximum distance of a node from its level k clusterhead is $z^k \times \frac{3^k-1}{2}$.
- $(C3)$ There exists a path from each clusterhead to all nodes in that cluster containing only nodes belonging to that cluster.
- $(C4)$ At all levels $k > 0$, there is at least one and at most 8 neighboring level k clusters for each level k clusterhead and there exists a path between any two neighboring clusterheads.

We note that the existence of such clustering solutions has been validated in previous research [57] and also been used in the context of object tracking.

Networks with non uniform density

Once the network has been partitioned into clusters, we impose a virtual grid on the network, as shown in Fig. 2.5. Each level 0 node belongs to some cell, but now each cell in the virtual grid may contain any number of nodes. In particular, cells may be empty and empty cells may be contiguous; we call sets of contiguous empty cells the *holes* of the network. We first describe how the case of over density is handled.

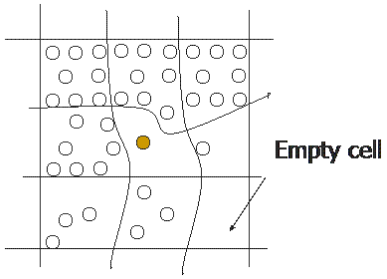


Figure 2.5: Virtual grid and cells with different densities

Over density cells: In the virtual grid, each cell gets a slot to transmit as described in algorithms $S0 - S1$. When a cell has more than one node, each node in the cell gets a turn over multiple rounds to send its data, resulting in time sharing between nodes of a cell to transmit its own data. However, once data is sent out from the source, the forwarding of the data does not incur this extra delay despite going through denser cells. This is because any node in the dense cell that gets a turn in a given round can forward the data heard in the previous round from neighboring cells.

Under density cells, holes, and imperfect clustering: If a particular cell is empty in sparse regions of the network, then the communication path between neighboring level k clusters ($0 < k < log(n)$) is lost.

Scheduling scheme (FS): Recall that a *round* is a unit of time in which information is exchanged between a level 1 clusterhead and all its neighboring level 1 clusterheads. In the general model, a level 1 cluster can cover up to a z hop neighborhood. Accordingly, the basic round scheduling introduced in Section 2 is adapted as follows. For ease of explanation, assume $z = 2$. Thus, a level 1 cluster comprises a minimum of the 1 hop neighborhood and a maximum of the 2 hop neighborhood. At the beginning of each round, level 1 clusterheads transmit. There can be at most

26 level 0 nodes in each cluster. Some may be absent because of holes or because of non uniform clustering. Depending on the position in the cluster, each level 0 node gets 2 slots per round such that information is exchanged from all 8 directions. The round length thus increases with z.

Distance sensitivity: A level k tree rooted at any node j spans all nodes in its own level k cluster and all nodes in the neighboring level k clusters, using fixed paths as described in Sections 3 and 4. If a neighboring level k cluster is completely absent, then there is no need to reroute the information. However if the neighboring level k cluster is present but the path is broken, then information has to be re-routed. We now investigate the impact on latency and resolution when the information takes a different path than normal.

Recalling the clustering specifications stated above, consider any two nodes i and j in the network. Let the shortest path between these two nodes in the presence of holes be p.

Lemma 2.3.14. *If k is the smallest level at which i and j are neighbors then $p > 3^{k-1}$.*

Proof. Note that i and j are not neighbors at level $k-1$. And if $p \leq 3^{k-1}$, then a level $k-1$ cluster cannot exist between i and j since from property $C1$, a level $k-1$ cluster has a minimum radius of $\frac{3^{k-1}-1}{2}$. □

Theorem 2.3.15. *Under CM, lgorithms $S1$, $S2$ yield snapshots that retain their distance sensitive properties (Since network is not regular, the distance sensitivity is strictly* hop *distance sensitivity and does not translate to geometric distance).*

Proof. From the previous lemma, the minimum distance between two nodes i and j for which level k is the smallest level at which i and j are neighbors is 3^{k-1}. Using $C3$ and $C4$ we have that level k information can be exchanged between i and j.

Despite the fact the trees are not formed along the regular grid pattern, it still holds that not more than 9 trees per level pass through any node. This is because there at most 8 neighboring level k clusters for any level k cluster. Moreover, the maximum degree of any node in all trees is still 8, by imposing the virtual grid for level 0. Therefore, the slot width allocations in algorithms $S1$, $S2$, $S3a$ and $S3b$ are sufficient to transmit all information despite the trees not being created in a regular pattern. □

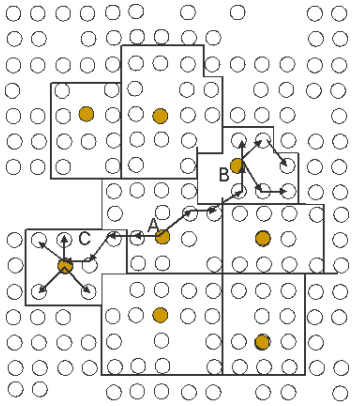


Figure 2.6: Handling holes in dense networks

Rerouting in Case 1 is illustrated in Fig. 2.6. The figure shows a level 1 cluster with a clusterhead A that has 7 neighboring level 1 clusters. The small unfilled circles represent cells of the virtual grid; these may contain one or more level 0 nodes. The level 1 clusters cover up to a 2 hop neighborhood. The figure also shows a level 1 tree rooted at A and extending up to clusters B and C.

Non uniform radio range

The design of algorithms $S0 - S3$ assumed a strictly 1 hop communication range. If communication range were relaxed to radio interference range varying from 1 to s hops, the basic scheduling for each round would need to take into account this additional interference. This would result in longer round lengths proportional to the size of interference region.

Implementation considerations

In this subsection, we highlight considerations for implementing our snapshot services in wireless sensor networks. In the past, we have implemented *Trail* [46], an asynchronous network protocol for querying object states and used this in the context of a distributed object tracking application. We have noted that as the objects in the network get densely located, interference issues lead to high latency and decreased efficiency. The snapshot services that we consider in this paper are high density operations and TDMA is naturally suited for such scenarios as interference can be avoided. But we do not need global time synchronization in the network. Nodes in their network can learn their TDMA slots by knowing their relative position to that of a clusterhead and locally scheduling in a non interference manner. Another issue to consider is that of localization. For our snapshot services, knowledge of relative location with respect to clusterheads is sufficient as opposed to the knowledge of precise coordinates.

2.4 Using the snapshot service for distributed object tracking

In the previous section, we described a snapshot service for wireless sensor networks that provided distance sensitive latency, resolution and rate. In this section, we describe how this service can be used for distributed tracking. Specifically, we describe how the location information for objects can be compressed such that we have an error that grows only linearly with distance as is required by the tracking application.

Let the number of bit m allocated for information at each level be equal to the number of evaders in the network. Recall that the identifiers of the evaders are known. Every clusterhead at level i sets the bit e in this information if object e is within its cluster. Thus the highest resolution at which the locatio of an evader is available is the size of the level 0 cluster. We relax our condition for *catch* to be such that the pursuer and the evader are in the same level 0 cluster.

Note that if the smallest level at which information about an evader e is available to pursuer p is k, then it must be the case that d_{pe} is at least 3^k. Thus, the maximum error in the estimation of distance to e is bounded by the radius of the level k cluster, i.e. $\frac{3^k}{2}$. Thus, we get that the maximum error is bounded by $\frac{d_{pe}}{2}$ when distance between p and e is d_{pe}. Thus we obtain an error that grows at most linearly with distance between p and e and is always less than the distance between p and e and satisfying conditions for optimal pursuit. We have already shown that latency grows as $O(log(n) * d)$ even when information is updated in every round. Thus using the snapshot service, information about e is available at a faster rate than required.

Thus we see that our distance sensitive snapshot service satisfies the conditions required for eventual pursuit in the distributed pursuer evader tracking application.

2.5 Related work

2.5.1 Pursuer evader games using sensor network

In previous work, Schenato et al studied a pursuit evasion game application using sensor networks. They considered a detailed system model with periodic time updates and presented models of vehicle dynamics and uncertainty in track information. Sensor network measurements are assumed to be fused at local stations to produce track information. Evader assignment and pursuer control strategy is calculated at the base station and then communicated to the pursuer agents. Network effects in communicating this information to the pursuer agents and communicating pursuer locations back to the base station are not considered. Within this framework, they derived a series of algorithms to coordinate the pursuers so as to minimize the time-to-capture of all evaders.

In this chapter, we have concentrated on the formulation of eventual pursuit control strategies despite network effects. We assume target track information has been established through local fusion of sensor data. This track information is communicated through the multi-hop wireless network infrastructure to pursuer agent, which calculates a pursuit strategy based on evader and its own state.

2.5.2 Snapshot services

Communicating periodic global state snapshots is a well studied problem in distributed systems [17] and consistency, timeliness and reliability have been the main

design considerations in those studies. But efficiency becomes essential when considering periodic snapshots for resource constrained wireless sensor networks. To the best of our knowledge algorithms for delivering periodic snapshots across a wireless sensor network have not been studied before.

A common approach to achieving compression for efficiency is to exploit the temporal and spatial correlation of data being shared. For example, in [66], the authors propose a framework for a one time all-to-all broadcast of sensor data assuming the data is spatially correlated. There has also been work [20] on compression mechanisms for correlated sensor data sent to a central base station. Instead, in this paper we do not require data to be correlated. At the same time, our algorithms can be used in conjunction with other forms of compression.

Fractionally cascaded information [22] is a form of distance sensitive resolution that is widely used in computational geometry community for speeding up data structures. Each node stores an ordered list of keys, shares a well distributed sample of that list with its neighbors, a sample of that sample with its two hop neighbors, and so on. Recently, fractional cascading has been used for sensor networks as an efficient storage mechanism [28, 65]. Data is first stored at multiple resolutions across the network, which is then used to efficiently answer aggregate queries about a range of locations without exploring the entire area. In contrast, we have considered a model where information is generated and consumed on an ongoing basis. Accordingly we describe push based services that regularly deliver to subscribers snapshots of the network in a pipelined manner. By providing snapshots with not just distance sensitive -resolution but also -latency and -rate, we achieve compression and thereby efficiency.

At the same time these services can be used in range based querying as well as in several other control applications.

An algorithm for creating the multi-resolution data structure based on probabilistic gossip mechanism has been discussed in [65]. In [65], the algorithm described is for a one shot dispersion and proceeds in stages while our services are for a model where information is consumed on an ongoing basis and accordingly we describe a pipelined implementation that is based on scheduling. In [65], the aggregation operations are duplicate insensitive and global time synchronization is assumed while we do not require either of these properties. Our communication costs and latency are lower than those in [65] and we also describe services that additionally have distance sensitive rate properties. But we note that while we assume hierarchical clustering in our solutions, the algorithm in [65] does not.

The idea of *distance sensitive rate* has also arisen in other contexts. Fisheye state routing is a proactive routing protocol [62] that reduces the frequency of topology updates to distant parts of the network. The spatial gossip protocol [40] is one in which each node in a peer-to-peer network chooses to communicate to another node with a probability that decreases polynomially with the distance between the pair.

Recently algorithms for bulk data collection in sensor networks have been proposed. In [41] data is collected from one node at a time, while [58] performs concurrent, pipelined ex-filtration of data using TDMA schedules. In [49], the authors describe TDMA based algorithms optimized for convergecast. Our algorithms can be specialized for the case of bulk convergecast and we additionally emphasize on efficiency using distance sensitive properties.

2.6 Summary

In this chapter, we considered a pursuer evader tracking scenario where multiple pursuers and evaders exist in a bounded region. We presented a pursuer control strategy and associated network support that guarantees an *eventual* catch despite network effects. We showed that error, latency and update periods that decrease linearly with distance are sufficient to achieve eventual catch in pursuer tracking application.

We then designed wireless sensor network algorithms that periodically deliver snapshots to all nodes in the network, which satisfy the above properties. Our algorithms are easily adapted to allow snapshots to be delivered only to a subset of nodes as opposed to all nodes. They are memory efficient, requiring only $O(3^f * log(N^{\frac{1}{f}}) * m)$ bits per node. They are readily realized in networks with irregular density, networks with arbitrary sized holes, imperfect clustering, and non unit disk radios. We quantify the maximum rate at which information can be generated at each node so that snapshots are periodically delivered across the network, the algorithms can of course be operated at lower rates than these. For our services, global time synchronization is not required; a local notion of time however is needed to ensure fair scheduling of transmission of nodes.

We have shown how our snapshot service can be used for the distributed pursuer evader tracking application described above. In future, we would like to explore the possibility of using this snapshot service in the context of other applications such as control of distributed parameter systems and process control systems, in which multiple distributed controllers require the state of the system periodically. We also expect to implement our snapshot algorithms in the context of applications such

as pursuer evader tracking and vibration control, and study their performance and tradeoffs more exhaustively in the future.

CHAPTER 3

DISTRIBUTED PURSUER EVADER TRACKING APPLICATION WITH OPTIMAL CATCH

In this chapter, we extend the pursuer evader tracking application to additionally require that the evaders be caught in an optimal manner. Specifically, we consider a pursuit-evasion game called "asset protection game" where pursuers try to protect a linear target by intercepting the evaders as far as possible from the target. This game structure has practical applications in real world applications of border and pipeline protection and the techniques used to design this application can be generalized to a wide variety of pursuit-evader games.

We assume target track information has been established through local fusion of sensor data. This track information is communicated through the multi-hop wireless network infrastructure to pursuer agent, which calculates optimal pursuit strategy based on evader and its own state. But as the network is scaled in size, high throughput rates for all pursuers cannot be sustained at all times, which necessitates a network communication strategy that adapts to pursuer information requirements.

In this chapter, we first concentrate on the formulation of optimal pursuit control strategies despite network effects. We adapt ideas from theory of differential games to networked games in the presence of non-periodic track updates, message loss and

delays to derive optimal strategies, bounds on their information requirements and the scaling properties of these bounds. In summary, we show (i) pursuer agents should dictate the information refresh rate based on the requirements of the pursuit strategy, and (ii) network delays and update periods should scale linearly with the pursuer-evader distance to guarantee the existence of optimal min-max pursuit strategies leading to Nash equilibrium.

We then describe an energy-efficient, fault-tolerant and hierarchy-free WSN service, *Trail*, that supports the optimal pursuit requirements for the asset protection game. The specification of *Trail* is to return the location of a particular object in response to an in-network subscriber issuing a *find* query regarding that object. *Trail* has a *find* cost that is linear ($O(d_f)$) in terms of the distance (d_f) of the subscriber from the object. To this end, *Trail* maintains a tracking data structure by propagating mobile object information only locally, and satisfying the distance sensitivity requirement for the track updates. The amortized cost of updating a track when an object moves a distance d_m is $O(d_m * log(d_m))$.

The rest of this chapter is organized as follows. First, in Section 3.1 we introduce the game model. In Section 3.2, we review the optimal min-max strategies for the pursuer and the evader under network communication constraints, and state lower bounds on network performance requirements. Next, in Section 3.3, we present a network service *Trail* that meets the required scalable information characteristics. In Section 3.4, we describe the implementation of *Trail* on a 105 node Mica2 network, that is used to support the asset protection game described above and use this implementation to also study the performance of *Trail*. In Section 3.5 we discuss work related to *Trail*. In Section 3.6, we present a summary.

3.1 Game Model

We consider a game between two players: a single pursuer and a single evader. (For many n pursuer – n evader games the min-max solution can be reduced to n two player games, by first solving the combinatorial problem of optimal pairing using the value function of the two player game). The game state is given by the two dimensional coordinates of the pursuer and evader $x = \{x_p, y_p, x_e, y_e\}$. Each player travels at constant speed v_p and v_e and controls the direction of its motion, denoted by θ_p and θ_e.

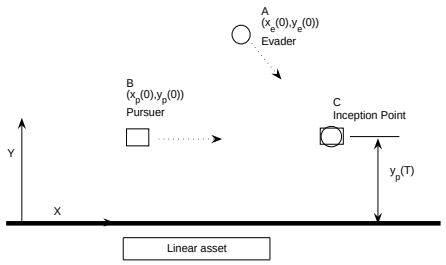


Figure 3.1: The pursuer and evader game

We assume that there are no obstacles in the environment to constrain the movement of the players. The players employ feedback strategies $(u_p(x(t)), u_e(x(t)))$ which determine their direction of motion given the current state. The linear asset is assumed to be infinitely long. With this assumption, the state space can be reduced to three dimensions by defining relative coordinates, $x_r = x_e - x_p$ and $y_r = y_e - y_p$. The

state vector x evolves according to:

$$\dot{x} = \frac{\partial}{\partial t}\begin{bmatrix} x_r \\ y_r \\ y_p \end{bmatrix} = f(x, \theta_p, \theta_e) = \begin{bmatrix} v_e\cos(\theta_e) - v_p\cos(\theta_p) \\ v_e\sin(\theta_e) - v_p\sin(\theta_p) \\ v_p\sin(\theta_p) \end{bmatrix}$$

A catch is said to happen when $x_r^2 + y_r^2 < r^2$, where r is the catch radius. In the following, we consider the limiting case of $r \to 0$.

Starting from the initial condition x_0, if the control strategies $u_p(x), u_e(x)$ satisfy the catch condition at time T then the payoff is given by $\mathcal{J}(u_p, u_e, x_0) = y_p(T)$. The game is zero-sum, so the pursuer's goal is to maximize $\mathcal{J}$ whereas the evader's goal is to minimize $\mathcal{J}$.

Min-max optimal feedback strategies $u_p^*(x), u_e^*(x)$ are defined by the saddle condition:

$$\mathcal{J}_{u_p}(u_p, u_e^*, x_0) \leq \mathcal{J}(u_p^*, u_e^*, x_0) \leq \mathcal{J}_{u_e}(u_p^*, u_e, x_0) \tag{3.1}$$

We also note that the min-max optimal strategy pair $u_p^*(x), u_e^*(x)$ is also the Nash equilibrium [59] for this zero-sum game, where none of the players have an incentive to change its strategy unilaterally given the rival is maintaining its strategy choice.

3.2 Conditions for optimal interception

In this section, we first state the optimal min-max strategy when information about evader is received continuously and instantaneously by the pursuer. We then characterize the conditions on the maximum rate at which this information can be received and also the maximum latency that be involved, in order to maintain the optimality of pursuit strategy.

3.2.1 Optimal pursuit under perfect information

Given the current location of the evader and pursuer, the set of points that the evader can reach before the pursuer is given by the well known Appolonius circle. The min-max optimal strategies for the pursuer and evader is to directly to the boundary point of the circle that is closest to the target. We state this formally in the following theorem, the proof of which can be found in [15].

Theorem 3.2.1. *If the ratio of the pursuer speed v_p to the the evader speed v_e α is larger than 1, then the min-max optimal strategy for the evader and pursuers is given by:*

$$\theta_e(x) = \tan^{-1}(\tan\gamma + \alpha\sqrt{1+(\tan\gamma)^2}) \tag{3.2}$$

$$\theta_p(x) = \tan^{-1}(\tan\gamma + \frac{\sqrt{1+(\tan\gamma)^2}}{\alpha}) \tag{3.3}$$

where $\gamma = \tan^{-1}(\frac{y_r}{x_r})$.

$$V(x) = y_p + \frac{\alpha^2 y_r + \alpha\sqrt{y_r^2 + x_r^2}}{\alpha^2 - 1} \tag{3.4}$$

At each time instant t, the pursuer will calculate the best location $(x', y'))$ that the evader can reach:

$$x' = \frac{x_r\alpha^2}{\alpha^2 - 1} + x_p \tag{3.5}$$

$$y' = \frac{\alpha^2 y_r + \alpha\sqrt{y_r^2 + x_r^2}}{\alpha^2 - 1} + y_p \tag{3.6}$$

then it will move towards that location.

We illustrate the performance of the optimal strategy using a simulation. The results are given in Figure 3.2. The solid line shows the pursuer-evader trajectories

when both employ min-max optimal strategies. The dashed lines show the case when evader uses non-optimal straight line strategies. We observe that min-max optimal pursuit strategy catches non-optimal evaders at a larger distance to the target.

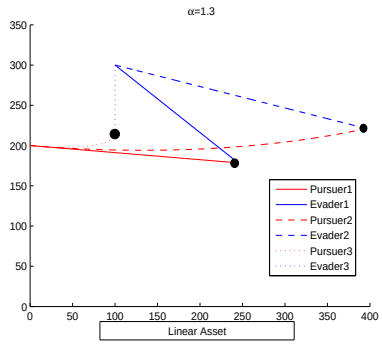


Figure 3.2: The P-E trajectory under perfect information

3.2.2 Sampling rate requirements of the optimal pursuit strategy

In the previous subsection, we assumed that the global state is available to the pursuer at all times. This is an unrealistic assumption for a sensor network implementation where the information can be provided only at discrete time intervals. In this section, we characterize the sampling rate requirements of the optimal strategy and show that it is inversely proportional to the relative distance between the pursuer and evader. we use the min-max solution concept to formulate a robust pursuit strategy that will perform satisfactorily irrespective of evader motion. To design for worst possible case of evader motion, we assume the pursuer has perfect information about the location of the evader and the sampling period. The sampling period is then chosen such that the evader does not benefit from switching from the optimal

direction given in Theorem. 3.2.1, although the evader's deviation will be detected by the pursuer after the sampling period interval.

Theorem 3.2.2. *The evader does not deviate from its min-max equilibrium strategy if and only if the distance moved by the pursuer before getting the next sample of state information satisfies:*

$$v_p T_{sample} < \frac{\sqrt{\alpha^2(x_r)^2 + (\alpha(y_r) + \sqrt{(x_r)^2 + (y_r)^2})^2}}{\alpha} \tag{3.7}$$

Equivalently, the pursuer can move up to $\frac{(\alpha^2-1)}{\alpha^2}$ *of the total distance to the predicted evader location before sampling the global state without loss of optimality.*

The proof of this result can be found in [15]. The result is particularly important for sensor network implementations using resource constrained nodes, because it informs how the information data rate can be reduced based on the state of the game so as to conserve the energy and bandwidth resources of the network.

We extend the previous result to derive the following scaling property of the sampling period T_{sample} with respect to the distance d_{pe} between the pursuer and evader:

Theorem 3.2.3. *Optimal pursuit-evasion strategies of the perfect information game also yield Nash equilibrium of the game with discrete time updates if:*

$$T_{samp}(d_{pe}) \leq \frac{\alpha - 1}{\alpha v_p} d_{pe}$$

In other words, the sampling period should decrease proportionally with decreasing distance between evader and pursuer to guarantee that the evader does not have an incentive to deviate from its strategy to move directly to the predicted intercept point.

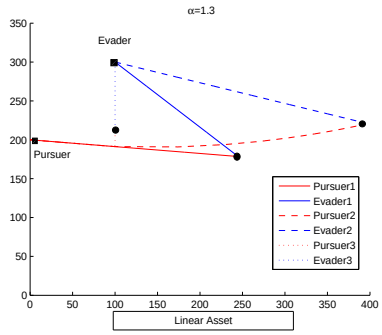


Figure 3.3: The P-E trajectory when using the T_{samp} update

We illustrate the performance of the reduced sample rate strategy using a simulation. The results are given in Figure 3.3. The solid line shows the pursuer-evader trajectories when both employ min-max optimal strategies, which is identical to the continuous update case. The dashed lines show the case when evader uses non-optimal straight line strategies. We observe that reduced sample rate pursuit strategy differs from its continuous information behavior for these cases but still catches these non-optimal evaders at a larger distance to the target.

3.2.3 Effect of Packet Delay

The evader location information needs to be routed from the local fusion center to the pursuer through wireless multiple hop links. The multiple hop communication imposes considerable delays on the evader state information. We assume the network is time synchronized and the packets are time-stamped at the source so that the pursuer will able to calculate the delay of the packets it received. To derive a robust pursuit strategy we design for the worst possible evader motion, by assuming the evader will have perfect information about the pursuer location. Therefore at time

increment t evader have access to state information $[x_p(t), y_p(t), x_e(t), y_e(t)]$ and the pursuer have access to state information $[x_p(t), y_p(t), x_e(t - \Delta t), y_e(t - \Delta t)]$. Then consider the following strategies:

Evader Strategy $\tilde{u}_e$: The evader uses the current location information for the pursuer to calculate the optimal direction as given in Theorem 3.2.1.

Pursuer Strategy $\tilde{u}_p$: The pursuer estimates the worst case location $(\hat{x}_e(t), \hat{y}_e(t))$ of the evader by considering all the points that the evader can reach at Δt and choosing the one that yields the lowest game value $V(\hat{x}_p(t), \hat{y}_p(t), x_e(t), y_e(t))$

Theorem 3.2.4. *The strategies $\tilde{u}_p$ and $\tilde{u}_e$ are a Nash equilibrium of the pursuer-evader game with packet delays if the delay at each point is bounded by:*

$$\Delta t < \frac{\alpha - 1}{\alpha v_p} d_{pe}(t - \Delta t)$$

where $dpe(t - \Delta t)$ is the pursuer-evader distance at the time of packet transmission.

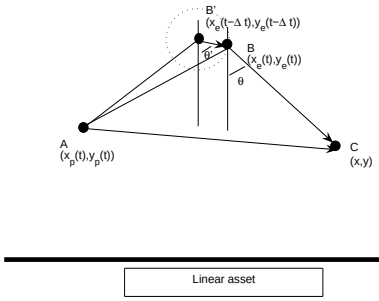


Figure 3.4: Effect of packet delay

In other words, the delay should decrease proportionally with decreasing distance between evader and pursuer to guarantee that the evader does not have an incentive to deviate from its strategy to move directly to the predicted intercept point.

3.3 Trail: Network service for distributed tracking

The results of the previous section indicate the following requirements on the network protocol responsible for communicating evader track information to the pursuer agents: (i) Pursuer should determine the information refresh rate based on the requirements of the pursuit strategy, and (ii) Network delays should scale with the pursuer-evader distance. In this section, we describe a middleware service called *Trail* that is compatible with these requirements.

The specification of *Trail* is to return the location of a particular object in response to an in-network subscriber issuing a *find* query regarding that object. *Trail* has a *find* cost that is linear ($O(d_f)$) in terms of the distance (d_f) of the subscriber from the object. To this end, *Trail* maintains a tracking data structure by propagating mobile object information only locally, and satisfying the distance sensitivity requirement for the track updates. The amortized cost of updating a track when an object moves a distance d_m is $O(d_m * log(d_m))$.

A basic *Trail* protocol can be refined by tuning certain parameters, thus resulting in a family of *Trail* protocols. Appropriate refinements of the basic *Trail* protocol are well suited for different network sizes and *find*/*update* frequency settings: One refinement is to tighten its tracks by progressively increasing the rate at which the tracking structure is updated; while this results in updating a large part of the tracking structure per unit move, which is for large networks still *update* distance sensitive, it

significantly lowers the *find* costs for objects at larger distances. Another refinement increases the number of points along a track, i.e., progressively loosens the tracking structure in order to decrease the *find* costs and be more *find-centric* when object *updates* are less frequent or objects are static. Moreover, *Trail* scales well to networks of all sizes. As network size decreases, *Trail* gradually eschews local explorations and updates and thus increasingly centralizes *update* and *find*.

We evaluate the performance of *Trail* by simulations in a 90×90 sensor network and experiments on 105 Mica2 nodes in the Kansei testbed[3]. This implementation has been used to support a distributed intruder interceptor tracking application where the goal of the interceptor is to catch the intruders as far away from an asset as possible.

Overview of solution: *Trail* maintains tracks from each object to only one terminating point, namely, the center of the network C; these tracks are *almost* straight to the center, with a stretch factor of at most 1.2 times the distance to C. Note that if the track to an object P is required to be always a straight line from C to the current location of P resulting in a stretch factor equal to 1, then every time the object moves, the track has to be updated starting from C, which would not be a distance sensitive approach. Therefore, we form the track as a set of *trail segments* and update only a portion of the structure depending upon the distance moved. Thus given a terminating point *Trail* maintains a track with lengths from the terminating point that is almost close to minimum. This is important because longer tracks have a higher cost of initialization and given that network nodes may fail due to energy depletion or hardware faults, longer tracks increase the probability of a failed node along a track as well as increase the cost of detecting and correcting failures in the track.

Given the track to an object, a *find* operation explores along circles of exponentially increasing radii until the track is intersected and then follows the track to the current location of the object. The tracks maintained in *Trail* only contain pointers to the current location of an object and not the state information of the object. Publishing the current state (namely location) of the object along all points in track will violate distance sensitivity of updates because every *move* of the object will result in updating the entire track. Following the *trail* of an object from any location leads to the current location of the object which contains the state of the object. Yet *Trail* is distance sensitive in terms of the *find* in the sense that the total cost of reaching the track for an object, and following the track to reach the object is proportional to the distance of the finder from the object. Note that a *find* explores along circles until a radii that is at most half the distance of the finder to C and then searches at C where the track is certain to be found. Thus C serves as a worst case landmark for finding objects in the network.

In our solution, we make some design decisions like choosing a single point to terminate tracks from all points in the network and avoiding hierarchy in maintaining the tracks. In Subsection 3.3.6, we analyze these aspects of our solution and compare them with other possible approaches.

In Subsection 3.3.1, we formally state the system assumptions for *Trail* and the specification. In Subsection 3.3.2, we design the basic *Trail* protocol for a 2-d real plane. Then, in Subsection 3.3.3, we present an implementation of the basic *Trail* protocol for a 2-d sensor network. In Subsection 3.3.4, we discuss refinements of the basic *Trail* protocol. In Subsection 3.3.6, we analyze some design decisions made in

our solution and compare them with other possible approaches. In Subsection 3.3.7, we present results of our performance evaluation in simulation.

3.3.1 System Model and Specification

The system consists of a set of *mobile objects*, and a network of static *nodes* that each consist of a sensing component and a radio component. Tracking applications execute on the mobile objects and use the sensor network to track desired mobile objects. Object detection and association services execute on the nodes, as does the desired *Trail* network tracking service.

The object detection and association service assigns a unique id, P, to every object detected by nodes in the network and stores the state of P at the node j that is closest to the object P. This node is called the *agent* for P and can be regarded as the node where P resides. The problem of detecting objects in the network and uniquely associating them with previous detections is thus orthogonal to the tracking service that we discuss in this paper. Detection and association services can be implemented in a centralized [69] or distributed [67] fashion; the latter approach would suit integration with the tracking service that we discuss in this paper. Detection and association could also be enabled by the objects being *cooperative*, for example, being tagged with RFID and announcing their identifier. We assume that the underlying detections and associations are always correct.

Trail Network Service: *Trail* maintains an in-network tracking structure, $trail_P$, for every object P. *Trail* supports two functions: *find(P, Q)*, that returns the state of the object P, including its location at the current location of the object Q issuing the

query and *move(P, p', p)* that updates the tracking structure when object P moves from location p' to location p.

Definition 10 (*find*(P, Q) Cost)**.** The cost of the *find*(P, Q) function is the total communication cost of reaching the current location of P starting from the current location of Q.

Definition 11 (*move(P, p', p)* Cost)**.** The cost of the $move(P, p', p)$ function is the total communication cost of updating $trail_P$ to the new location p and deleting the tracking structure to the old location p'.

We note that our network service does not assume knowledge of the motion model of objects being tracked, in contrast to some query services [53], and as such the scope of every query in our case is the entire network as opposed to a certain locality. Nor does it assume a bound on the number of querying objects in the network or any synchrony between concurrent queries.

Network Model: To simplify our presentation, we first describe *Trail* in a 2-d real continuous plane. We then refine the *Trail* protocol to suitably implement in a random connected deployment of a wireless sensor network. In this model, we impose a virtual grid on the random deployment and snap each node to its nearest grid location (x, y). Each node is aware of this location. We refer to unit distance as the one hop communication distance. $dist(i, j)$ now stands for distance between nodes i and j in these units. We describe this model in more detail and the implementation of *Trail* in this discrete model in Section 4.

Fault Model: In the wireless sensor network, we assume that nodes can fail due to energy depletion or hardware faults or there could be insufficient density at certain

regions, thus leading to *holes* in the network. However, we assume that the network may not be partitioned; there exists a path between every pair of nodes in the network.

3.3.2 Trail on a 2-d Plane

In this section, we use geometric ideas to design *Trail* for a bounded 2-d real plane. Let C denote the center of this bounded plane.

Tracking Data Structure

We maintain a tracking data structure for each object in the plane. Let P be an object being tracked, and p denote its location on the plane. We denote the tracking data structure for object P as $trail_P$. Before we formally define this tracking structure, we give a brief overview.

Overview: If $trail_P$ is defined as a straight line from C to P, then every time the object moves, $trail_P$ has to be updated starting from C. This would not be a distance sensitive approach. Hence we form $trail_P$ as a set of *trail segments* and update only a portion of the structure depending upon the distance moved. The number of *trail segments* in $trail_P$ increases as $dist(p, C)$ increases. The end points of each *trail segment* serve as marker points to update the tracking structure when an object moves. The point from where the update is started depends on the distance moved. Only, when P moves a sufficiently large distance, $trail_P$ is updated all the way from C. We now formally define $trail_P$.

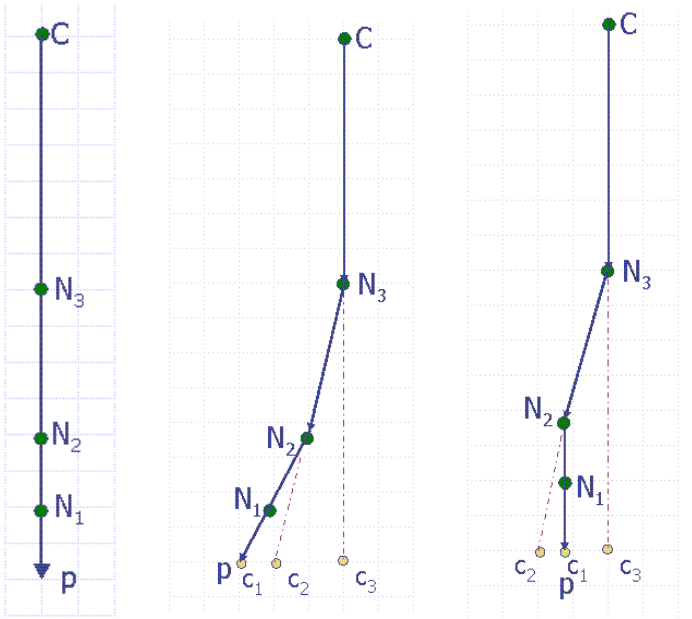
Definition 12 ($trail_P$)**.** The tracking data structure for object P, denoted by $trail_P$, for $dist(p, C) \geq 1$ is a path obtained by connecting any sequence of points $(C, N_{mx}, ..., N_k, ..., N_1, p)$ by line segments, where $mx \geq 1$, and there exist auxiliary points $c_1..c_{mx}$ that satisfy

the properties $(P1)$ to $(P3)$ below. mx is defined as $\lceil(log_2(dist(C, p_o)))\rceil - 1$, where p_o is the location of p when $trail_P$ was initialized or updated starting from C.

For brevity, let N_k be the level k vertex in $trail_P$; let the level k *trail segment* in $trail_P$ be the segment between N_k and N_{k-1} ; let $Seg(x, y)$ be any line segment between points x and y in the plane.

- (P1): $dist(c_k, N_k) = 2^k$, $(mx \geq k \geq 1)$.
- (P2): N_{k-1}, $(mx \geq k \geq 1)$, lies on $Seg(N_k, c_{k-1})$; N_{mx} lies on $Seg(C, c_{mx})$.
- (P3): $dist(p, c_k) < 2^{k-b}$, $(mx \geq k \geq 1)$ and $b \geq 1$ is a constant.

If $(dist(p, C) = 0)$, $trail_P$ is C; and if $(0 \leq dist(p, C) < 1)$, $trail_P$ is $Seg(C, p)$. □



(a) Initial (b) P moves 3 units from c_3 (c) P moves back towards c_3

Figure 3.5: Examples of Trail to an Object P

Observations about $trail_P$: From the definition of $trail_P$, we note that the auxiliary points $c_1..c_{mx}$ are used to mark vertices $N_1..N_{mx}$ of $trail_P$. $P1$ and $P2$ describe the relation between the auxiliary points and the vertices of $trail_P$. Given $trail_P$, points $c_1..c_{mx}$ are uniquely determined using $P1$ and $P2$. Similarly given p and $c_1, ..c_{mx}$, $trail_P$ is uniquely determined. These properties are stated in the following Lemmas.

Lemma 3.3.1. *Given $trail_P$, points $c_1..c_{mx}$ are uniquely determined.*

Proof. Extend $Seg(C, N_{mx})$ of $trail_P$ by a distance of 2^{mx} to obtain c_{mx}. Similarly extend $Seg(N_k, N_{k-1})$ by 2^{k-1} to obtain c_{k-1} for $0 < k \leq mx$. Thus using properties $P1$ and $P2$ of $trail_p$, points $c_1..c_{mx}$ are uniquely determined given $C, N_{mx}, .., N_1, p$ of $trail_P$. □

Lemma 3.3.2. *Given $c_1, ...c_{mx}$ and p, $trail_P$ is uniquely determined.*

Proof. N_{mx} lies on $Seg(C, c_{mx})$ such that $dist(c_{mx}, N_{mx}) = 2^{mx}$. Similarly N_{k-1} lies on $Seg(N_k, c_{k-1})$ such that $dist(c_{k-1}, N_{k-1}) = 2^{k-1}$ for $1 < k \leq mx$. Thus $C, N_{mx}, .., N_1, p$ of $trail_P$ are uniquely determined. □

By property $P3$, the maximum separation between p and any auxiliary point c_k decreases exponentially as k decreases from mx to 1. When an object moves a certain distance away from its current location, $trail_P$ has to be updated from the smallest index k such that property $P3$ holds at all levels. By changing parameter b in property $P3$, we can tune the rate at which the tracking structure is updated. We discuss these refinements in Section 3.3.4.

Note from the definition of $trail_P$ that mx is defined as $\lceil (log_2(dist(C, p_o))) \rceil - 1$ where p_o was the location of the object when $trail_P$ was either created or updated from

C. The value of mx which denotes the number of *trail segments* in $trail_P$, depends on the distance of P from C. When $trail_P$ is first created, $c_1, ..., c_{mx}$ are initialized to location p_o, the number of levels mx is initialized to $\lceil (log_2(dist(C, p_o))) \rceil - 1$ and $trail_P$ is a straight line. The value of mx is updated when $trail_P$ has moved a sufficient distance to warrant an update of $trail_P$ all the way from C. The update (and create) procedure for $trail_P$ is described in more detail in the following subsection.

We now show 3 examples of the tracking structure in Fig. 1. In this figure, $b = 1$. Fig. 3.5(a) shows $trail_P$ when $c_3, ..c_1$ are collocated. When P moves away from this location, $trail_P$ is updated and Fig. 3.5(b) shows an example of $trail_P$ where c_2, c_1 are displaced from c_3. In Fig. 3.5(b), $dist(c_3, c_2) = 2$ units, $dist(c_2, c_1) = 1$ unit, and p and $c1$ are collocated. Moreover, N_3 lies on $Seg(C, c_3)$, N_2 lies on $Seg(N_3, c_2)$ and so on. In Fig. 3.5(c) we show an example of a zigzag trail to an object P, when P moves away from c_3 and then moves back in the opposite direction.

Updating the trail

We now describe a procedure to update the tracking structure when object P moves from location p' to p such that the properties of the tracking structure are maintained and the cost of update is distance sensitive.

Overview: When an object moves distance d away, we find the minimal index m, along $trail_P$ such that $dist(p, c_j) < 2^{j-b}$ for all j such that $mx \geq j \geq m$ and $trail_P$ is updated starting from N_m. In order to update $trail_P$ starting from N_m, we find new vertices $N_{m-1}, ..., N_1$ and a new set of auxiliary points $c_{m-1}, ..., c_1$. Let $N'_{m-1}, ..., N'_1$ and $c'_{m-1}, ..., c'_1$ denote the old vertices and old auxiliary points respectively. Starting from N_m, we follow a recursive procedure to update $trail_P$. This procedure is stated below:

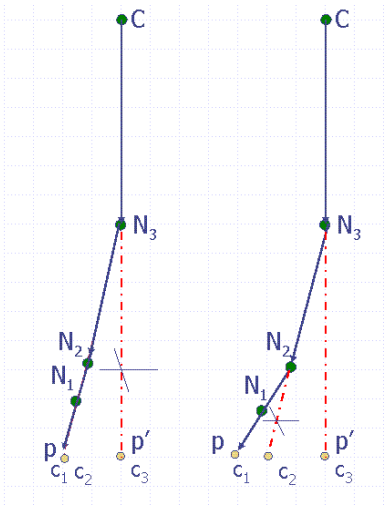


Figure 3.6: Updating $trail_P$

Update **Algorithm:**

1. Let m be the minimal index on the trail such that $dist(p, c_j) < 2^{j-b}$ for all j such that $mx \geq j \geq m$.

2. $k = m$

3. while $k > 1$

   (a) $c_{k-1} = p$; Now obtain N_{k-1} using property $P2$ as follows: the point on segment N_k, c_{k-1}, that is 2^{k-1} away from c_{k-1}.

   (b) $k = k - 1$

If no indices exist such that $dist(p, c_j) < 2^{j-b}$, then the trail is created starting from C. This could happen if the object is new or if the object has moved a sufficiently large distance from its original position. In this case, mx is set to $(\lceil log_2(dist(C, p)) \rceil) - 1$. c_{mx} is set to p. N_{mx} is marked on $Seg(C, p)$ at distance 2^{mx} from c_{mx}. Step 2 is executed with $k = mx$. □

Fig. 3.6 illustrates an update operation, when $b = 1$. In Fig. 3.6a, $dist(p, p')$ is 2 units. Hence update starts at N_3. Initially c_3, c'_2, c'_1 are at p'. We use the *update* algorithm to determine new c_2, c_1 and thereby the new N_2, N_1. Using step (3a) of the update algorithm, the new c_2 and c_1 lie at p. The vertex N_2 then lies on $Seg(N_3, c_2)$ and N_1 lies on $Seg(N_2, c_1)$. In Fig. 3.6b, P moves further one unit. Hence update now starts at N_2. Using step (3a) of the update algorithm, the new c_1 lies at p and N_1 lies on $Seg(N_2, c_1)$.

Note: The *Trail* update algorithm described above is for a continuous 2-d plane. In this algorithm, we have stated the minimum level from which $trail_P$ is updated and described how the vertices and auxiliary points are updated starting from this level. In Section 3.3.3, we describe *Trail* protocol on a wireless sensor network. A formal specification of the algorithm, in the form of guarded command actions, is also provided in Section 3.3.3. □

Lemma 3.3.3. *The update algorithm for* Trail *yields a path that satisfies* $trail_P$.

Proof.

1. Let m be the index at which *update* starts. By the condition in step 1, $dist(c_j, p) < 2^{j-b}$ for all $mx \geq j \geq m$. Now, for $m > j \geq 1$, $c_j = p$. Therefore for $m > j \geq 1$, $dist(c_j, p) < 2^{j-b}$. Thus property $P3$ is satisfied.

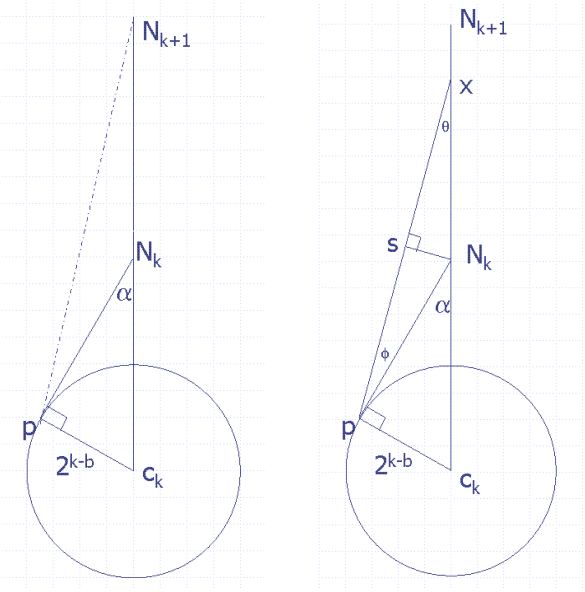
2. Properties $P2$ and $P1$ are satisfied because $m \geq k > 1$, we obtain N_{k-1} as the point on $Seg(N_k, c_{k-1})$, that is 2^{k-1} away from c_{k-1}.

3. mx is defined for $trail_P$, when $trail_P$ is created or updated starting from C. When mx is (re)defined for $trail_P$, c_{mx} is the position of the object and mx is set to $(\lceil log_2(dist(C, p))\rceil) - 1$. □

□

Definition 13 (Trail Stretch Factor)**.** Given $trail_P$ to an object p, we define the trail stretch factor for any point x on $trail_P$ as the ratio of the length along $trail_P$ from x to p, to the Euclidean distance $dist(x, p)$. □

Lemma 3.3.4. *The maximum* Trail Stretch Factor *for any point along* $trail_P$*, denoted as* TS_p *is* $sec(\alpha) * sec(\frac{\alpha}{2})$ *where* $\alpha = arcsin(\frac{1}{2^b})$.



(a) Max Angle in Trail (b) Analyzing Stretch

Figure 3.7: Analyzing Trail Stretch Factor

Proof. We prove Lemma 3.3.4 by using the following steps.

◇ *(Maximum angle* $(\angle pN_kc_k)$*)* Let the maximum angle formed by p and c_k at N_k in $trail_P$ for $(mx \geq k \geq 1)$ be denoted as α. Refer Fig. 3.7(a). Recall from properties of $trail_P$ that $dist(N_k, c_k) = 2^k$ and $dist(p, c_k) < 2^{k-b}$. Note that $\angle pN_kc_k$ is maximum when $Seg(N_k, p)$ is tangent to a circle of radius 2^{k-b} and center c_k. Therefore we have the following condition.

$$\alpha < arcsin(\frac{1}{2^b}) \tag{3.8}$$

$\diamond$ *(Maximum stretch at a given level k)* Let x be any point on $trail_P$ which lies on $Seg(N_{k+1}, N_k)$. Refer Fig. 3.7(b). We note the following equation based on the geometry of Fig. 3.7(b).

$$\frac{(dist(x, N_k) + dist(N_k, p))}{dist(x, p)} = \frac{(sin(\theta) + sin(\phi))}{sin(\theta + \phi)} \tag{3.9}$$

Also note that $(\theta + \phi) = \angle pN_kc_k$. Using this, we get Eq. 3.10

$$(\theta + \phi) \leq \alpha \tag{3.10}$$

Let $f(\theta, \phi)$ denote the following function.

$$f(\theta, \phi) = \frac{(sin(\theta) + sin(\phi))}{sin(\theta + \phi)} \tag{3.11}$$

It can be shown that $f(\theta, \phi)$, where $\theta > 0$, $\phi > 0$ and $\theta + \phi \leq \alpha$ is maximum when $\theta = \phi = \frac{\alpha}{2}$. We state this as Proposition A.0.1 and prove it in Appendix A. Substituting $(\theta = \phi)$ in Eq. 3.9, we get the following condition.

$$\frac{(dist(x, N_k) + dist(N_k, p))}{dist(x, p)} \leq sec(\frac{\alpha}{2}) \tag{3.12}$$

Thus, we see that at a single level k the maximum stretch for $trail_P$ occurs when $\angle pN_kc_k$ is α for a point x on $Seg(N_{k+1}, N_k)$ such that $\angle pxN_k = \angle xpN_k$. Since $\angle pxN_k = \angle xpN_k$ when the maximum occurs, we also get the following equation.

$$\frac{(dist(x, N_k))}{dist(x, p)} = \frac{(dist(N_k, p))}{dist(x, p)} = \frac{sec(\frac{\alpha}{2})}{2} \tag{3.13}$$

◇ *(Maximum trail stretch factor from vertex N_k to p)* In order to find the maximum stretch factor over multiple levels, we consider $trail_P$ to be *split* at vertices N_k to N_2 in such a way that the stretch is maximized at each level. Thus we let $\angle pN_jc_j = \alpha$ and $\angle pN_jN_{j-1} = \angle N_jpN_{j-1}$ for all $k \geq j > 1$. In Fig. 3.8, we show one such *trail segment*, $Seg(N_j, N_{j-1})$ of $trail_P$.

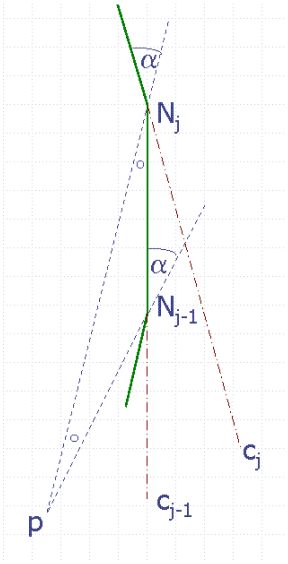


Figure 3.8: Analyzing Trail Stretch

Using Eq. 3.13, we get the following equations:

$$\frac{dist(N_{j-1}, p)}{dist(N_j, p)} = \frac{sec(\frac{\alpha}{2})}{2} \qquad \forall j : \; k \geq j > 1 \tag{3.14}$$

$$\frac{dist(N_j, N_{j-1})}{dist(N_j, p)} = \frac{sec(\frac{\alpha}{2})}{2} \qquad \forall j : \; k \geq j > 1 \tag{3.15}$$

When the above configuration is repeated at all levels of $trail_P$, we determine the ratio of the lengths of two successive *trail segments*, $Seg(N_{j-1}, N_{j-2})$ and $Seg(N_j, N_{j-1})$. Using Eq. 3.14 and Eq. 3.15, we get the following equation.

$$\frac{dist(N_{j-1}, N_{j-2})}{dist(N_j, N_{j-1})} = \frac{sec(\frac{\alpha}{2})}{2} \qquad \forall j: \ k \geq j > 2 \tag{3.16}$$

Let L_k denote the length along $trail_P$ from vertex N_k. Using Eq. 3.16, we get:

$$L_k = dist(N_2, N_1) * \sum_{j=0}^{j=(k-2)} (2 * cos(\frac{\alpha}{2})^j) + dist(N_1, p) \tag{3.17}$$

Let R_k denote the trail stretch factor from N_k.

$$R_k = \frac{L_k}{dist(N_k, p)} \tag{3.18}$$

Upon simplification using Eq. 3.14, Eq. 3.15 and Eq. 3.17, we get:

$$R_k = \frac{1}{2 * cos(\frac{\alpha}{2}) - 1} + \frac{1}{(2 * cos(\frac{\alpha}{2}))^{k-1}} * (1 - \frac{1}{2 * cos(\frac{\alpha}{2}) - 1}) \tag{3.19}$$

Since, $\alpha < \frac{\pi}{6}$ for $b \geq 1$, $0 < 2 * cos(\frac{\alpha}{2}) - 1 \leq 1$. Therefore we get:

$$R_k \leq \frac{1}{2 * cos(\frac{\alpha}{2}) - 1} \tag{3.20}$$

Since, $cos(\alpha) = 2 * cos^2(\frac{\alpha}{2}) - 1$, $cos(\frac{\alpha}{2}) \leq 1$ and $0 < 2 * cos(\frac{\alpha}{2}) - 1 \leq 1$, we get:

$$R_k \leq \frac{1}{cos(\alpha)} \tag{3.21}$$

◇ *(Maximum trail stretch factor from any point in $trail_P$ to p)* Let x be any point on $trail_P$ which lies on a level $k+1$ segment, i.e $Seg(N_{k+1}, N_k)$, but is not a *vertex* point. Let L_{xp} denote the length along $trail_P$ from x to p. Using Eq. 3.21 and Eq. 3.12, we have the following inequalities:

$$\begin{aligned} L_{xp} &\leq dist(x, N_k) + dist(N_k, p) * sec(\alpha) \\ &\leq (dist(x, N_k) + dist(N_k, p)) * sec(\alpha) \\ &\leq sec(\frac{\alpha}{2}) * sec(\alpha) * dist(x, p) \end{aligned}$$

Thus we have proved that the maximum *Trail Stretch Factor* for any point along $trail_P$, denoted as TS_p is $sec(\alpha) * sec(\frac{\alpha}{2})$ where $\alpha = arcsin(\frac{1}{2^b})$. □

Lemma 3.3.5. *The length of $trail_P$ for an object P starting from a level $k(0 < k \leq mx)$ vertex, denoted as L_k is bounded by $(2^k + 2^{k-b}) * TS_p$.*

Proof Sketch: $dist(c_k, p) < 2^{k-b}$. Therefore, $dist(N_k, p) < 2^k + 2^{k-b}$. Then using Lemma 3.3.4, the result follows. □

Theorem 3.3.6. *The upper bound on the amortized cost of updating $trail_P$ when object P moves distance $d_m(d_m \geq 1)$ is $4 * (2^b + 1) * TS_p * d_m * log(d_m)$.*

Proof. Note that in *update* whenever $trail_P$ is updated starting at the level k vertex, we set $c_{k-1} = p$. P can now move a distance of 2^{k-1-b} before another update starting at the level k vertex. Thus, between any two successive updates starting from a level k vertex, the object must have moved at least a distance of 2^{k-1-b}. The total cost to create a new path and delete the old path starting from a level k vertex costs at most $2 * L_k$.

When an object moves a total distance d_m where $d_m \geq 1$, it could involve multiple updates at smaller distances. The object could be detected at multiple instances over this distance d_m. Therefore we calculate the upper bound on the amortized cost of *update* when the object moves distance d_m. We consider the minimum distance to

trigger an update to be 1 unit. Note that between any two successive updates starting from a level k vertex, the object must have moved at least a distance of 2^{k-1-b}. Thus over a distance d_m, update can start at level $(b+1)$ vertex at most d_m times, update can start at level $(b+2)$ vertex can at most $\lfloor d_m/2 \rfloor$ times, and so on. The *update* can start at level $(\lfloor log_2(d_m) \rfloor + b + 1)$ vertex at most once. Adding the total cost, the result follows.

□

Remark on *update* **cost:** We note the following points about the *update* cost characterized in Theorem 3.3.6. (1) First of all, over a distance of d_m, the object can be *detected* in multiple instances and consequently multiple *update* of the track can result over a distance d_m, but the total cost of all the updates over a distance d_m is $O(d_m * log(d_m))$. We consider the minimum distance to trigger an update to be 1 unit. We add the total cost resulting from each *update* and the sum of the cost of *all* these possible updates results in an upper bound on the amortized cost, stated in Theorem 3.3.6. (2) Secondly, it is *only* the amortized cost of *update* which is of the order of $O(d_m * log(d_m))$ and distance sensitive. During a move of distance d_m by any object P, there could be updates triggered over a smaller distance d_{ms} (for instance, $d_{ms} = 1$), that result in $trail_P$ being updated from a much larger level $k >> 1$. But the *update* algorithm of *Trail* guarantees that whenever $trail_P$ is updated starting at the level k vertex, P can now move a distance of 2^{k-1-b} before another update starting at the level k vertex. Thus, between any two successive updates starting from a level k vertex, the object must have moved at least a distance of 2^{k-1-b}. There can only be updates at levels smaller than k between two updates at level k. Upon adding

up all these *update* costs, we get the amortized *update* cost which is of the order of $O(d_m * log(d_m))$ and distance sensitive.

b	Trail Stretch	*Update* Cost
1	1.2	$14 * d_m * logd_m$
2	1.05	$21 * d_m * logd_m$
> 3	Approaches 1	$4 * (2^b + 1) * d_m * logd_m$

Figure 3.9: Effect of b on *Update* cost

For illustration, we summarize the *Trail Stretch factor* and *update* costs for different values of b in Fig. 3.9. We explain the significance of the refinement of *Trail* by varying b in Section 5.

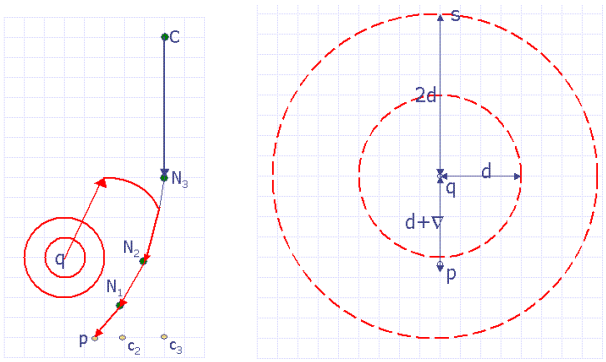
Basic Find Algorithm

Given $trail_P$ exists for an object P, we now describe a basic *find* algorithm that is initiated by object Q at point q on the plane. We use a basic ring search algorithm to intersect $trail_P$ starting from Q in a distance sensitive manner. We then show from the properties of the *Trail* tracking structure that starting from this intersection point, the current location of P is reached in a distance sensitive manner.

Basic *find* Algorithm:

1. With center q, successively draw circles of radius $2^0, 2^1, ...2^{\lfloor log(dist(q,C))\rfloor - 1}$, until $trail_P$ is intersected.

2. If $trail_P$ is intersected, follow it to reach object P; else follow $trail_P$ from C (note that if object exists, $trail_P$ will start from C).



(a) *find* path (b) Farthest *find* point

Figure 3.10: Basic find algorithm in *Trail*

Theorem 3.3.7. *The cost of finding an object P at point p from object Q at point q is $O(d_f)$ where d_f is $dist(p, q)$.*

Proof. Note that as q is distance d_f away from p, a circle of radius $2^{\lceil log(d_f) \rceil}$ will intersect $trail_P$. Hence the total length traveled along the circles before intersecting $trail_P$ at point s is bounded by $2 * \pi * \sum_{j=1}^{\lceil log(d_f) \rceil} 2^j$, i.e., $8 * \pi * d_f$. The total cost of connecting segments between the circles is bounded by $2 * d_f$.

Now, when the trail is intersected by the circle of radius $2^{\lceil log(d_f) \rceil}$, the point s at which the trail is intersected can be at most $3 * d_f$ away from the object p. This is illustrated in Fig. 3.10(b). In this figure, q is $d_f + \nabla$ away from p. Hence the trail can be missed by circle of radius $2^{log(d_f)}$. From Lemma 3.3.4, we have that distance along the trail from s to p is at most $3 * TS_p * d_f$. Thus, the cost of finding an object P at point p from object Q at point q is $O(d_f)$ where d_f is $dist(p, q)$. □ □

Update **cost when consisting of discrete jumps:** Note that the *update* costs characterized in Fig. 3.9 are for the continuous update case when updates are performed after every unit distance of move. Now consider the case where a total move

of distance d_m consists of multiple *discrete* jumps. In each jump, an object disappears at a certain location and is later detected at another location at distance d_x away such that $d_x >> 1$. In this case, we note that the cost of individual smaller updates need not be added. However there is an additional cost of exploring to find an existing trail because we do not assume memory of the previous location of an object. (In the continuous setting where updates performed after every one unit of move, we ignore the cost of finding an existing $trail_P$ as it will exist within a distance of 1 unit.) But recall from the $find$ algorithm that when P moves distance d_x away, an existing $trail_P$ will be intersected at an upper bound cost of $8 * \pi * d_x$, i.e. $O(d_x)$. Therefore the amortized *update* cost still remains $O(d_m * log(d_m))$ over a total distance d_m consisting of such discrete jumps.

3.3.3 Implementing Trail in a WSN

In this section, we describe how to implement the *Trail* protocol in a WSN that is a discrete plane, as opposed to a continuous plane as described in the previous section. *Trail* can be implemented in any random deployment of a WSN aided by some approximation for routing along a circle. We describe one such implementation below.

System model: Consider any random deployment of nodes in the WSN. We impose a virtual grid on this deployment and snap each node to its nearest grid location (x, y) and assume that each node is aware of this location. We refer to unit distance as the one hop communication distance. $dist(i, j)$ now stands for distance between nodes i and j in these units. The separation along the grid is less than or equal to the unit distance. When the network is dense, the grid separation can be smaller. The

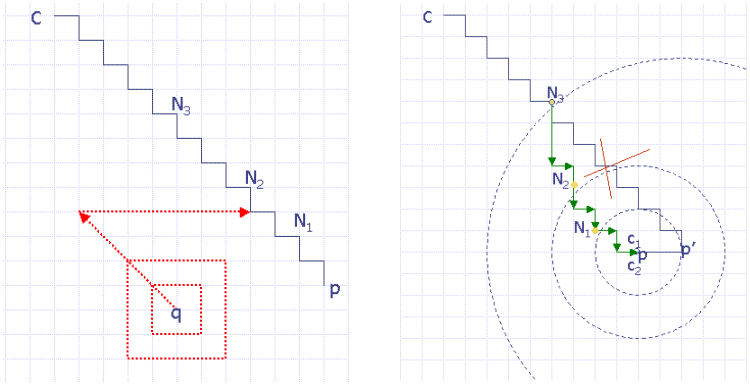
neighbors of a node are a set of all nodes within unit distance of the node. Thus when the grid separation is unit distance, there are at most 4 neighbors for each node. We also assume the existence of an underlying geographic routing protocol such as GPSR [38], aided by an underlying neighborhood service that maintains a list of neighbors at each node.

Note: We have implicitly assumed in the above model that each location on the grid is mapped to a unique node. This can be achieved by decreasing the size of the grid to be equal to the smallest separation between any 2 nodes in the network. However this assumption is not necessary. In other words, all nodes need not be necessarily assigned to some location on the grid. Nodes can take turns to play the role of a given location. But for ease of exposition, we have abstracted away these possibilities in our model.

Fault Model: We assume that nodes in the network can fail due to energy depletion or hardware faults, or there could be insufficient density at certain regions, thus leading to *holes* in the network. However, we assume that the network may not be partitioned; there exists a path between every pair of nodes in the network. A node may also transiently fail. But we assume that the failed nodes *return* in a clean or *null* state without any knowledge of tracks previously passing through them. We do not consider arbitrary state corruptions in the nodes.

When implementing on a WSN grid, *Trail* is affected by the following factors:(1) discretization of points to nearest grid location; (2) Overhead of routing between any two points on the grid; and (3) *holes* in the network. We discuss these issues in this section.

Routing stretch factor: When using geographic routing to route on a grid, the number of hops to communicate across a distance of d units will be more than d. We measure this stretch in terms of the routing stretch factor, defined as the ratio of the communication cost (number of transmissions) between any two grid locations, to the Euclidean distance d between two grid locations. It can be shown that the upper bound on the routing stretch factor for the WSN unit grid is $\sqrt{2}$. If we consider the grid to be of smaller separation than the communication range (denser grid), then the routing stretch factor will decrease as any straight line will now be approximated more closely when moving along the grid.



(a) *find* in a WSN grid (b) *update* in a WSN grid

Figure 3.11: Find and update algorithm in a WSN grid

Implementing *Update* on WSN Grid

Storage: For each object P in the network, $trail_P$ is maintained by parent / child pointers at each node in the network. Starting from C, following the child pointers for P will lead to the current location of P. Similarly, starting from p, following the parent pointers at each node will lead to C. Some of the nodes along $trail_p$ are marked as vertex nodes. Each node keeps memory of whether it is a vertex node.

Each vertex node at level m keeps memory of c_i for all levels $mx \geq i \geq m$, which is used to determine the smallest level at which an update should start. An array of size $log(N)$ is allocated at each node to store the auxiliary points, where the network is of size $N \times N$.

Protocol Trail at node j in NxN network
Var
$j.child_p$: child pointer for object p
$j.prnt_p$: parent pointer
$j.detect_p$: boolean
$j.level_p$: level at which node j belongs
$j.vertex_p$: boolean indicating if j is a vertex
$j.mx_p$: maximum levels
$j.c_p$: array [0..log(N)] of auxiliary points
nh : auxiliary variable to store the next hop for any message

Figure 3.12: Trail: State at Node j for Object P

The state of node j for a given object P being tracked is shown in Fig. 3.12. The actions involved in updating $trail_P$ (for $b = 1$) are shown in Fig. 3.13 and Fig. 3.14. A description of these actions follows.

Actions: We use three types of messages in the update actions. The update actions are initiated at a node where an object P is detected. Recall from Section 2 that this detection is flagged by the underlying detection and association service at the node that is closest to the location of P.

Initially, when an object is detected at a node j, it sends an *explore* message (**Action** $U1$). The parameters of this message are the object id and the location of the object. The *explore* message travels in around the square perimeters of side lengths, $2^0, 2^1, ...2^{\lfloor log(dist(q,C)) \rfloor - 1}$ until it meets $trail_P$ or else the *explore* message travels to C.

Note that if the object is updated continuously as it moves, then the *explore* message will intersect the trail within a 1 hop distance (first level of search). The auxiliary or thought variable nh is used to refer to the next hop of any message.

When an *explore* message is received at a node j (**Action** $U2$), if the node does not belong on $trail_P$, it simply forwards the message to the next hop of exploration. If *explore* message intersects $trail_P$, the message is forwarded along its parent pointer until the level m vertex node where m is the minimal index such that $dist(c_m, p) < 2^{m-1}$ for all i such that $mx \geq i \geq m$. Starting from the level m node where update is started, a new track is created by sending a *grow* message towards the current location of P. The parameters of a *grow* message are the identifier of the object, the current location of the object, the level w of the subsequent vertex, the maximum levels of $trail_P$ and the array containing values of c_i for all levels $mx \geq i \geq m$. Geographic routing is used to route the message towards the current location. If the exploration reaches C, the maximum levels of $trail_P$ are reset and a *grow* message is sent towards the location of P.

Upon receiving a *grow* message (**Action** $U3$), node j checks to see if it is a vertex node. The node closest to, but outside or on a circle of radius 2^w around c_w is marked as N_w. This node copies the values of maximum levels in $trail_p$, the level of node j in $trail_P$, and the values of c_i for all $mx \geq i > w$, from the grow message. Additionally, the value of c_w is set to current location of P. The child and parent pointers are updated to the sender of the *grow* message and the next hop towards the current location of P respectively. The *grow* message is then forwarded to the next hop using geographic routing [38]. If node j is not a vertex node, it simply marks its level in $trail_P$, updates the parent and child pointers and forwards the message to the next

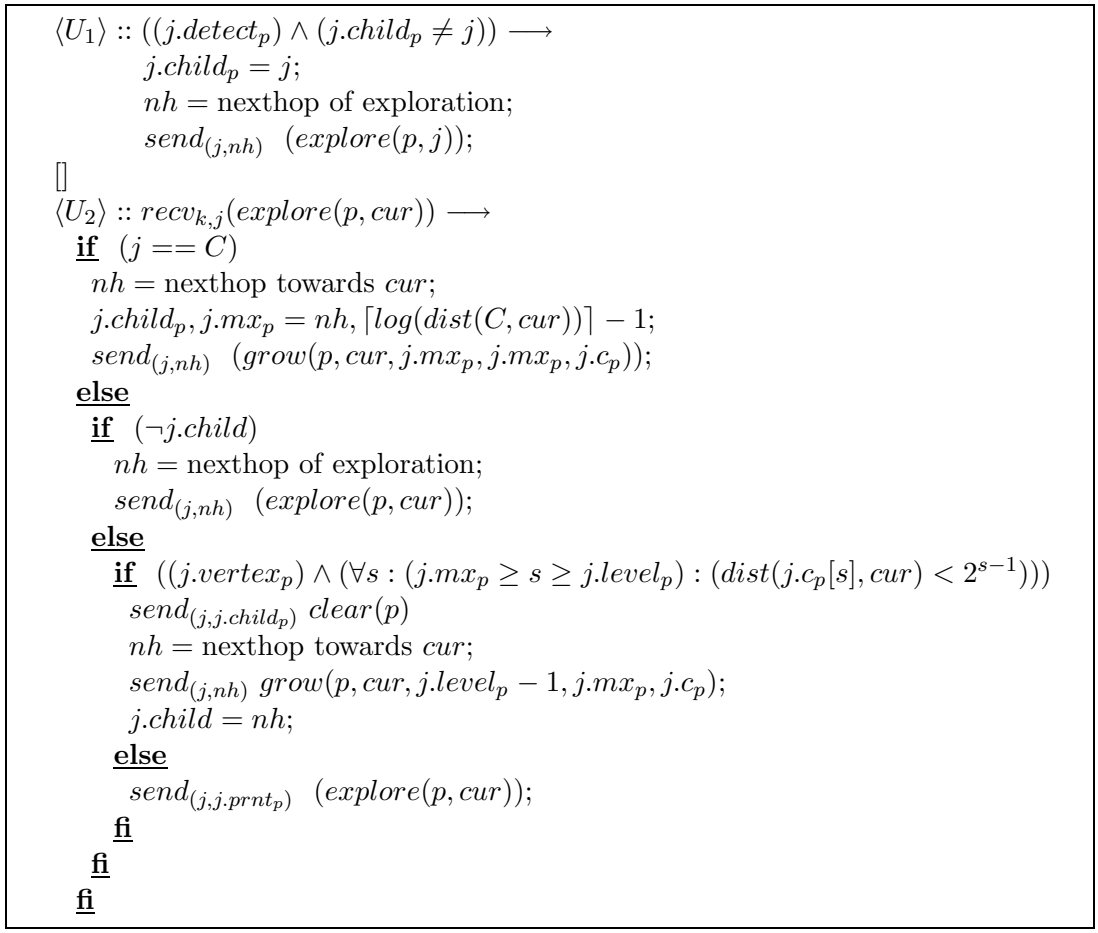


Figure 3.13: Trail: Update Actions (U1 and U2)

hop towards the current location of P. This procedure is then repeated at subsequent nodes and the trail is updated. If j is the current location of the object, the *grow* message is not propagated further. Fig. 3.11(b) shows how a trail is updated in the grid model with the grid spacing set equal to the unit communication distance. The vertex pointers $N_3, ...N_1$ are shown approximated on the boundary of the respective circles.

Also, starting from the level m node where update is started, a *clear* message is used to delete the old path. Upon receiving a *clear* message (**Action** $U4$), node j forwards the message to its child and the state of j with respect to object P is reset. The *clear* message is passed along child pointers of $trail_P$ until the node where object P previously resided.

Implementing $find$ on WSN Grid

We now describe how to implement the *find* algorithm in the WSN grid. As seen in Section 3.3.2, during a find, exploration is performed using circles of increasing radii around the finder. However, in the grid model, we approximate this procedure and instead of exploring around a circle of radius r, we explore along a square of side $2 * r$. The perimeter of the square spans a distance $8 * r$ instead of $2 * \pi * r$. We characterize the upper bound on the *find* cost in the following Lemma.

Lemma 3.3.8. *The upper bound on the cost of finding an object P at point p from object Q at point q is $(32 + 3 * sec(\alpha) * sec(\frac{\alpha}{2}) * \sqrt{2}) * d_f$ where d_f is $dist(p, q)$ and $\alpha = arcsin(\frac{1}{2^b})$.*

Proof. In the WSN virtual grid, the total cost of exploring along squares up to level $2^{log(d_f)}$is given by $8 * \sum_{j=0}^{\lceil logd \rceil} 2^j$, i.e $32 * d_f$. Recall from the proof of Theorem 3.7 that

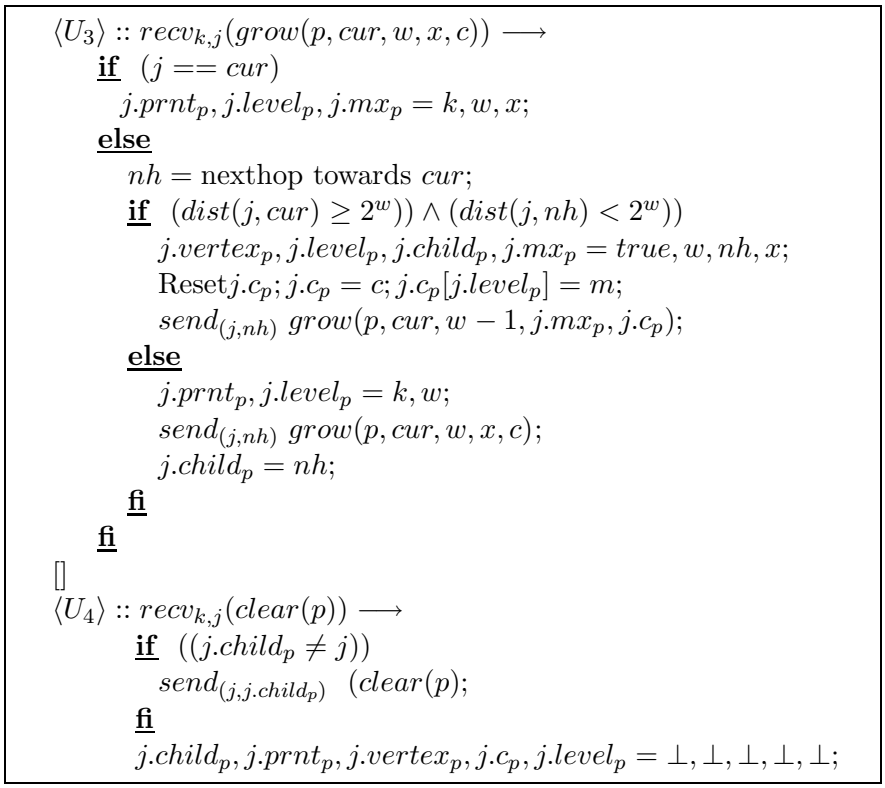


Figure 3.14: Trail: Update Actions (Actions U3 and U4)

when the trail is intersected by the circle of radius $2^{\lceil log(d_f) \rceil}$, the point s at which the trail is intersected can be at most $3 * d_f$ away from the object p. The cost of reaching p from the point of intersection with $trail_P$ is bounded by $3 * d_f * TS_p * \sqrt{2}$ where TS_p is the maximum trail stretch factor possible for P. Note that there is an additional stretch of $\sqrt{2}$ because of routing along a grid. The result follows.

□

The actions for the *find* algorithm are shown in Fig. 3.15. Upon receiving a *find(p,q)* message at node j (**Action** $F1$), if node j does not belong to $trail_P$, then the message is forwarded to the next hop of exploration or else the message is forwarded to the child node. If the current location of P is reached, then a *found* message is sent towards q. Upon receiving a *found* message (**Action** $F2$), the state of object P is returned to q using geographic routing.

Fault-Tolerance

Due to energy depletion and faults, some nodes may fail leading to *holes* in the WSN. *Trail* supports a graceful degradation in performance in the presence of node failures. As the number of failures increase, there is only a proportional increase in *find* and *update* costs as the tracking data structure and the find path get distorted. This is especially good when there are a large number of small *holes* in the network that are uniformly distributed across the network as has been the case in our experiments with large scale wireless sensor networks [9]. We discuss the robustness of *Trail* under three scenarios: during *update*, during *find* and maintaining an existing trail.

Tolerating node failures during *update*: A *grow* message is used to update a trail starting at a level k node and is directed towards the center of circle $k-1$. In the

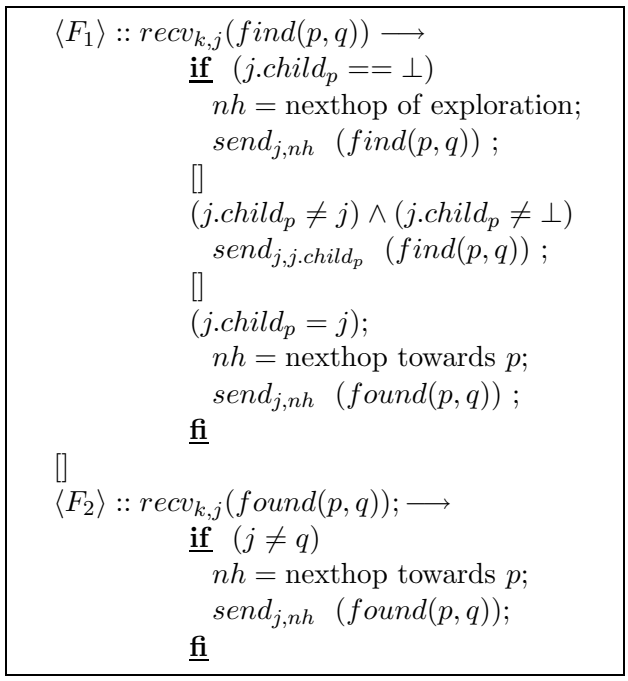
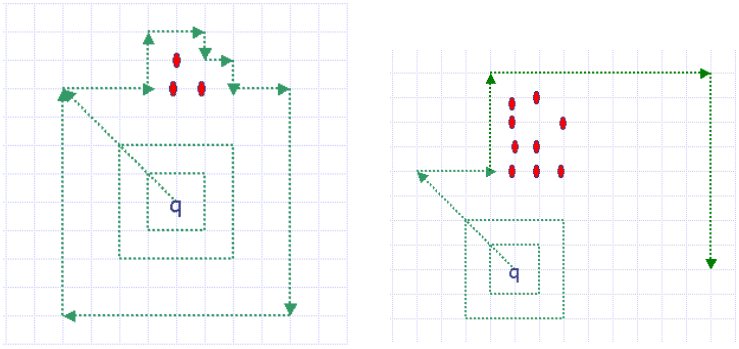


Figure 3.15: Trail: Find Actions

presence of holes, we use a right hand rule, such as in [38], in order to route around the hole and reach the destination. As indicated in the *update* algorithm for WSN grid, during routing the node closest to, but outside a circle of radius 2^{k-1} around c_{k-1} is marked as N_{k-1}. Since we assume that the network cannot be partitioned, eventually such a node will be found. (If all nodes along the circle have failed, the network is essentially partitioned).

Tolerating failures during a *find*: We now describe how the *find* message explores in squares of increasing levels. When a *find* message comes across a hole, it is rerouted around the hole using geographic routing only radially outwards of the current level square. If during the re-route, we reach a distance from the source of the *find* corresponding to the next level of search, we continue the search at the next

level and so on. Thus, in the presence of larger holes, we abandon the current level and move to the next level, instead of routing around the hole back to the current level of exploration.



(a) Find re-route along same level

(b) Find re-route to the next level

Figure 3.16: Tolerating failures during *find*

Maintaining an existing trail: Nodes may fail after a trail has been created. In order to stabilize from these states, we use periodic *heartbeat* actions along the trail. We assume that if a node has a transient failure, the node returns in a *null* state. We do not handle arbitrary state corruptions in the nodes.

Var

Hb : Interval for heartbeats
$j.time$: Local time
$j.sendHb_p$: Last time heartbeat was sent
$j.recvHb_p$: Last time heartbeat was received
$j.nextvertex$: next high level vertex along Trail

Figure 3.17: Trail: Additional State at Mote j for Stabilizing Actions

The heartbeat actions are sent by each node along $trail_P$ to its child. At any node r, if a heartbeat is not received from its parent, a *search* message is sent using geographic routing tracing the boundary of the hole. $trail_P$ is reinforced starting from the first node where the *search* message intersects $trail_P$ using a *reinforce* message along the reverse path. (If the goal is to find $trail_P$ in the shortest time, the *search* should likely be enforced in both directions along the boundary of the hole).

We have formally stated these maintenance actions in guarded command notation in Fig. 3.18.

Tolerating failure of C: The terminating point C provides a sense of direction to the trail and serves as a final landmark for the *find* operation. If C fails, the node that is closest to C will takeover the role of C. However, even in the transient stage when there is no C, the failure of C is tolerated locally. We describe this below.

Consider that C and all nodes in a contiguous region around C have failed. In this case, a *search* message will be initiated from the node closest to C that belongs to $trail_P$. Because a contiguous set of nodes surrounding C have failed, the *search* message eventually returns to the node initiating the *search* by following the boundary of the hole. Thus an existing $trail_P$ terminates at the node that belongs to $trail_p$ and is closest to C. Thus if a *find* message is unable to reach C, then routing along the boundary of the *hole* will intersect $trail_P$.

When a new node takes over the role of C, we do not assume that the state of the original C is transferred. The new C has no knowledge of tracks passing through it. Eventually, the maintenance actions for *Trail* will result in all tracks terminating at C.

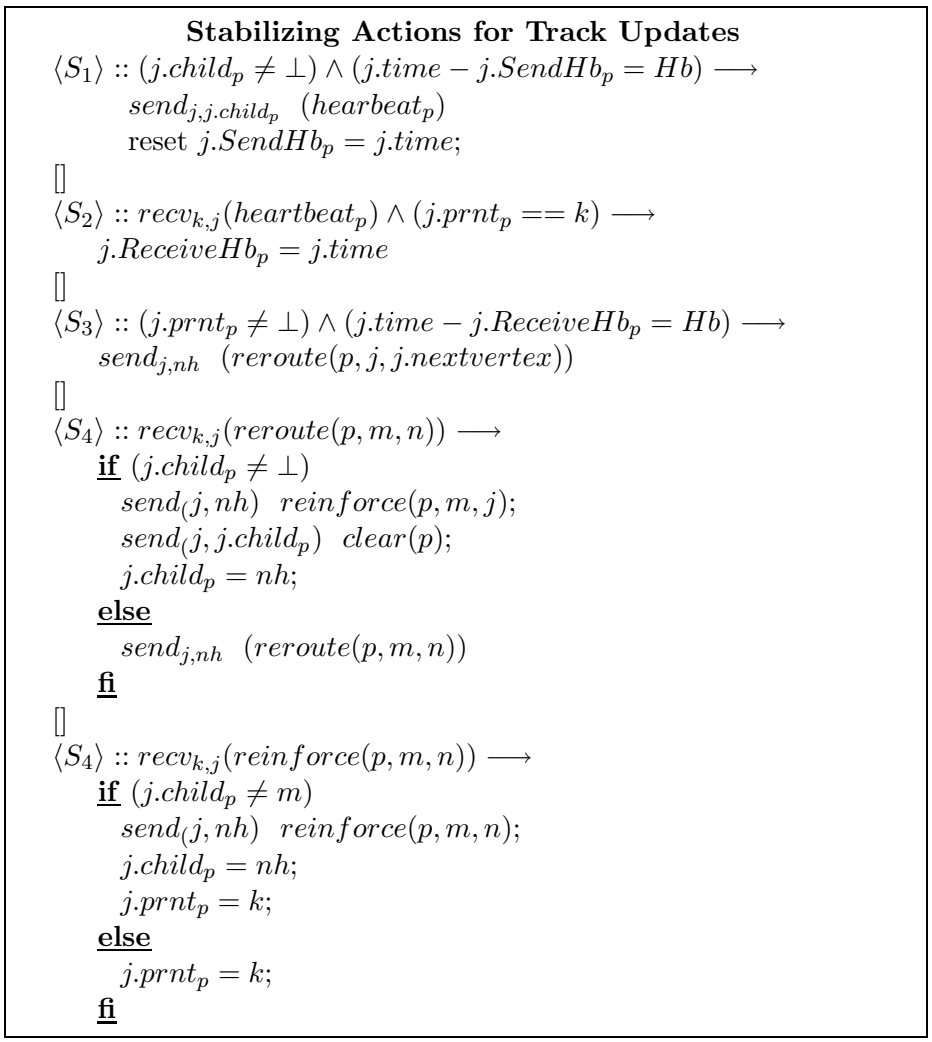


Figure 3.18: Trail: Stabilizing Actions

3.3.4 Refinements of Trail

In Subsections 3.3.2 and 3.3.3, we have described the basic *Trail* protocol. In this subsection, we discuss two techniques to refine the basic *Trail* network protocol: (1) tuning how often to update a *Trail* tracking structure, and (2) tuning the shape of a *Trail* tracking structure.

In the first refinement we alter the parameter b to values greater than 1. By increasing b, we update the track of an object more often. This results in straighter tracks with smaller stretch factor. As tracks get straighter, the *find* at higher levels of the search can follow a triangular pattern (as illustrated in Fig. 3.21) as opposed to complete circles. In fact, as b increases circular explorations can be avoided at more levels of the *find*. Thus the average find cost in the network decreases as b increases. In sum by increasing b, we increase the cost of update and decrease the cost of find. This refinement can be used when the rate of updates is small as compared to the rate of find. For example, as the speed of objects in the network decreases, we can increase the value of b.

In the second refinement we change the length of each segment in *Trail*. In the basic protocol, the segment at each level is a straight line to the next lower level. In this refinement, we modify the length of each segment to be a straight line to the next level plus an arc of length $x \times 2^k$. As x increases the amount of update at each level increases, but the *find* exploration can now be smaller. Specifically when $x = 2 \times \pi$, the *find* is a straight line towards the center of the network. We call this particular parameterization, the *find-centric Trail* protocol.

Tightness of Trail Tracking Structure

The frequency at which $trail_P$ is updated depends on parameter constant b in property $P3$ of $trail_P$. As seen in Section 3, for values of $b > 1$, $trail_P$ is updated more and more frequently, hence leading to larger update costs. However, $trail_P$ becomes tighter and tends to a straight line with the trail stretch factor approaching 1. We exploit this tightness of $trail_p$ to optimize the *find* strategy.

Optimization of *find*

We now describe the details of this optimization.

Lemma 3.3.9. *Given* $trail_P$, $(\angle C, p, N_k) < (mx - k + 1) * (arcsin(\frac{1}{2^b}))$, *where* $(mx \geq k \geq 1)$.

Proof. Recall from Eq. 3.8, $(\angle p, N_j, c_j) < (arcsin(\frac{1}{2^b}))$, where N_j is any level j vertex where $mx \geq j \geq 1$. Since N_j, N_{j-1} and c_{j-1} form a straight line, recall that $\angle N_j p N_{j-1} + \angle p N_j N_{j-1} = \angle p N_j c_j$ for $mx > j \geq 1$. Similarly, since C, N_{mx} and c_{mx} form a straight line, also recall that $\angle C p N_{mx} + \angle p C N_{mx} = \angle p N_{mx} c_{mx}$. Using these we have the following equations.

$$(\angle N_j p N_{j-1}) < (arcsin(\frac{1}{2^b})) \quad \forall j : \quad (mx > j \geq 1) \tag{3.22}$$

$$(\angle C, p, N_{mx}) < (arcsin(\frac{1}{2^b})) \tag{3.23}$$

Using Eq. 3.22 and Eq. 3.23, we obtain $\angle C p N_k$ by summing up as follows.

$$\begin{aligned} (\angle C, p, N_k) &= \angle C p N_{mx} + \sum_{j=k+1}^{mx} (\angle N_j p N_{j-1}) \\ &< (mx - k + 1) * (arcsin(\frac{1}{2^b})) \end{aligned}$$

□

From hereon, we let d_{pC} denote the distance of any object P from C. After the value of mx is defined for a $trail_P$, object P can move for a certain distance before mx is redefined. Therefore, given d_{pC}, the value of mx in $trail_P$ cannot be uniquely determined; however, we note that the value of mx can be bounded given d_{pC} and we define $\hat{mx}_p$ as the highest possible value of mx in $trail_P$, given d_{pC}. We now determine $\hat{mx}_p$.

Let R denote the network *radius*, defined as the maximum distance from C. Recall that mx denotes the number of levels in the track for an object P. mx is defined as $\lceil(log(dist(C, p_o)))\rceil - 1$ where p_o is the position of the object when $trail_P$ was (re)created from C. Given network radius R, let $\top$ be the highest number of levels possible for any object in the network. Thus in a given network, $\top = \lceil(log(R))\rceil - 1$.

Lemma 3.3.10. *Given d_{pC}, $\hat{mx}_p = minimum(\lceil log(d_{pC})\rceil, \top)$.*

Proof. Let mx be the index of the highest level in $trail_P$. Using property $P3$ we get that $dist(p, c_{mx}) < 2^{mx-b}$.

$$\begin{aligned} dist(C, c_{mx}) &\leq d_{pC} + dist(p, c_{mx}) \\ &< d_{pC} + 2^{mx-b} \end{aligned}$$

By the definition of mx, $2^{mx} < dist(C, c_{mx})$. Therefore we get the following equation.

$$2^{mx} < d_{pC} + 2^{mx-b}$$

Since $b \geq 1$, we get the following equation.

$$d_{pC} > 2^{mx-1}$$

Thus $mx \leq \lceil(log(d_{pC}))\rceil$. Hence, $\hat{mx}_p = minimum(\lceil log(d_{pC})\rceil, \top)$. □

Since given d_{pC}, the index of highest level mx in $trail_p$ cannot be uniquely determined, we state the maximum angle formed by $\angle CpN_k$ in terms of $\hat{mx}_p$ rather than

mx. When the actual mx in $trail_P$ is lesser than $\hat{mx}_p$, then the actual maximum angles formed by $\angle C, p, N_k$ where $1 \leq k \leq mx$ is lower than the maximum angles stated in the following equation.

$$(\angle C, p, N_k) < (\hat{mx}_p - k + 1) * (arcsin(\frac{1}{2^b})) \tag{3.24}$$

In the analysis below, we characterize the minimum size of exploration required at each level of exploration given the distance of finder object q from C. Only for ease of explanation, we assume that $b = 3$.

Let Q be the finder at distance d_{qC} from C. Thus $\hat{mx}_q = minimum(\lceil log(d_{qC}) \rceil, \top)$. Let P be an object which should be found at the level k exploration. At level k of the exploration, $trail_P$ for any location of P within the circle of radius 2^k around q should be intersected. A circular exploration of radius 2^k around q is sufficient to achieve this. We now characterize the necessary exploration.

We show that, at levels of exploration k where $k \geq \hat{mx}_q - 7$, circular explorations can be avoided and instead a pattern of exploration along the base of an isosceles triangle with apex q and length of base determined by Fig. 3.20 is sufficient to intersect the trails of all objects at distance 2^k from Q. The base of the isosceles triangle is such that $segment(C, q)$ is the perpendicular and equal bisector of the base of the triangle. At levels of exploration $k < \hat{mx}_q - 7$, exploration along the entire circle is necessary.

Analysis of necessary exploration for optimized find algorithm Let Q be the finder at distance d_{qC} from C. Thus $\hat{mx}_q = minimum(\lceil log(d_{qC}) \rceil, \top)$. In the basic *find* algorithm, the *find* operation will explore at all levels k where $0 \leq k \leq \lfloor log(d_{qC}) \rfloor - 1$ and at each level in circles of radius 2^k. In terms of $\hat{mx}_q$, the highest level of exploration for any location of q in the network is $\hat{mx}_q - 1$. This is because

when $\hat{mx}_q = \top$, it follows that $\hat{mx}_q = \lfloor log(d_{qC}) \rfloor$ and the highest level of exploration in $\hat{mx}_q - 1$. When $\hat{mx}_q = \lceil log(d_{qC}) \rceil$, the highest level of exploration is $\hat{mx}_q - 2$. Thus in either case, the level of exploration is bounded by $\hat{mx}_q - 1$.

Let P be an object which should be found at the level k exploration. At level k of the exploration, $trail_P$ for any location of P within the circle of radius 2^k around q should be intersected. A circular exploration of radius 2^k around q is sufficient to achieve this. We now determine the necessary exploration.

Since $dist(p, q) \leq 2^k$, $d_{pC} \leq d_{qC} + 2^k$. Thus at any level k of the exploration, $\hat{mx}_p \leq \lceil log(d_{qC} + 2^k) \rceil$. Note that $d_{qC} \leq \hat{mx}_q$ and $k \leq (\hat{mx}_q - 1)$. Therefore $\hat{mx}_p \leq \hat{mx}_q + 1$.

We now outline our procedure for level of exploration $k = \hat{mx}_q - 2$.

Level of Exploration $k = \hat{mx}_q - 2$ Refer to Fig. 3.19. Let α_b denote the value of $arcsin(\frac{1}{2^b})$ for a given b. Since in our analysis we consider $b = 3$, using Eq. 3.24 we get that the maximum angle $\angle CpN_k$ is $(\hat{mx}_p - k + 1) * (\alpha_3)$. Since the finder object Q is unaware of $\hat{mx}_p$, the worst case estimate for $\hat{mx}_p$ is used, i.e. $\hat{mx}_p = \hat{mx}_q + 1$. Thus, the maximum angle $\angle CpN_k = 3 * \alpha_3$.

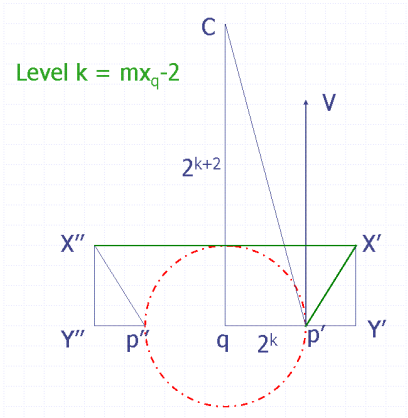


Figure 3.19: Level of exploration $k = \hat{mx}_q - 2$

Given this angle, we are interested in determining the smallest segment(X",X') that will intersect $trail_P$ for any location of P within the dotted circle. This is obtained by drawing a segment from point p' and p'' at angle $3*\alpha_3$ with segment(C,p') and segment(C,p") respectively. Point X' is obtained by extending this segment such that $dist(X', Y') = 2^k$. Point X'' is obtained similarly. Now segment(X', X") will intersect trails of all objects at distance 2^k from q, where $k = \hat{mx}_q - 2$.

From Fig. 3.19, we note that $\angle qCp' = arctan(\frac{1}{4})$ and $\angle Cp'X') = 3 * \alpha_3$. Plugging in the values we obtain the minimum required exploration at level $k = \hat{mx}_q - 2$ as follows (approximated to one decimal place):

$$dist(X', X'') \quad = \quad 2.5 * 2^k$$

Similarly, we determine the necessary pattern of exploration at levels $0, .., \hat{mx}_q - 1$. Finally we show that, at levels of exploration k where $k \geq \hat{mx}_q - 7$, circular explorations can be avoided and instead a pattern of exploration along the base of an isosceles triangle with apex q and length of base determined by Fig. 3.20 is sufficient to intersect the trails of all objects at distance 2^k from Q. The base of the isosceles triangle is such that $segment(C, q)$ is the perpendicular and equal bisector of the base of the triangle. At levels of exploration $k < \hat{mx}_q - 7$, exploration along the entire circle is necessary.

Note: All distances d_{qC} in the range $2^{\hat{mx}_q - 1} < d_{qC} \leq 2^{\hat{mx}_q}$, result in the same value of $\hat{mx}_q$. But in the above analysis, we assumed $d_{qC} = 2^{\hat{mx}_q}$. This results in finding the maximum exploration needed because when d_{qC} is smaller, $\angle Cp'X'$ increases, thus decreasing $\angle Vp'X'$ and lowering the length of exploration. □

Optimized find algorithm (b=3):

Exploration level k	Length of triangle base	Height of triangle
$m\hat{x}_q - 1$	$2 * 2^k$	2^k
$m\hat{x}_q - 2$	$2.5 * 2^k$	2^k
$m\hat{x}_q - 3$	$3.1 * 2^k$	2^k
$m\hat{x}_q - 4$	$3.7 * 2^k$	2^k
$m\hat{x}_q - 5$	$4.3 * 2^k$	2^k
$m\hat{x}_q - 6$	$5 * 2^k$	2^k
$m\hat{x}_q - 7$	$6.2 * 2^k$	2^k

Figure 3.20: Optimized *find*: pattern of exploration

1. Explore at levels k ranging from 0 to $(\lfloor log(d_{qC}) \rfloor - 1)$. If $k < (m\hat{x}_q - 7)$, explore using a circle of radius 2^k around q. Else explore along the base of an isosceles triangle with apex q and length of base determined by Fig. 3.20. The base of the triangle is such that $segment(C, q)$ is the perpendicular and equal bisector of the base of the triangle. □

An example for the modified *find* algorithm is shown in Fig. 3.21. In this figure, the object q is at distance 48 units from C. $m\hat{x}_q$ is 6. $\lfloor log(d_{qC}) \rfloor - 1 = 4$. Therefore, the levels of exploration are in the range 0..4. Exploration is along the base of triangles at all levels. (This figure is not to scale but for illustration.)

Impact of the Optimization: The optimization of *find* at higher levels is thus significant in that it yields: (1) smaller upper bounds for objects that are far away from the finder; and (2) lower average cost of $find(p, q)$ over all possible locations of q and p.

As described earlier when $b = 3$, circular explorations are avoided at the highest 7 levels of the *find* operation. As the value of b increases, the number of levels at which circular explorations can be avoided, increases. But by increasing b, we update the

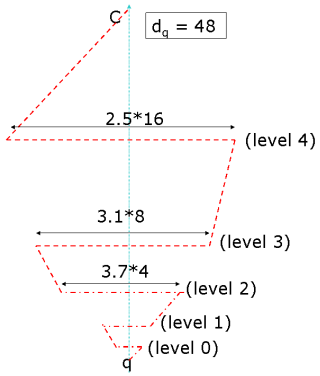


Figure 3.21: Optimized *find*: example

track of an object more often. Thus by increasing b, we increase the cost of update and decrease the cost of find.

We note that there are limits to tuning the frequency of updates, because for extreme values of b distance sensitivity may be violated. For example, for large values of b, that cause $dist(p, c_k) < y$ where y is a constant we end up with having to update the entire $trail_P$ when an object moves only a constant distance y. Similarly, for values of $b < 0$, the *Trail Stretch Factor* becomes unbounded with respect to distance from an object. Thus an object could be only δ away from a point on $trail_P$, yet the distance along $trail_P$ from this point to the p could travel across the network.

3.3.5 Modifying Trail Segments

The second refinement to *Trail* is by varying the shape of the tracking structure by generalizing property $P2$ of $trail_P$. Instead of *trail segment* k between vertex N_k and N_{k-1} being a straight line, we relax the requirement on *trail segment* k to be of length at most $(2 * \pi + 1) * 2^k$. By publishing information of P along more points, the *find* path can be more straight towards C. An extreme case is when trail segment k is

a full circle of radius 2^k centered at c_k and $Seg(N_k, N_{k-1})$. We call this variation of *Trail* the *find-centric Trail*.

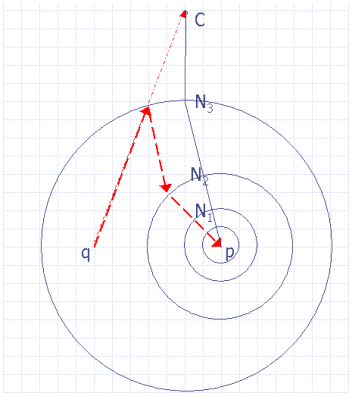


Figure 3.22: Find-centric Trail

Find-centric Trail

In this refinement, the *find* procedure eschews exploring the circles (thus traversing only straight line segments) at the expense of the update procedure doing more work. This alternative data structure is used when objects are static or when object updates are less frequent than that of *find* queries in a system. Let $trail_P$ for object P consist of segments connecting $C, N_{mx}, .., N_1, p$ as described before and, additionally, let all points on the circles $Circ_k$ of center c_k and radius 2^k contain pointers to their respective centers, where $mx \geq k > 0$.

Starting at q, the *find* path now is a straight line towards the center, as seen in Fig. 3.22. If a circle with information about object P is intersected then, starting from this point, a line is drawn towards the center of the circle. Upon intersecting the immediate inner circle (if there is one), information about its respective center is found, with which a line is drawn to this center. Object P is reached by following

this procedure recursively. We characterize the upper bound on the *find* cost in the following Lemma.

Lemma 3.3.11. *In* find-centric Trail*, when* $b = 1$*, the total cost of finding an object* P *at point* p *from object* Q *at point* q *is* $16 * d_f$ *where* $d_f = dist(p, q)$. □

Proof. Let Q lie between circles of level k and $k-1$ of the *find-centric* trail for P. The worst case find cost occurs when q is just outside the level $k-1$ circle. Note that $dist(p, c_{k-1}) < 2^{k-2}$ and therefore $dist(p, q) > 2^{k-2}$

Now, q can travel distance $2 * 2^k$ to reach circle k. Let the point of intersection of the *find* path from q and circle k be t. The cost of following pointers from t to the centers of inner circles recursively and reaching P is given by $(2^0 + 2^1 + ... + 2^k)$, i.e., $2 * 2^k$.

The ratio of find cost to the distance is thus less than $\frac{4*2^k}{2^{k-2}}$, i.e 16. Hence if $dist(p, q) = d$, then the maximum cost of finding object P is $16 * d$.

□

We note that when events are static, the optimal publish structure is much smaller than publishing along circular tracks. We have studied optimal publish structures for querying in a static context in a related work [23].

Summary: In this section, we presented two refinements of Trail that lead to a family of protocols. In the first refinement we altered the rate of updates. In the second refinement, we altered the amount of updates at each level. One can choose an appropriate parameterization depending on the expected rate of updates and finds in the network.

As an example, given the expected rate of updates and expected rate of find operations in the network, we can use the value of b in refinement 1 that minimizes the sum of update costs and find cost over a given interval of time. We can compare this cost with that of *find-centric Trail* and then choose the most appropriate parameterization. The find centric version of trail is especially beneficial when the rate of updates is much smaller than that of finds.

3.3.6 Discussion

In our solution, we made some design decisions like choosing a single point to terminate tracks from all points in the network and avoiding hierarchy in maintaining the tracks. In this section, we analyze these aspects of our solution and compare them with other possible approaches. We find that by avoiding hierarchy, we do not need to partition the network into clusters and maintain these clusters, we can be more locally fault-tolerant and we can obtain tighter tracks for any object. We also formally define the notion of terminating points, differentiate those from clusterheads of a hierarchy, and analyze the effect of more terminating points on the maximum *find* cost and maximum *update* cost in the network.

Terminating points vs clusterheads: There are some hierarchy based solutions [24, 27] for the problem of object tracking in a distance sensitive manner, where the network is hierarchically partitioned into clusters and information of objects is maintained at clusterheads[1] at each level. Even in these solutions, information about an object is published across the network to local clusterhead(s) at each level in the hierarchy, all the way up to one or more clusterheads at the highest level in the

[1]Note that the responsibility of a clusterhead for every cluster could be shared by multiple nodes and not necessarily just one node. We refer to the abstraction of a leader for every cluster as clusterhead

hierarchy. We call these points at the highest level of the hierarchy as terminating points.

Formally, a terminating set τ is *a* smallest set of points such that tracks of objects from every location in the network pass through at least one point in τ. The cardinality of a set τ is denoted as μ_τ. There can be one or more terminating sets, each with one or more terminating points. In *Stalk* [24] there is a unique clusterhead at the highest level, thus there is a single terminating set with a single terminating point. In *LLS* [1] and *DSIB* [27], there are multiple clusterheads at the highest level. Thus there are multiple terminating sets, each with one terminating point. Tracks from every point in the network pass through each of those clusterheads. Thus each clusterhead at the highest level constitutes a terminating set by itself. In *Trail* there is a unique terminating set with a unique terminating point, namely C.

It is in the process of maintaining tracks from a terminating point that we have avoided hierarchy in *Trail*. In hierarchy based solutions, to maintain tracks and to answer queries, tracks from terminating points necessarily pass through these clusterheads, where as *Trail* avoids hierarchy by determining anchors for the tracking paths on-the-fly based on the motion of objects.

Merits of avoiding hierarchy: By avoiding hierarchical solutions we do not need either a distributed clustering service that partitions the network into clusterheads at different levels and maintains this clustering or a special (maybe centralized) allocation of infrastructure nodes. By avoiding a hierarchy of such special nodes *Trail* is also more locally fault tolerant. For example in the case of a *find* operation, failure to retrieve information from an information server at a given level would require the *find* to proceed to a server at the higher level [27]. This is particularly expensive at

higher levels of the hierarchy. On the other hand in *Trail* a *find* operation redirects around a *hole* created by failed nodes using routing techniques such as the left hand rule [38] and such faults can be handled, in a sense, proportional to the size of the fault. Similarly, tracks to existing objects can be repaired more efficiently. As the number of failures increase, there is only a proportional increase in *find* and *update* costs.

Moreover, avoiding hierarchy allows for minimizing the length of tracking paths given a terminating point. We analytically compare the performance of *Trail* with that of other hierarchy based solutions for tracking objects in Section 3.5 and we observe that *Trail* is more efficient than other solutions. *Trail* has about 7 times lower *update* costs at almost equal *find* costs. By using a tighter tracking structure, we are also able to decrease the upper bound *find* costs at larger distances and thereby decrease the average find cost across the network.

Choice of terminating points: In Appendix B, we have formally analyzed the choice of a unique terminating point for tracks from all points in the network and the tradeoffs associated with multiple terminating sets and multiple terminating points per set in terms of the maximum *find* and *update* costs in the network. We provide a summary of our analysis here.

We show in our analysis that in order for *find* to be distance sensitive, it must the case that all terminating points in a terminating set must be traversable in $O(N)$ where the network is of $N \times N$ dimensions. This precludes the possibility of track from every point terminating at itself (or in other words, the terminating set being equal to the set of all points in the network). So the question arises as to what are the choices for number of terminating points and terminating sets. We consider 3 cases:

a single terminating set with multiple terminating points, multiple terminating sets each with one terminating point and a different terminating point for each *type* of object.

Single terminating set with multiple terminating points: Intuitively, there exists a possibility of decreasing the maximum track length in the network by dividing the network into regions and having a local terminating point per region. The maximum track length and therefore the maximum *update* cost in the network thus depends on the size of the largest region. We are faced with the question of how small can these regions be. We show that in order to maintain *find* distance sensitivity, the diameter of the largest region can only be a constant order less than the diameter of the network, i.e., at least $\Omega(N)$ in a $N \times N$ network. Thus, there can be only a constant order of cost decrease compared to having only one terminating point. Moreover, decreasing the maximum *update* cost by dividing the network into smaller regions results in proportionate increase in the maximum *find* cost. This is because if from any finder location, a track belonging to an object in any location in the network is to be found, then it must be the case that the *find* trajectory contains all points in the terminating set, thereby increasing the worst case *find* cost.

Multiple terminating sets each with one terminating point: If there are multiple terminating sets then it is sufficient for *find* to traverse the terminating point in any such set. In this case there is a likelihood of decreasing the maximum *find* cost when compared to having only set of terminating points because tracks can be *found* by reaching a terminating point in any of the terminating sets. The maximum *find* cost in the network depends on the size of the largest region. However, we show in Appendix A that to maintain *update* distance sensitivity, the size of the largest region

has to be at least $\Omega(N)$ in a $N \times N$ network. Also when the number of terminating sets is greater than 1, *update* has to traverse the terminating point in all terminating sets in the worst case. Thus the maximum *update* cost in the network increases.

Maintaining a track with respect to local terminating points could be advantageous if it is more likely that querying object and the object being found are closer. Thus, a *find* will never run into the scenario of having to traverse all regions in the network. Similarly, maintaining a track with respect to multiple terminating point sets could be advantageous if objects are likely to move within bounded regions within a network. In this paper we consider all distances between querying object and tracked object to be equally likely and do not restrict mobility of the objects. Hence we consider only the case where there is a unique terminating set with a single terminating point, namely C.

Different terminating point for each type of object: We note that it is also feasible to select a different terminating point for different *types* of objects. In this paper we describe how to maintain tracks for objects with respect to one terminating point and guarantee *find* and *update* distance sensitivity. A different terminating point can be chosen for each type of object based on hash functions and each *type* of finder object can choose the respective terminating points as a worst case landmark; but this concept is orthogonal to that of maintaining tracks with respect to a given terminating point and is therefore compatible with *Trail*.

We now summarize some of the performance aspects of *Trail* in terms of load balancing (fairness), maintenance and memory.

Load balancing and fairness: In hierarchy based solutions, to maintain tracks and to answer queries, tracks from terminating points necessarily pass through these

clusterheads, where as *Trail* avoids hierarchy by determining anchors for the tracking paths on-the-fly based on the motion of objects. By avoiding hierarchy, in *Trail* load is in fact more evenly distributed (more fairness) than hierarchical solutions in which queries have to answered by specific locations and the same nodes are taxed.

Maintenance: The tracks maintained in *Trail* are almost straight with a stretch factor of less then 1.2. By maintaining shorter tracks that are not convoluted, the cost of maintaining the tracks in *Trail* is actually lower. We have also discussed in detail the fault-tolerance actions of *Trail* in Section 4 where we describe how to handle failures of nodes along existing tracks, failures during *update* and $find$ and also how failures of terminating point C can be handled locally.

Tolerating failure of the terminating point: In Section 3.3.3 we have shown that in *Trail*, we can locally tolerate the failure of even C. Our protocol actions are such that when C fails or a set of nodes in a contiguous region around C fail, track for an object will terminate at any node that is closest to the boundary formed by the *hole*. Thus during a *find* operation, a redirection rule as in GPSR is bound to intersect the track and once again the faults are tolerated locally.

Memory efficiency: We note that the tracks maintained in *Trail* only contain pointers to the current location of an object and not the state information of the object. We also note that the storage required at each node is $O(log(N))$, per object, where the network is of size $N \times N$.

Handling bottleneck at C: We have previously discussed the option of having multiple terminating sets, each with one terminating point. One example of this is to construct a circle of constant radius δ around the center of the network and to define each point on this circle as a terminating set. Thus tracks from any point in

the network pass through all points on this circle. A *find* trajectory can intersect any point on this circle to obtain a pointer to the object. The maximum update cost increases by a factor of δ and the maximum *find* cost decreases by a factor of δ, and still maintaining distance sensitivity. The responsibility of handling queries is now distributed evenly around C.

3.3.7 Performance Evaluation

In this subsection, we evaluate the performance of *Trail* using simulations in *JProwler* [70]. The goals of our simulation are: (1) to study the effect of routing stretch and discretization errors on the trail stretch factor, (2) to study the effect of uniform node failures on the performance of *Trail*, (3) to compare the average costs for *find* and *update*, as opposed to the upper bounds we derived earlier, and (4) to analyze the performance of *Trail* when scaled in the number of objects. Our simulation involves a network of 8100 Mica2 motes arranged in a 90×90 grid.

Setup

Our simulation involves a 90×90 Mica2 mote network arranged on a grid. The center of the network is placed at one corner, thus essentially simulating one quadrant of the network. This setup lets us test the protocol over larger distances without update and find operations reaching the center. We use *JProwler* as our simulation platform with a Gaussian radio fading model. The mean interference range is a grid area of 5×5 square units. Packet transmission time is set at 40 ms. We implement geographic routing on a grid to route messages in the network. In the presence of failures we use a left hand rule to route around the failure [38]. We assume an

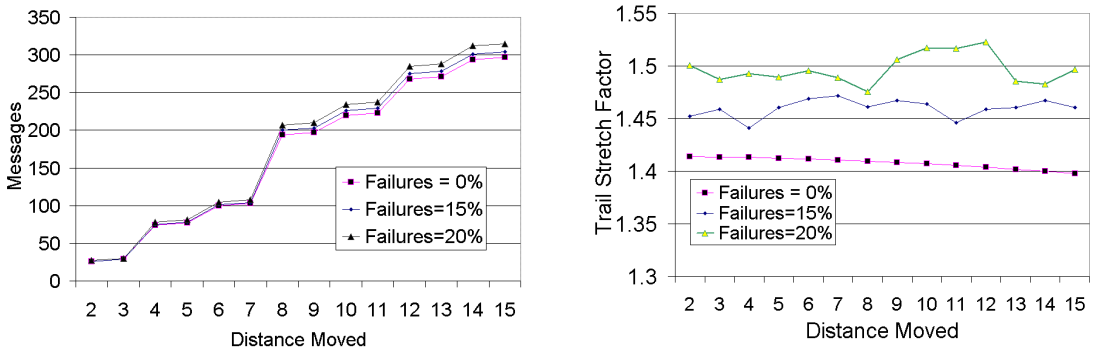
underlying link maintenance layer because of which the list of *up* neighbors is known at each node.

Performance of update operations

We determine the number of messages exchanged for object updates over different distances when an object moves continuously in the network. We consider the unit grid separation, where each node has at most 4 communication neighbors. The number of neighbors may be lesser due to failures. We calculate the amortized cost by moving an object in different directions and then observing the cumulative number of messages exchanged up to each distance from the original position to update the tracking structure. The results are shown in Fig. 3.23(a). The jumps visible at distances 4 and 8 show the impact of the $log(d)$ factor in the amortized cost. At these distances, the updates have to be propagated to a higher level. We also study the effect of uniform failures in the network on the increase in update costs. We consider fault percentages up to 20. We see from the figure that even with failures the average communication cost increases log linearly with distance. This indicates that the failures are handled locally.

Trail stretch factor

From Section 3.3.2, we note that in the continuous model, for an object P at distance d_{pC} from C, length of $trail_P$ is less than $1.2 * d_{pC}$. We now study the effect of routing overhead and the discretization factor on the length of the tracking structure that is created. We measure the trail length in terms of the number of hops along the structure. Fig. 3.23(b) shows the average ratio of distance from C to the



(a) Trail update cost (amortized)

(b) Trail stretch factor

Figure 3.23: Trail update costs and Trail stretch factor

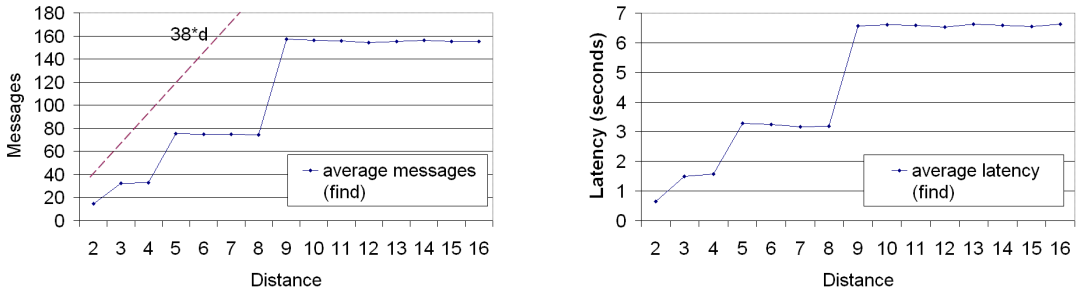
length of the trail during updates over different distances from the original position. The parameter $b = 1$ in these simulations.

When the trail is first created, the trail stretch factor is equal to the routing stretch factor from C to the original location. In the absence of failures, we notice that the trail stretch factor is around 1.4 at updates of smaller distances and then starts decreasing. This can be explained by the fact the trail for an object starts bending more uniformly when the update is over a large distance. Even in the presence of failures, the trail stretch factor increases to only about 1.6 times the actual distance.

Performance of *find*

We first compare the average *find* costs of *Trail* with upper bounds derived. We study this in the presence of no interference, i.e. there is only one finder in the network. We fix the finder at distance 40 units from C. We vary the distance of object being found from 2 to 16. We evaluate using the basic *find* algorithm with

$b = 1$. In Fig. 3.24(a) and Fig. 3.24(b), we show the average number of messages and the average latency for the *find* operation respectively.



(a) Average messages (find)

(b) Average latency (find)

Figure 3.24: Trail: find cost

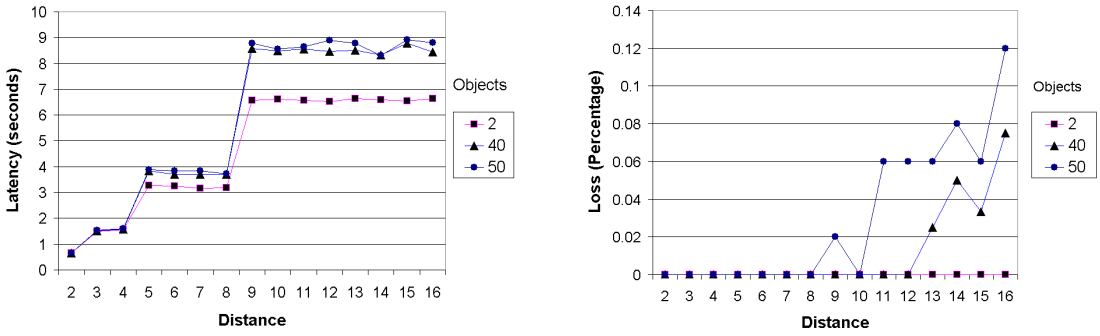
The analytical upper bound $38 * d$ (obtained from Lemma 3.3.8 for $b = 1$) is indicated using dotted lines in Fig. 3.24(a), and we see that the number of messages exchanged during *find* operations are significantly lower. When there is only one finder in the network, there is no interference. Therefore there are no re-transmissions except for the case when there is a loss due to probabilistic fading. Therefore the latencies are roughly equal to the number of messages times the message transmission time per hop. The jumps at distances 3, 5 and 9 are due to increase in levels of exploration at these distances. The results of the above simulations thus validate our theoretical bounds derived in Section 3 and 4.

Impact of interference

We now evaluate the effect of interference when multiple objects are present in the network. Note that *Trail* operates in a model where queries are generated in

an asynchronous manner. We first evaluate the effect on *find* latency when objects are uniformly distributed across the network. We then evaluate the performance in a more severe environment where all the objects being found are collocated in the network.

In the presence of interference, messages are likely to be lost and we have implemented the following reliability mechanism to counter that. Forwarding a *find* message by the next hop is used as an implicit acknowledgment. The *find* messages are retransmitted up to 4 times by each node. The interval to wait for an acknowledgment is doubled after every retransmission starting with 100 ms for the first retransmission. Note that 100 ms is a little more than twice the transmission time for each message. Also upon sensing traffic, the MAC layer randomly backs off the transmission within a window of 10 ms. The maximum number of MAC retries are set to 3.

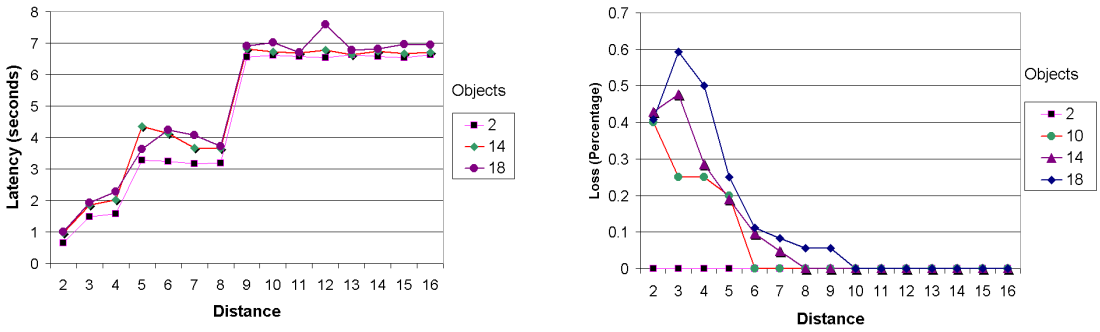


(a) Average latency (find)

(b) Loss percentage (find)

Figure 3.25: Effect of interference on find cost

In the first scenario we uniformly distribute the objects in a 50×50 area in the center of the quadrant being simulated. By distributing the objects in the middle of the quadrant being simulated, we avoid the decrease in *find* messages simply because an object is close to the boundary. We simulate 2, 10, 20, 30, 40 and 50 objects in the network. We observe no significant increase in latency from 2 objects up to 30 objects. For the case of 40 and 50 objects in the network we observe increase in average latency especially at larger distances. At larger distances, *find* messages from different objects interfere to a significant extent. Despite messages being retransmitted up to 4 times, we also see losses during the find operation at distances greater than 12 units. The latency and loss percentages at different distances for 2, 40 and 50 objects are shown in Fig. 3.25(a) and Fig. 3.25(b) respectively.



(a) Average latency (find)

(b) Loss percentage (find)

Figure 3.26: Effect of interference on find cost: objects being found collocated

We now consider a more severe scenario where all the objects being found are at the same location. We compute the average latency for the find operation when objects issuing *find* query are uniformly distributed around this location, at different

distances. Fig. 3.26(a) shows the latency with respect to distances for 2, 14 and 18 objects when all objects being found are at the same location. Fig. 3.26(b) shows the loss percentages for 2, 10, 14 and 18 objects when all objects being found are at the same location. As expected, we see from Fig. 3.26(a) and Fig. 3.26(b) that interference is severe at smaller distances. We see loss percentages as high as 60% when there are 18 objects at small distances.

Summary of evaluation

We observe from the above figures that *Trail* has a *find* time that grows linearly with distance. When scaled in the number of objects up to 50, with objects uniformly distributed in a 50×50 area and concurrently issuing queries, the query response time still does not increase substantially. However at a scale of 40 objects and distances of greater than 12 units we observe losses of around 10% during the find operation. This is because, at larger distances *find* messages from different objects interfere to a significant extent. In a potentially more severe scenario where all objects being found are at the same location and objects issuing *find* are distributed uniformly around that location, interference is significant at smaller distances. We see loss percentages as high as 60% when there are 9 pairs of objects at small distances.

3.4 Implementation of Trail in a Real Network

We have implemented *Trail* for the special case of a long linear topology network for demonstrating an intruder interceptor tracking application. We have experimentally validated the performance of this version of *Trail* on 105 Mica2 motes in the Kansei testbed [3] under different scaling factors such as the number of objects in the system, the frequency of queries, and the speed of the objects in the network. This

implementation has been used to support a distributed intruder interceptor tracking application where the goal of the interceptor is to catch the intruders as far away from an asset as possible. This application was demonstrated at Richmond Field Station in Berkeley in 2005 as part of the DARPA NEST Program. In this section, we describe the results of these experiments.

3.4.1 Experimental setup

We use a network of 105 XSM-Stargate pairs in a 15×7 grid topology with 3 ft spacing in the Kansei testbed. The XSMs are a Mica2 family of nodes with the same Chipcon radio but with additional sensors mounted on the sensor board. The XSMs are connected to Stargate via serial port and the Stargates are connected via Ethernet in a star topology to a central PC. We are able to adjust the communication range by adjusting the power level and the XSMs can communicate reliably up to 6 ft at the lowest power level but the interference range could be higher. *Trail* operates asynchronously with no scheduling to prevent collisions. Hence, we implement an implicit acknowledgment mechanism at the communication layer for per-hop reliability. The forwarding of a message acts as acknowledgment for the sender. If an acknowledgment is not received, then messages are retransmitted up to 3 times.

Object traces

We now describe how the object motion traces are obtained. Motes were deployed in a grid topology with 10 m spacing at Richmond Field Station. Sensor traces were collected for objects moving through this network at different orientations. Based on these traces, tracks for the objects are formed using a technique described in [2]. These tracks are of the form (timestamp, location) on a 140m $\times$ 60m network. These

object tracks are then converted to tuples of the form (id, timestamp, location, grid position) where grid position is the node closest to the actual location on the 15×7 network and id is a unique identifier for each object. These detections are injected into the XSM in the testbed corresponding to the grid position via the Stargate at the appropriate time, using the injector framework in *Kansei*. Thus, using real object traces collected from the field and using the injector framework, we emulate the object detection and association layer to evaluate the performance of our network service.

3.4.2 Results

We evaluate the performance of *Trail* under different scaling factors such as increasing number of objects, higher speed of objects and higher query frequency in terms of the reliability and latency of the service. We run *Trail* with 2, 4, 6 and 10 mobile objects, in pairs. One object in each pair is the object issuing *find* query and the other object is the object being located. In each of these scenarios, we consider query frequency of 1 Hz, 0.5 Hz, 0.33 Hz and 0.25 Hz. The object speed affects the operation of *Trail* in terms of the rate at which *grow* and *clear* messages are generated. We consider 3 different object update rates, one in which objects generate an update every 1 second, every 2 seconds and every 3 seconds. Considering that the object traces were collected with humans walking across the network acting as objects with average speed of about 1 m/s, object update rates of 1 Hz and 0.5 Hz enable a tracking accuracy of 1m and 2m respectively.

In the 4, 6 and 10 objects scenario, we consider a likely worst case distribution of the objects where all objects issuing *find* and all objects being found are within communication range. Moreover, as optimal pursuit control requirements suggest

[15], the query frequencies depend on relative locations and are lesser when objects are far apart, but we consider all objects issuing queries at the same frequency. If the replies are not received before the query period elapsed, then the message is considered lost. The loss percentages are based on 100 *find* queries at every distance and the latencies are averaged over that many readings.

Scaling in number of objects

Fig. 3.27 shows the latency and loss for *find* operations as the number of objects increases with query frequency fixed at 0.33 Hz and object updates fixed at 0.5 Hz. Fig. 3.28 shows the latency and loss for *find* operations as the number of objects increases with query frequency fixed at 0.5 Hz and object updates fixed at 0.5 Hz.

Scaling in query frequency

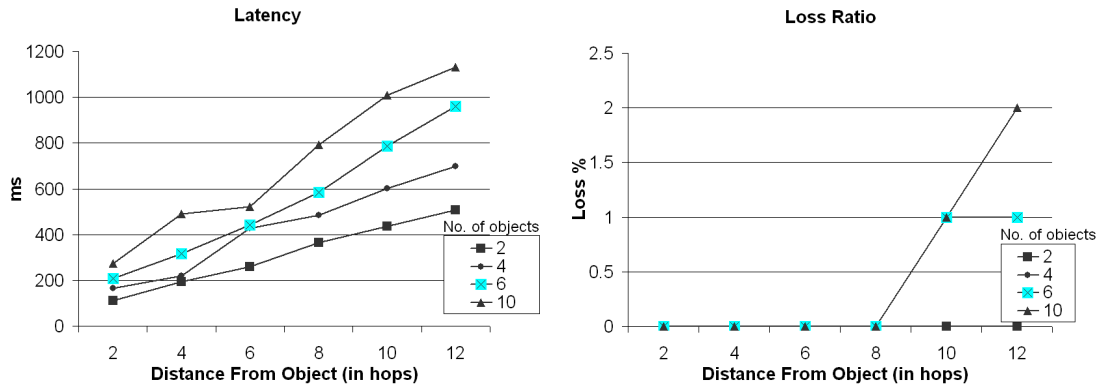
Here we analyze how the latency and reliability of *Trail* are affected as the query frequency increases. In Fig. 3.29, we show the reliability and latency of *Trail* with 6 objects under query frequencies of 1 Hz, 0.5 Hz, 0.33 Hz and 0.25 Hz, with object update rate of 0.5 Hz.

Scaling in object speed

Fig. 3.30 shows the latency and loss for *find* operations with increasing object speeds that generate updates at 0.33 Hz, 0.5 Hz and 1 Hz. The query frequency is 0.5 Hz and the number of objects is 6.

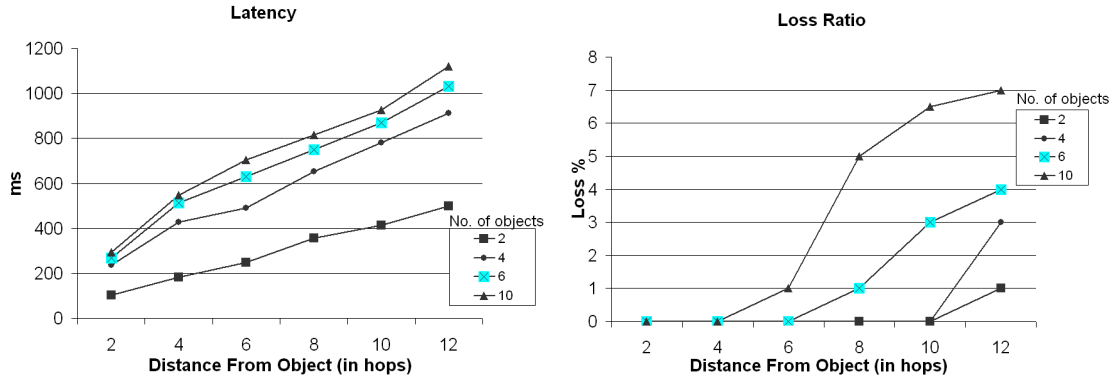
Summary of experimental evaluation

We observe from the above figures that *Trail* offers a query response time that grows linearly with the distance from an object. *Trail* operates in an environment



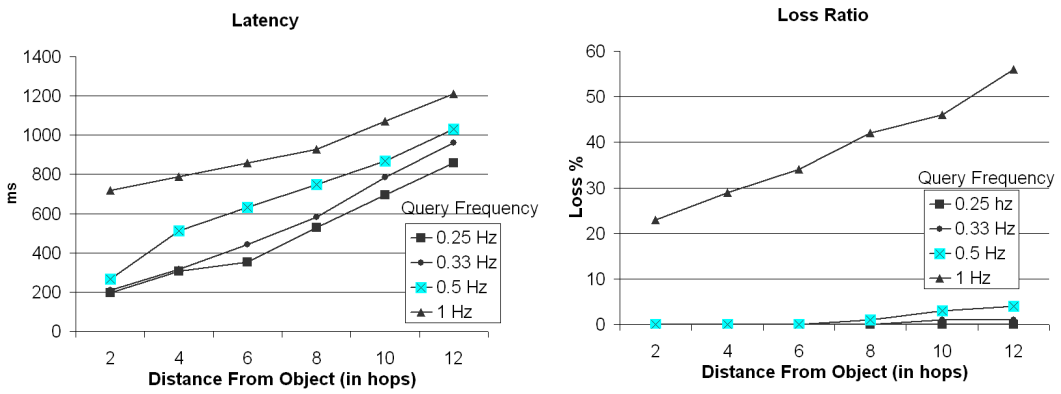
(a) Latency of *Trail* (b) Loss ratio

Figure 3.27: Scaling in number of objects (query frequency 0.33 Hz, object update 0.5 Hz)



(a) Latency of *Trail* (b) Loss ratio

Figure 3.28: Scaling in number of objects (query frequency 0.5 Hz, object update 0.5 Hz)



(a) Latency of *Trail*

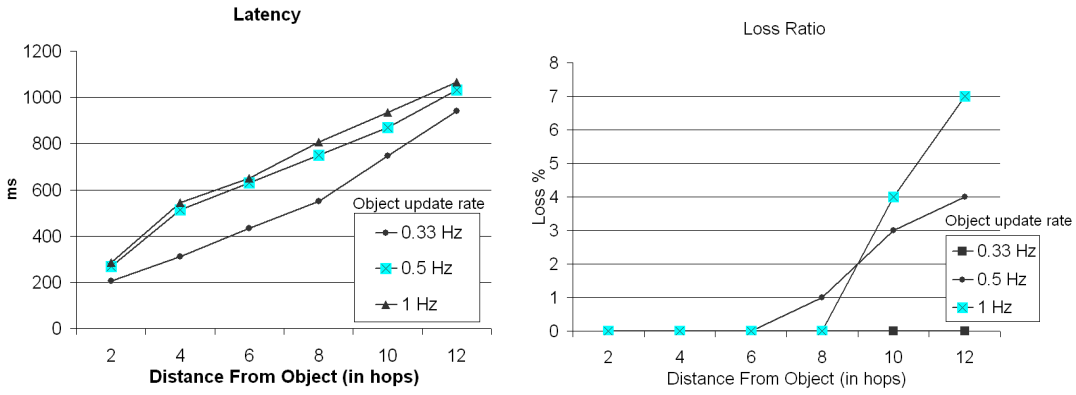
(b) Loss ratio

Figure 3.29: Scaling in query frequency (6 objects, object update rate 0.5 Hz)

where objects can generate updates and queries asynchronously and in such an environment, interference increases the response time. From our experiments, we observe that query latency and loss percentages increase with number of objects and speed of objects but the loss ratio is not severe. As seen in Fig. 3.27, scaling the number of objects up to 10 yields a loss rate of up to 7% with a query frequency 0.5 Hz and an object update rate of 0.5 Hz. As seen in Fig. 3.30, scaling the object speeds to generating 1 update per second results in a loss rate of up to 7 % even with 6 objects in the system and query frequency of 0.5 Hz. Increasing query frequencies has a more severe impact on loss percentages especially with more objects in the network. In Fig. 3.29, we notice that with 6 objects in the network, loss increases substantially as the query frequency becomes 1 Hz; this happens due to higher interference leading to congestion.

Handling interference: To tolerate network interference, spatial and temporal correlations that exist in the application can be exploited in the following way. The

rate at which information is needed by the pursuer is known. The network service can be notified of the next instant at which the state of the evader is needed and the location where the query results need to be sent. The query results can then arrive *just in time*. Advance knowledge about the query and the deadline can be used to decrease the interference in the network when multiple pursuers query about evaders in the network and more so when the objects are densely located. Another possibility to deal with interference is a synchronous *push* version of the network tracking service where snapshots of objects are published to subscribers in a distance sensitive manner thereby avoiding interference. By the same token, applications should be aware of other extreme conditions (in terms of object number and speed) for effectively using the service. For example, applications may compensate for losses by increasing their query frequency, but this should account for extreme scenarios where the increased frequency itself results in higher interference.



(a) Latency of *Trail*

(b) Loss ratio

Figure 3.30: Scaling in object speed (6 objects, query frequency 0.5 Hz)

3.5 Related Work

In this section, we discuss related work and also compare the performance of *Trail* with other protocols designed for distance sensitive tracking and querying.

Tracking: As mentioned earlier, mobile object tracking has received significant attention [5, 24, 30] and we have focused our attention on WSN support for tracking. Some network tracking services [25] have non-local updates, where update cost to a tracking structure may depend on the network size rather than distance moved. There are also solutions such as [5, 24, 1] that provide distance sensitive updates and location.

Locality Aware Location Services *(LLS)* [1] is a distance sensitive location service designed for mobile ad-hoc networks. In LLS, the network is partitioned into hierarchies and object information is published in a spiral structure at well known locations around the object, thus resulting in larger update costs whenever an object moves. The upper bound on the update cost in LLS is $128 * d_m * log(d_m)$, where d_m is the distance an object moves, as opposed to the $14 * d_m * log(d_m)$ cost in *Trail*; the upper bounds on the *find* cost are almost equal. Moreover, as seen in Section 5, we can further reduce the upper bound on the *find* cost at higher levels in *Trail*.

The Stalk protocol [24] uses hierarchical partitioning of the network to track objects in a distance sensitive manner. The hierarchical partitioning can be created with different dilation factors ($r \geq 3$). For $r = 3$ and 8 neighbors at each level, at almost equal *find* costs, *Stalk* has an upper bound update cost of $96 * d * log(d)$. This increase occurs because of having to query neighbors at increasing levels of the partition in order to establish *lateral* links for distance sensitivity [24].

Both *Stalk* and *LLS* use a partitioning of the network into hierarchical clusters which can be complex to implement in a WSN, whereas *Trail* is cluster-free. Moreover, in *Stalk*, the length of the tracking structure can span the entire network as the object keeps moving and, in *LLS*, the information about each object is published in a spiral structure across the network. In comparison, *Trail* maintains a tighter tracking structure (i.e., with more direct paths to the center) and is thus more efficient and locally fault-tolerant.

	Update Cost	Find Cost	Size
Trail	14*d*logd	38*d	1.2*d_c
LLS	128*d*logd	36*d	16*D
Stalk	96*d*logd	39*d	3*D
Awerbuch Peleg	O(d*logd*logN)	16*d*log(2N)	4*D

D – n/w Diameter ; N – No. of Nodes ; d_c – Distance from C

Figure 3.31: Trail: analytical comparison

In [5], a hierarchy of regional directories is constructed and the communication cost of a find for an object d_f away is $O(d_f * log(N))$ and that of a move of distance d_m is $O(d_m * log(D) * log(N))$ (where N is the number of nodes and D is the network diameter). A topology change, such as a node failure, however, necessitates a global reset of the system since the regional directories depend on a non-local clustering program that constructs sparse covers.

Querying and storage: Querying for events of interest in WSNs has also received significant attention [36, 63, 52] and some of them focus on distance sensitive querying. We note that *Trail*, specifically the *find-centric* approach can also be used in the context of static events.

In [52], a balanced push-pull strategy is proposed that depends on the query frequency and event frequency; given a required query cost, the advertise operation is tuned to do as much work as required to satisfy the querying cost. In contrast, *Trail* assumes that query rates depend on each subscriber (and potentially on the relative locations of the publisher and subscriber), and it also provides distance sensitivity during find and move operations, which is not a goal of [52]. In directed diffusion [36], a tree of paths is created from all objects of interest to the tracker. All these paths are updated when any of the objects move. Also, a controller initiated change in assignment would require changing the paths. By way of contrast, in *Trail*, we impose a fixed tracking structure, and tracks to all objects are rooted at one point. Thus, updates to the structure are local. Rumor routing [14] is a probabilistic algorithm to provide query times proportional to distance; the goal of this work is not to prove a deterministic upper bound. Moreover, its algorithm does not describe how to update existing tracks locally and yet retain distance sensitive query time when objects move.

Geographic Hash tables [63] is a lightweight solution for the in-network-querying problem of static events. The basic *GHT* is not distance sensitive since it can hash the event information to a broker that is far away from a subscriber. The distance sensitivity problem of GHT can be alleviated to an extent by using geographically bounded hash functions at increasing levels of a hierarchical partitioning as used in DIFS protocol. Still, attempting such a solution suffers from a multi-level partitioning problem: a query event pair nearby in the network might be arbitrarily far away in the hierarchy. However, we do note that *GHT* provides load balancing across the network, especially when the types of events are known and this is not the goal of *Trail*.

Distance Sensitive Information Brokerage [27] protocol performs a hierarchical clustering of the network and information about an event is published to neighboring clusters at each level. Using *Find-centric Trail* we can query information about static events in a distance sensitive manner. We also note that when events are static, the optimal publish structure is much smaller than publishing along circular tracks. We have studied optimal publish structures for querying in a static context in a related work [23].

Spatio-temporal query services Motivated by a class of applications in which mobile users are interested in continuously gathering information in real time from their vicinities, a network data service called *spatio temporal query* has been proposed in [53]. The spatial constraint for the network service comes from an energy efficiency point of view; only nodes relevant to a query area should be involved in contributing the query result. The temporal constraint is due to a requirement on data freshness for the query results. An approximate motion model for the mobile user is assumed. Specifically the motion can be predicted over a small interval of time. The query area at any time is a function of the current location of the mobile user. The key difference in *Trail* is that the query area in consideration is the entire network as opposed to a function of the querier location as in [53].

We note that, when *Trail* is used in the context of a distributed pursuer evader game, spatial and temporal correlations that exist in the application can be exploited using ideas presented in [53] to improve the performance of the application and the network. The rate at which information is needed by the pursuer is known. The network service can be notified of the next instant at which the state of the evader is needed and the location where the query results need to be sent. The query results

can then arrive *just in time*. Constraints on evader speed can be exploited as follows. Subsequent queries for evader location can originate from the previous evader location and the results can be routed back to the pursuer.

3.6 Summary

In this chapter, we considered a pursuer-evader game known in the literature as "asset protection", and formulated optimal strategies under communication constraints, established bounds on the information requirements of these strategies, and derived scaling laws for these bounds. In particular, we showed that the min-max optimal pursuer strategy of the full information game extends to networked games, provided that the sampling period and the delay for the evader state information updates scale linearly with the pursuer-evader distance. We proposed a novel min-max equilibrium concept for networked differential games by introducing an omniscient opponent which can maximally exploit the delays and the inter-sample periods in the information state updates.

We then presented *Trail*, a family of protocols for distance sensitive distributed object tracking in WSNs. *Trail* avoids the need for hierarchical partitioning by determining anchors for the tracking paths on-the-fly, and is more efficient than other hierarchy based solutions for tracking objects: it allows 7 times lower updates costs at almost equal *find* costs and can tolerate faults more locally as well.

Importantly, *Trail* maintains tracks from object locations to only one terminating point, the center of the network. Moreover, since its tracks are *almost* straight to the center with a stretch factor close to 1, *Trail* tends to achieve the lower bound on the total track length. By using a tight tracking structure, *Trail* is also able to decrease

the upper bound *find* costs at larger distances and thereby decrease the average find cost across the network.

Trail is a family of protocols and we have shown that refinements of the basic *Trail* protocol are well suited for different network sizes and query frequency settings. We have validated the distance sensitivity and fault tolerance properties of *Trail* in a simulation of 90×90 network using *JProwler*. We have also successfully implemented and tested the *Trail* protocol in the context of a pursuer evader application for a medium size (over 100 node) mote network.

Trail operates in an environment where objects can generate updates and queries asynchronously. We note that in such an environment, due to the occurrence of collisions, there can be an increase in the message complexity for querying and updates especially when the objects are densely located in the network. As future work, we are considering a *push* version of the network tracking service where snapshots of objects are published to subscribers in a distance sensitive manner, both in time and information, in order to increase the reliability and energy efficiency of the service when the density of objects in the network is high.

We also note that spatial and temporal correlations that exist in the application can be exploited to improve the performance of the application and the network. Two such examples are as follows. (1) The rate at which information is needed by the pursuer is known. The network service can be notified of the next instant at which the state of the evader is needed and the location where the query results need to be sent. The query results can then arrive *just in time*. Advance knowledge about the query and the deadline can be used to decrease the interference in the network when multiple pursuers query about evaders in the network and more so when the objects

are densely located. (2) Constraints on evader speed can be exploited as follows. Subsequent queries for evader location can originate from the previous evader location and the results can be routed back to the pursuer. Thus the energy efficiency of the network can be improved.

CHAPTER 4

CLASSIFICATION OF INTRUDERS AND TRACK MONITORING

In this chapter, we study the application of sensor networks to the problems of classifying and monitoring tracks of targets. We consider a surveillance application scenario, whose objective is to identify a breach along a perimeter or within a region. The intruding object, or target, may be an unarmed person, a soldier carrying a ferrous weapon, or a vehicle. The three fundamental user requirements of this application are target detection, classification, and tracking.

Detection requires that the system discriminate between a target's absence and presence. Successful detection requires a node to correctly estimate a target's presence while avoiding false detections in which no targets are present. The key performance metrics for detection include the probability of correct detection, and the probability of false alarm.

Classification requires that the target type be identified as belonging to one of several classes including person, soldier, and vehicle. More generally, classification is the result of M-ary hypothesis testing and depends on estimation, which is the process of determining relevant parameters of the detected signal including, for example, its peak amplitude, phase, duration, power spectral density, etc. Successful classification

requires that targets are labeled by the system as being members of the class to which they actually belong.

Tracking involves maintaining the target's position as it evolves over time due to its motion in a region covered by the sensor network's field of view. Successful tracking requires that the system estimate a target's initial point of entry and current position with modest accuracy and within the allowable detection latency. Implicit in this requirement is the need for target localization. The tracking performance requirements dictate that tracking accuracy, or the maximum difference between a target's actual and estimated position, be both bounded and specified, within limits, by the user. The system is not required to predict the target's future position based on its past or present position.

We design a dense, distributed, and 2-dimensional sensor network-based classification and tracking system using inexpensive sensor nodes. In this model, intrusion data are processed locally at each node, shared with neighboring nodes if an anomaly is detected, and communicated to an ex-filtration gateway with wide area networking capability. The motivation for this approach comes from the spatial- and temporal-locality of environmental perturbations during intrusions, suggesting a distributed approach that allows individual sensor nodes, or clusters of nodes, to perform localized processing, filtering, and triggering functions. Collaborative signal processing enables the system to simultaneously achieve better sensitivity and noise rejection, by averaging across time and space, than is possible with an individual node which averages only across time.

Our approach thus demonstrates how dense, resource-constrained sensor networks yield improved spatial fidelity of sampling the environment. More specifically, we

introduce a spatial statistic called the influence field, realize an estimator for it using a binary sensor field, and use it as the basis for a new type of classifier and tracker. The influence field of an object thus depends on the type(s) of sensors being used for detection (magnetometers in Fig. 4.1), and it is essentially characterized by the area and the shape of the region. Figure 4.1 illustrates the differences between magnetic influence fields for two objects, a person carrying a metal rod and a vehicle. The size and shape of the influence fields shown in this figure depend on the amount and distribution of metallic content in each object type and the orientation of the object (e.g., the dumbbell shape of the vehicle influence field is attributed to the positions of its axles). Each sensor node merely has to detect a binary "presence" of an object; network-based aggregation of these bits yields the influence field without requiring substantial or complex node operation.

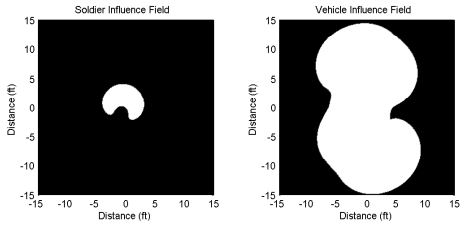


Figure 4.1: Magnetometer based influence fields for two object types.

The key challenge in realizing the influence field is the unreliability of wireless sensor networks. Event loss –both in nodes and in the network– is fundamental to wireless sensor node platforms and its impact on the application can be substantial. Thus both node and network unreliability have to be dealt with in estimating the influence field.

In Section 4.1, we state our system model, describe our approach for classification and tracking by estimation of influence fields. In Section 4.2, we derive necessary conditions for reliably estimating the area and shape of various objects' influence field in a manner that preserves their difference despite faulty detection, network level fading and channel contention. We then design network parameters and algorithms to meet these conditions. In Section 4.3, we describe how our results and techniques can be composed and applied to identify and track objects. We have demonstrated our influence field based classification and tracking approach in the context of 2 sensor network based surveillance applications namely, *A Line in the Sand* (18m by 7m area and 100 nodes), and *ExScal* (1Km by 200m area and over 1000 nodes). In Section 4.4, we provide some details of the implementation of our techniques in the context of a surveillance application demonstration, *A Line in the Sand*, and show how we dealt with different fault classes in our design. In Section 4.5, we describe extensions to our approach using multiple sensing modalities. In Sec. 4.6, we describe previous work related to ours. In Section 4.7, we present a summary.

4.1 Influence Field Based Classification and Tracking

In this section, we state the system and fault model and describe our solution for classification and tracking by estimation of influence fields. We then identify the challenges in reliably estimating the influence field.

4.1.1 System Model

The system consists of N wireless sensor nodes, each with a unique identifier. We assume a localization service that provides the relative or absolute position for each node and a global time synchronization service that enables each node to timestamp

its detections. The sensor nodes are distributed uniformly over a geographic region that is to be monitored. We model this region as a large finite number, Ω, of perfectly spaced logical points that serve as the potential locations where nodes can be deployed. When we refer to the area A of a subregion, we mean A is the number of these Ω points in the subregion. We denote the ratio N/Ω by ρ, which represents the sensor density in the network.

We assume that the wireless network is connected, hence it is possible to aggregate messages from any subset of the nodes in the system, albeit that such aggregation may require multi-hop communications. If this is the case, we assume a central aggregator node that is known to the rest of the nodes.

Recall that the *influence field* of an object j with respect to a given sensing modality is the region surrounding j where j will be "detected" by a sensor of that type located at any point in that region. For simplicity of presentation, we assume that the size of the influence field of j is invariant with respect to its location. Likewise, the shape of the influence field is invariant with respect to location, up to rotation. We limit our attention to one given sensing modality for simplicity. The modality will remain implicit in our notation.

4.1.2 Fault Model

Sensor networks are subject to a large class of faults, resulting from inexpensive hardware, limited resources, unreliable communications and extreme environmental conditions. We consider both node and network fault types.

Node faults

Sensor nodes fail in a variety of ways, including hardware and software failures, or simply in the form of a transient event loss. They also occasionally generate false positives due to unreliable hardware, environmental perturbations and transient state corruption. Thus at any time, the net effect of a node fault can be modeled as missing the detection of an object, i.e., a false negative, or asserting a detection when there is no object, i.e., a false positive.

> **Node fault model:** The probability that any node misses the detection of an object, whether it is due to a transient, permanent or intermittent node fault, is $1 - p_n$, while the probability that any node generates a false positive is p_{fp}.

In other words, we assume that false negatives and false positives at nodes are independent of each other and of the objects. It follows trivially that for any meaningful detection to be possible, the probability of a node detecting an event, p_n, must be greater than the probability of a node generating a false positive, p_{fp}.

Network faults

Wireless communications in a multi-hop network are subject to both fading and contention effects. Channel fading loss depends on link characteristics such as distance, relative orientation of sender and receiver, environmental conditions, etc.

> **Fading model:** The probability of message loss due to fading for any single hop communication in the network is $1 - p_f$. It follows that in the absence of any other faults, the probability of message loss due to fading for any h hop communication is $1 - p_f^h$.

Channel contention losses occur when multiple senders try to transmit a message at the same time. The degree of message loss depends on several factors like the

type of Medium Access Control (MAC) protocol used, the inherent synchronicity of message transmissions, and most importantly the number of nodes trying to send data simultaneously. We assume a standard CSMA/CA MAC protocol wherein nodes try to avoid collisions using random backoffs and channel sensing.

> **Contention model:** The probability of message contention loss, which is a function of the number of nodes simultaneously trying to send a message and of the number of slots available for a node to choose its random backoff from, is $1 - p_c$.

The end-to-end network reliability of message reception for a given source depends on several factors including the size of the traffic load, the number of hops traversed by each message and the routing protocol. We assume a convergecast routing model, wherein each message is forwarded by a node to its next hop neighbor until it reaches the aggregator.

> **End-to-end reliability model:** The probability of end-to-end message loss for a traffic source i, which is a function of the number of senders in i, and the distance or number of hops h between the traffic source and the aggregator, is $1 - p_{rcv_h^i}$.

Our model postulates that network unreliability is uniform across nodes equidistant from the aggregator. It is possible to realize the aforementioned end-to-end reliability model by careful design of the routing protocol used. The GridRouting protocol, achieves this by uniformly balancing the trac load over paths including stable, reliable links.

4.1.3 Estimating the influence field

Estimation consists of calculating the area and the shape of the influence field. Recall that the distribution of sensor nodes is uniform. If we assume sensor node density ρ exceeds some lower bound that depends on the targets at hand, the estimation

of the area A of the influence field of j is effectively reduced to counting the number of nodes that detect j. With uniform distribution, the number of sensors in any region of area A follows a binomial distribution with parameters (A, ρ). For practical values of A and ρ, we can exploit the rule of thumb for the normal approximation to this binomial distribution that the value of the random variable lies within 3 times the standard deviation of the expected value in 99% of the trials.

Proposition 1 Given uniform deployment of sensors, with high probability (whp), the number of sensors that lie in the influence field of j is in the interval

$$(A \times \rho) \; - \; 3 \times \sqrt{A \times \rho \times (1-\rho)} \quad .. \quad (A \times \rho) \; + \; 3 \times \sqrt{A \times \rho \times (1-\rho)}$$

Relation to sensor coverage. Sensor coverage in a region is the *minimum* number of sensors that "cover" (i.e., will detect) each point in the region. While this typical definition of the concept is independent of the type of objects at hand, a more useful definition for our purposes would be one that is with respect to each object type. (Thus, the sensor coverage of object j in a region may be different from the sensor coverage of another object in that region.)

Classification is an example of an application that can exploit area estimation. Different object types may be classified via separation between the areas of their respective influence fields. Errors in area estimates can thus result in mis-classifications.

Tracking is an example of an application that can exploit shape estimation. Object location may be tracked from the locations of the sensors that detect it, i.e., by computing the centroid of their locations. Shape distortion errors can thus result in inaccuracy of tracking. (The same argument would apply for classification based on the shape of the influence field.)

Node and network faults impact estimation of the area and the shape of the influence field of j. In particular, the impact may not be simply proportional to the area and the distribution of the sensors whose detections are successfully aggregated may no longer be uniform.

We are therefore led to the problem of how to reliably estimate the size and shape of the influence fields of objects so that they can still be compared. More specifically, we focus on two subproblems of reliable estimation:

1. How to ensure that for objects whose respective influence field areas are separable, the separation remains between the fault-affected estimates of their respective influence field areas?

2. How to preserve the shape of the influence field of an object in the fault-affected estimate of the object influence field, by preserving the uniformity of distribution of the sensors whose detections are not affected?

4.2 Reliable Estimation Despite Faults

In this section, we address problems 1 and 2 outlined above successively for each of the fault models discussed. Specifically, for each fault model,

- we analytically identify necessary conditions for accurately preserving the separation between area estimates and the shape estimates,

- we derive an ideal density for node deployment with respect to the given sensing modality for the efficiency of estimation, and

- we characterize algorithmic techniques and parameter tuning options to deal with situations where the ideal density is not achievable, including cases where the deployed density is less than and more than ideal.

Also, where appropriate, we provide experimental corroboration of our analysis and algorithmic techniques via experiments realized on Mica2 sensor motes. Our approach is compositional and thus the analysis, which is presented separately for each fault type, can build upon the constraints/distributions identified for other fault types.

4.2.1 Tolerating false reports

We model the net effect of node faults is modeled as false negatives and false positives.

False Negatives

Let A_1, A_2, ... A_k be the influence fields of k types of objects ranging from the smallest to the largest. Recall that p_n is the probability that a sensor node detects an object. The number of non-faulty nodes in a region of area A_i thus has a binomial distribution with parameters (n_i, p_n), where n_i is the number of nodes in the area A_i. However, recall from Proposition 1 that n_i itself is a random variable that has a binomial distribution, characterized by the following equation.

$$E(n_i) = \rho \times A_i \tag{4.1}$$

$$V(n_i) = A_i \times \rho \times (1 - \rho) \tag{4.2}$$

The mean and variance of the number of nodes detecting object i, $E(d_i)$ and $V(d_i)$ respectively are given by the following equations.

$$E(d_i) = \rho \times A_i \times p_n \tag{4.3}$$

$$V(d_i) = A_i \times \rho \times p_n \times (1-\rho \times p_n) \tag{4.4}$$

For this distribution, for variously chosen values of A_i,ρ and p_n, we heuristically observe that in 99% of the trials, the value of the random variable lies within three standard deviations of the mean.

In order for separation between estimated influence fields of object types to be maintained whp, we require the following inequality to hold for each pair $(i, i+1)$ of objects.

$$E(d_{i+1})-(3\times\sqrt{V(d_{i+1})}) > E(d_i)+(3\times\sqrt{(V(d_i)}) \tag{4.5}$$

Let $\rho_{fn_{i(i+1)}}$ be the minimum density required to distinguish between objects i and $i+1$ whp. By solving Eqns. 4.3, 4.4, and 4.5, we obtain $\rho_{fn_{i(i+1)}}$ as:

$$\rho_{fn_{i(i+1)}} = \frac{B}{(p_n+B\times p_n)}$$

where

$$B = 9 \times \frac{(\sqrt{A_{(i+1)}}+\sqrt{A_i})^2}{(A_{(i+1)}-A_i)^2}$$

Theorem 4.2.1. *The minimum network density, hatρ_{fn}, required to maintain separation between the estimated influence fields of all objects in the presence of false negatives is the maximum of all pairwise densities $\rho_{fn_{i(i+1)}}$.* □

False positives

We now study the impact of false positives on estimation. In order for separation to be maintained between the estimated influence fields of objects i and $i+1$, we require that the number of detections in area A_i, when combined with the false positives in the region of area $A_{i+1}-A_i$, should be lower than the number of detections in area A_{i+1}. The mean and variance of the number of false positives in the area $A_{i+1}-A_i$, respectively $E(fp_i)$ and $V(fp_i)$, are as follows.

$$E(fp_i) = \rho_{i(i+1)} \times (A_{i+1} - A_i) \times p_{fp} \tag{4.6}$$

$$V(fp_i) = E(fp_i) \times (1-\rho_{i(i+1)} \times p_{fp}) \tag{4.7}$$

We thus require the following inequality to hold in order to maintain separation between the estimated influence fields of two objects A_i and A_{i+1}.

$$E(n_i)+(3\times\sqrt{(V(n_i)})+E(fp_i)+(3\times\sqrt{V(fp_i)}) < E(n_{i+1})-(3\times\sqrt{V(n_{i+1})}) \tag{4.8}$$

Note that the term on the right hand side does not contain any expression for false positives. This is because false positives outside the area A_{i+1} would affect the estimation of both objects equally. Let $\rho_{fp_{i(i+1)}}$ be the minimum density necessary to distinguish between objects i and $i+1$, obtained by solving Eqns. 4.6, 4.7 and 4.8.

Theorem 4.2.2. *The minimum network density, $\hat{\rho_{fp}}$, required to maintain separation between the estimated influence fields of all objects in the presence of false positives is the maximum of all pairwise densities $\rho_{fp_{i(i+1)}}$.* □

Density compensation techniques

The conditions derived above state the minimum density required for preserving separation between the estimated influence fields of objects. However, in many cases, network density is a function of other factors such as cost and communication range, and thus may not be a parameter of choice. We therefore present techniques for dealing with both inadequate and excess network density.

Temporal Aggregation: This technique is used to obtain the necessary separation of estimated influence fields when the network density does not meet the minimum requirements. *Temporal Aggregation* involves the following steps.

1. Using Theorems 1 and 2, compute minimum density $\hat{\rho}$ required to distinguish all objects
2. Given network density ρ, choose the *aggregation interval* t such that $\rho \times t > \hat{\rho}$
3. Aggregate node detections over time t to estimate object influence fields.

The aggregated influence field, used in a spatio-temporal context, is the area covered by an object in time t and it depends on the size, shape and motion model of the object. The proof of why temporal aggregation helps achieve separation can be deduced by rewriting Eq. 4.5 as follows.

$$E(d_{i+1}) - E(d_i) > (3 \times \sqrt{V(d_{i+1})}) + (3 \times \sqrt{(V(d_i))}) \tag{4.9}$$

We see that when aggregated over time, expected values grow faster than standard deviations, hence the desired inequality can be satisfied. Thus, temporal aggregation can also be used to distinguish between objects that have the same influence field but different speeds or motion models.

Theorem 4.2.3. *Separation between estimated influence fields of objects is preserved in networks with insufficient density or for objects with different speeds, by aggregating detections over interval t.*

Probabilistic Reporting:

This technique is used to improve system efficiency and lifetime in cases where network density exceeds the minimum specified by Theorems 1 and 2. *Probabilistic Reporting* involves the following steps.

For each node:

1. Compute probability p_r of reporting a detection.
2. For each detected object, send message to aggregator with probability p_r

Computing p_r: Consider the analysis presented earlier wherein each node detects an object with probability p_n. If each detecting node reports with probability p_r, the number of reporting nodes in an area A_i is a random variable with mean and variance as given below.

$$E(r_i) = \rho \times A_i \times p_n \times p_r \tag{4.10}$$

$$V(r_i) = A_i \times \rho \times p_n \times p_r \times (1 - \rho \times p_n \times p_r) \tag{4.11}$$

In order for separation between estimated influence fields to be maintained whp, the following inequality must hold for each pair $(i,\ i{+}1)$ where $(p_r)_{i(i+1)}$ is the probability of reporting.

$$E(r_{(i+1)}) - (3 \times \sqrt{(V(r_{(i+1)}))}) > E(r_i) + (3 \times \sqrt{V(r_i)}) \tag{4.12}$$

Let $(p_r)_{i(i+1)}$ be the minimum probability required to distinguish between objects i and $i+1$. By solving Eqns. 4.10, 4.11 and 4.12, we get the following.

$$(p_r)_{i(i+1)} = \frac{B}{((p_n \times \rho)+(B \times p_n \times \rho))}$$

where

$$B = 9 \times \frac{(\sqrt{A_{(i+1)}}+\sqrt{A_i})^2}{(A_{(i+1)}-A_i)^2}$$

Theorem 4.2.4. *The minimum probability, p_r, of reporting a detection required to distinguish between all object types is the maximum of all pairwise reporting probabilities $(p_r)_{i(i+1)}$.* □

4.2.2 Network faults

In this subsection, we discuss the impact of fading and contention faults on reliable estimation of influence fields.

Channel fading

Recall that p_f is the probability of message reception over a single hop in the presence of fading. Thus, over h hops, the probability of successful reception of a message equals $(p_f)^h$. Since messages from nodes closer to the aggregator have lower probability of failure, the estimated influence field of a small object close to the aggregator can overlap with that of a large object far away. We therefore need to compensate for the effect of distance of an object from the aggregator during its estimation.

Implementing distance insensitivity

We now present necessary conditions to maintain separation between the estimated influence fields of object types in the presence of fading over multiple hops en route to the aggregator. For simplicity, we assume that the object size is small as compared to the distance from the aggregator, hence all detections corresponding to the same object travel the same number of hops.

Probabilistic Reporting. To compensate for non-uniform reception probability, we use the probabilistic reporting technique presented earlier. In this case however, the reporting probability p_r is not uniform for all nodes, rather it depends on the distance to the aggregator.

Let D be the maximum number of hops to the aggregator in the network . The probability of reporting for a sensor at distance of h hops from the aggregator is chosen to be $p_f^{(D-h)}$. Thus, the probability of reporting for nodes D hops from the aggregator is 1, while for $h = 1$, the probability of reporting is $p_f^{(D-1)}$.

We first show that distance-dependent probabilistic reporting compensates for the effect of distance on estimation. Since fading errors are independent, the number of successful transmissions at any distance has a binomial distribution. The number of messages successfully received for an object i which is h hops away from the aggregator, is thus a random variable $f(h)$ whose mean and variance are obtained as follows.

$$E(f(h)_i) = A_i \times \rho \times p_f^{D-h} \times p_f^h = A_i \times \rho \times p_f^D \tag{4.13}$$

$$V(f(h)_i) = A_i \times \rho \times p_f^D \times (1 - (\rho \times p_f^D)) \tag{4.14}$$

From Eq. 4.13 and Eq. 4.14, we observe that the distribution of the number of successfully received messages is now independent of the number of hops.

We now derive a necessary condition for distinguishing objects whp. To achieve this, it can be shown that for each pair of objects (i, $i+1$), the number of messages successfully received whp for the smaller object i located at $h = 1$ should be less than the number of messages successfully received whp for the larger object $i+1$ located at $h = D$, as this represents the worst case. We thus have the following equation.

$$E(f(D)_j) - (3\sqrt{V(f(D)_j)}) > E(f(1)_i) + (3\sqrt{V(f(1)_i)}) \tag{4.15}$$

Let $\rho_{i(i+1)}$ be the minimum density required to maintain separation between estimated influence fields of objects i and $i+1$. Solving Eqns. 4.13, 4.14 and 4.15, we get the following.

$$\rho_{i(i+1)} = \frac{B}{(p_f^D + (B \times p_f^D))}$$

where

$$B = 9 \times \frac{(\sqrt{A_{(i+1)}} + \sqrt{A_i})^2}{(A_{(i+1)} - A_i)^2}$$

Theorem 4.2.5. *Let each node h hops from the aggregator report its detections with probability $p_f^{(D-h)}$ where D is the maximum number of hops to the aggregator. The minimum density $\hat{\rho}$ required to maintain separation between the estimated influence fields for all object types in the presence of multi-hop fading faults, is the maximum of all pairwise densities $\rho_{i(i+1)}$.* □

Note that since our techniques are compositional, we can deal with conditions where we have less than or more than this density $\hat{\rho}$ by using the techniques discussed earlier.

Spatial Reconstruction. As an alternative to probabilistic reporting, we present the spatial reconstruction technique which involves the following steps:

1. Upon detecting an object, each node sends a message to the aggregator.
2. For each distance h, aggregator scales number of received messages from that distance by $\frac{1}{p^h}$.

Thus, if the aggregator receives k messages from distance h, it considers this as having received $\frac{k}{p^h}$ messages. The scaling factor is a result of the uniform probability of receiving a message from a node h hops away being p_f^h. Spatial Reconstruction is the dual of distance-dependent Probabilistic Reporting. In this case, all detecting nodes transmit with the same probability, which may be lower than 1 for reasons of efficiency. The minimum network density required to distinguish between object types is the same as in Theorem 5 because the probability of fading loss is independent of the number of messages being transmitted.

Channel contention

In this subsection, we analyze the effect of interference due to channel contention on a single hop. In our event based traffic model, all nodes detect an object and hence compete for the channel at nearly the same instant. Thus, as the event size increases, the message losses increase too. We analyze the effect of channel contention on aggregation, under the assumption of the following one hop model.

Suppose n nodes, all within one hop of each other and the aggregator, want to send a detection message to the aggregator. Each node randomly chooses one of c time slots for transmitting the message. Let c be greater than n. If multiple nodes

choose the same slot, their messages collide and all of them are lost. This models a random backoff MAC scheme, commonly used in wireless communications.

The expected number of messages successfully received by the aggregator is the expected number of time slots that are chosen by exactly one node. This is an instance of a classical occupancy problem in combinatorics. The probability that a slot is chosen by exactly one node is equal to the probability that all other nodes choose different slots. This is the probability that a message does not get lost due to channel contention, which we denoted as p_c in Sec. 3.3.1.

$$p_c = (1 - 1/c)^{(n-1)} \tag{4.16}$$

The number of nodes with a time slot for themselves, i.e., the number of messages that do not get lost due to channel contention is a random variable having a binomial distribution with parameters (n, p_c). The mean and variance of the distribution, denoted as $E(s)$ and $V(s)$, are as follows:

$$E(s) = n \times p_c = n \times (1 - \frac{1}{c})^{(n-1)} \tag{4.17}$$

$$V(s) = n \times p_c \times (1 - p_c) \tag{4.18}$$

From Eq. 4.17, it is seen that for a given c, as n increases, the expected number of successful messages reaches a maximum and then starts decreasing.

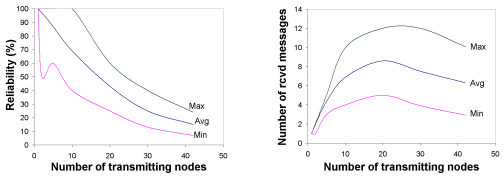
Definition 14 (Inversion point)**.** The *inversion point* of a network with respect to a given observer is the number of senders for which the expected number of messages received is maximum.

The inversion point, denoted as n_{inv}, obtained by solving for the maxima of Eq. 4.17 is as follows.

$$n_{inv} = \frac{1}{ln(1 + \frac{1}{c})} \tag{4.19}$$

Due to inversion, the aggregator may receive fewer detection messages for a larger object than it receives for a smaller object, hence the separation between the estimated influence fields of the object types may not be preserved.

Experimental results.



(a) (b)

Figure 4.2: Inversion in a one hop network.

Fig. 4.2 shows the experimentally measured impact of increasing the number of transmitters on the network reliability of the single hop model. This experiment was performed using Mica2 motes running TinyOS [32], using globally synchronized time to generate concurrent messages. The nodes were placed within one hop of each other and of the aggregator and their transmit power was set to be high enough to negate fading losses. The experimental results in Fig. 4.2 have been averaged over 50 trials for each of the traffic loads of size 2,5,10,20,30 and 40 sources. For these traffic loads, not only does the reliability of the network decrease significantly as the number of nodes increases as seen in Fig. 4.2(a), but it leads to the inversion effect seen in

Fig. 4.2(b). This inversion may cause an overlap between the number of messages received at the aggregator implying that previously separable influence fields will no longer be separable.

Implementing contention insensitivity

In this subsection, we describe two techniques to compensate for inversion effects produced by network contention to preserve separation between estimated influence fields.

Probabilistic Reporting. We use the same technique described earlier to compensate for contention effects, however the constraints for choosing the reporting probability p_r differ as follows.

If each detecting node reports with probability p_r, the number of reporting nodes is a random variable with expected value and variance as follows.

$$E(r_i) = A_i \times \rho \times p_r \tag{4.20}$$

$$V(r_i) = A_i \times \rho \times p_r \times (1 - \rho \times p_r) \tag{4.21}$$

Recall from Eq. 4.16 that the probability of a message being successfully received for an object i is dependent on the number of reporting nodes, which itself is a random variable. We make a simplifying, yet conservative, assumption that while the number of reporting nodes for an object is a random variable, the probability of successful reception is uniform and depends on the expected number of reporting nodes. This assumption results in a smaller traffic load than the expected value being subjected to larger contention than it would really experience. Similarly, larger traffic loads

are subjected to lower contention than actual. Consequently, the interval over which the number of received messages is distributed subsumes the interval that would be obtained in practice. Hence, the necessary conditions for maintaining separation between object types, resulting from our assumption are conservative. We now have for object i, the following equation.

$$p_{c_i} = (1 - 1/c)^{(E(r_i)-1)} \tag{4.22}$$

Using Eqns. 4.20 and 4.22, the number of messages that are successfully received for this object is now a random variable whose mean and variance are given by the following equation.

$$E(s_i) = A_i \times \rho \times p_r \times p_{c_i} \tag{4.23}$$

$$V(s_i) = A_i \times \rho \times p_r \times p_{c_i} \times (1 - (\rho \times p_r \times p_{c_i})) \tag{4.24}$$

For separation between estimated influence fields to be maintained whp, we require the following inequality to hold for each pair $(i, i+1)$.

$$E(s_{(i+1)}) - 3 \times \sqrt{V(s_{(i+1)})} > E(s_i) + 3 \times \sqrt{V(s_i)} \tag{4.25}$$

Selecting p_r: Solving the above inequality yields a quadratic whose solutions denote the minimum and maximum probabilities of reporting for which the two object types can be distinguished. The following procedure can be used to select the probability of reporting such that the estimated influence fields for all object types are separable.

1. For each pair $(i, i{+}1)$, where $1 \leq i$ and $i < k$, using Eq. 4.23, 4.24 and 4.25, obtain a range of probabilities given by the closed interval $(min((p_r)_{ij}), max((p_r)_{ij}))$.

2. Let $(p_r min, p_r max)$ denote the intersection of all such ranges.

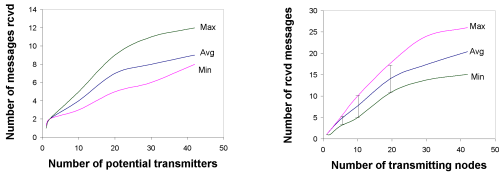
3. If the intersection is not empty, choose $p_r = p_r min$.

Theorem 4.2.6. *Assume p_r is the probability determined by the selection procedure. Separation between estimated influence fields of all objects is achieved when each node reports its detections with probability p_r.* □

Note that there may be cases where the selection procedure returns an empty intersection in Step 2. We now describe an additional algorithmic technique to deal with such cases.

Temporal Segregation. If the procedure described above returns an empty range of feasible reporting probabilities, it means that there exist objects a, b, c in order of increasing influence field sizes such that $max((p_r)_{ab}) < min((p_r)_{bc})$. Thus, there exist pairs for which eliminating inversion requires such a small probability of reporting that other objects are no longer distinguishable. To overcome this problem, the *inversion point* n_{inv}, has to be increased. According to Eq. 4.19, this can be achieved by increasing the number of time slots c. In other words, we temporally segregate the messages. Temporal segregation is also achieved by using additional application level backoffs before reporting a detection. Note that one can also eliminate the problem of channel contention by precisely scheduling the transmission of messages. One example of such a scheme is TDMA. The drawback of all such schemes is that they incur an additional delay overhead.

Experimental results. Recall the inversion effect demonstrated experimentally in Fig. 4.2. We now demonstrate how the techniques described above compensate for the inversion effect, allowing us to distinguish between object types under consideration. The experimental setup is the same as in the previous experiments with the same internode distances, same transmit power and the same traffic source sizes.



(a) Probabilistic reporting

(b) Temporal segregation

Figure 4.3: Dealing with inversion using Contention compensation techniques.

Fig. 4.3(a) demonstrates the results of using Probabilistic Reporting with probability 0.5. At each concurrent sending event, all the potential transmitters independently decide whether or not to transmit their message. The graph shows that by using probabilistic reporting, inversion is avoided for the traffic loads under consideration. However, note that the intersection in this case is empty for sources of size 5, 10 and 20. Thus, even with probabilistic transmission, we do not achieve disjoint ranges of message reception.

Fig. 4.3(b) demonstrates how Temporal Segregation helps achieve this separation. In this experiment, the number of slots available for choosing when to transmit was increased 4 times as compared to the previous case. As seen from the graph, by increasing the number of slots, we are able to avoid inversion for the traffic loads under consideration. The graphs also demonstrate that the overlap between messages

received is eliminated for sources of size 5, 10 and 20. In fact, if the number of slots were increased even further, there would be no overlap for any source size.

4.3 Putting it all together: Classification and Tracking

In this section, we show how the analysis and techniques for isolated fault classes presented earlier are composed, and discuss the effect of end-to-end unreliability on reliable estimation. We also present a procedure for distinguishing objects from false positives and discuss how object locations can be tracked by shape estimation of influence fields.

4.3.1 Composing fault classes

In this subsection, we describe how the necessary conditions for maintaining separation between estimated influence fields of objects can be derived when multiple faults can occur simultaneously. We illustrate this compositional approach through an example of node faults; other fault classes can be dealt with similarly.

Recall from Section 4.2, Eqns. 4.5 and 4.8, which specify conditions for maintaining separation between the estimated influence fields of two objects i and $i+1$ in the presence of false negatives and false positives respectively. However, if both these faults can occur, neither of these conditions is sufficient. The necessary condition for distinguishing between two objects i and $i+1$ in the presence of both type of node faults can thus be stated using the following equations.

$$E(d_i)+(3\times\sqrt{(V(d_i)})+E(fp_i)+(3\times\sqrt{V(fp_i)}) < E(d_{i+1})-(3\times\sqrt{V(d_{i+1})}) \quad (4.26)$$

Let $\rho_{node_{i(i+1)}}$ be the minimum density required to distinguish between i and $i+1$, obtained using Eq. 4.26.

Theorem 4.3.1. *The minimum density ρ_{node} required to distinguish between all objects in the presence of both false negatives and false positives is the maximum of all pairwise densities $\rho_{node_{i(i+1)}}$.*

4.3.2 End-to-end reliability

The method described above can be used to analyze the effects of multiple fault classes affecting reliable estimation. We now present a unified analysis for studying the impact of end-to-end reliability. This net reliability encapsulates all losses that may be encountered including false negatives, fading and contention losses. We do distinguish false positives in this analysis as they are additive faults. As discussed in Sec. 3.3.1 and as we will demonstrate in the next section, we characterize end-to-end reliability empirically. We denote the uniform probability of receiving a detection from a node in the influence field as $p_{rcv_{ih}}$. Using distance-dependent Probabilistic Reporting or Spatial Reconstruction, we can compensate for the effect of distance on reliability. The mean and variance of the number of detections received by the aggregator for object i, respectively $E(rcv_i)$ and $V(rcv_i)$, then are as follows:

$$E(rcv_i) = \rho \times A_i \times p_{rcv_i} \tag{4.27}$$

$$V(rcv_i) = A_i \times \rho \times p_{rcv_i} \times (1 - \rho \times p_{rcv_i}) \tag{4.28}$$

Similarly, the mean and variance of the number of false positives in the area $A_{i+1} - A_i$ which are received by the aggregator, respectively $E(fp_i)$ and $V(fp_i)$, are as follows.

$$E(fp_i) = \rho_{i(i+1)} \times (A_{i+1} - A_i) \times p_{fp} \times p_{rcv_i} \tag{4.29}$$

$$V(fp_i) = E(fp_i) \times (1 - \rho_{i(i+1)} \times p_{fp} \times p_{rcv_i}) \tag{4.30}$$

From Eqns. 4.27, 4.28, 4.29 and 4.30, we obtain the following equations.

$$E(rcv_i) + 3 \times \sqrt{V(rcv_i)} + Efp_i + 3 \times \sqrt{V(fp_i)} < E(rcv_{i+1}) - 3 \times \sqrt{V(rcv_{i+1})} \tag{4.31}$$

Let $\rho_{net_{i(i+1)}}$ be the minimum density required to maintain separation between the estimated influence fields of objects i and $i+1$, obtained using Eq. 4.31.

Theorem 4.3.2. *The minimum network density ρ_{net} required to maintain separation between all object types whp in the presence of false positives and end-to-end unreliability in the network is the maximum of all pairwise densities $\rho_{net_{i(i+1)}}$.* □

4.3.3 Identifying and isolating objects

The analysis presented earlier describes how to choose network density and/or fault compensation techniques so that the estimated influence fields of all objects can be distinguished. Assuming these requirements are satisfied, we now describe a procedure to filter false positives from a set of detections received for one or more objects so that these objects can then be classified and tracked.

Let ρ_{net} be the network density which, as described in the previous subsection, suffices to distinguish between all object types in the presence of faults. We now define the *active density* ad_i for object i as the ratio of the minimum number of detection messages that may be received for object i to the influence field area of object i.

$$ad_i = \frac{(E(rcv_i) - 3 \times \sqrt{V(rcv_i)})}{A_i} \tag{4.32}$$

Let ad_{min} be the minimum active density among all objects and let i_{min} be the object with this minimum active density. We then apply the following spatial filtering algorithm to isolate object detections from false positives:

1. For each received detection m, apply filtering windows of size A_1 around it.
2. If a window contains more than A_1 x ad_{min} detections, mark all these as object detections, else mark m as a false positive.
3. Repeat steps 1 and 2 till no more detections are unmarked.

It can be shown that the spatial filtering window size in Step 1 should be the influence field area of the smallest object, A_1. The uniform reliability property of the routing protocol in our model guarantees that if the minimum active density threshold is exceeded in a window, then all detections must belong to some object type. The above procedure thus identifies and isolates object boundaries.

4.3.4 Tracking using shape estimation

Having derived the minimum network density and/or compensation technique to achieve separation between estimated influence fields of detected objects, we can use the estimated shape of the influence field to track the object location. For instance,

a metallic object, which generates a uniform, circular influence field around it can be tracked at the centroid of the locations of magnetometers that detect it, while a light source producing a conical beam can be tracked at the vertex of the photosensors detecting it.

To achieve tracking, we require that nodes from which detections used in shape estimation are received, should be distributed uniformly across the influence field, otherwise the estimated shape of the influence field may be distorted. False negatives and false positives occur independently at nodes hence their distribution is uniform across the influence field. The one-hop contention model for faults is based on nodes randomly selecting the same slot for transmission, hence the distribution of these faults is uniform. Also, recall from Sec. 3.3.1 that the GridRouting protocol has the property of uniform reception probability for nodes equidistant from the aggregator. In the multi-hop case though, the probability of failure is non-uniform because farther nodes are subject to a higher loss rate. However, as shown in Section 4.2, the techniques of distance dependent probabilistic reporting and spatial reconstruction compensate for this non-uniformity of network failures.

4.4 Case study: A Line In The Sand

In this section, we describe the design and implementation of a distributed classification and tracking system which we called *A Line In The Sand*. This system consists of 90 Mica2 motes deployed in a 1.5m spaced grid to cover a 18m x 7m area. *A Line In The Sand* has been deployed in several outdoor settings to accurately distinguish between civilians, soldiers and vehicles by estimating their influence fields based on

magnetometer and micro-power impulse radar sensors. For simplicity of presentation, we only describe classification between a soldier and a car using magnetometer based influence fields. We first describe how we derived system parameters like density using the theorems presented earlier. We then validate, both theoretically and experimentally, that accurate classification and tracking can be achieved whp (99%) in this network.

4.4.1 Experimental measurements

We first describe the experimental setup to measure key system parameters. To measure the magnetic influence fields of a soldiers and a car, a dense, regular grid of Mica2 motes with magnetometers was deployed. These objects were then made to traverse this network at different speeds and orientations. We then averaged the observations from over 100 such trials. The influence field areas A_1 and A_2, for a soldier and a car, were thus measured to be $12m^2$ and $63m^2$ respectively. The probability of node faults, $1 - p_n$, which included node failures and false negatives was measured to be 10%. By observing the number and distribution of false positives in the network over time and location, we determined the probability of false positives, p_{fp} to be 2%.

4.4.2 Determining network density

Substituting for the experimentally measured values of A_1, A_2, p_n and p_{fp} in Theorem 7, we obtained the following conditions for minimum network density:

$$\rho_{01} > 0.6 \quad , \quad \rho_{12} > 0.65$$

where ρ_{01} is the minimum density needed to distinguish false positives from a soldier and ρ_{12} is the minimum density needed to distinguish a soldier from a car. The minimum density needed to distinguish between all three, $\hat{\rho}$ was thus 0.65. Based on the communication radius of these nodes, we deployed 90 nodes in a 1.5m spaced grid to cover the 18m x 7m area with a network density of 0.7.

4.4.3 Effect of network unreliability

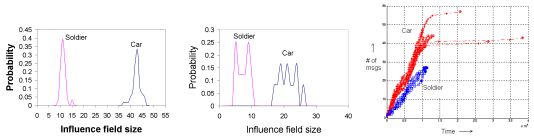
As mentioned in Section 4.3, we characterized the end-to-end network reliability function empirically. To do so, we deployed the network with the desired density of 0.7 and repeated the earlier trials with a soldier and car traversing the network and measured the network reliability at the aggregator. By averaging over more than 100 such trials, we obtained the values of p_{rcv_1} and p_{rcv_2} as 0.85 and 0.55 respectively. Substituting for p_{rcv_1},p_{rcv_2} and p_{fp} in Eqns. 4.27, 4.28, 4.29 and 4.30, we obtain the following equations.

$$E(rcv_1) = 7.1,\ V(rcv_1) = 2.9$$
$$E(rcv_2) = 24.25,\ V(rcv_2) = 14.9$$
$$E(fp_1) = 0.6,\ V(fp_1) = 0.6$$

It can be seen that these values satisfy the inequality in Eq. 4.31, thereby validating that the estimated influence fields of a soldier and a car can be separated in the presence of end-to-end unreliability and false positives.

4.4.4 Experimental validation

We now present experimental data to demonstrate that we were able to distinguish between the given object types in the presence of node faults and network unreliability.



(a) Measured at motes (b) Measured at aggregator (c) Temporal evolution

Figure 4.4: Impact of network reliability on influence fields in *A Line In The Sand*

Fig. 4.4(a) shows the probability distribution function for the influence fields of a soldier and a car as measured at the aggregator. In this outdoor experiment, timestamped detections were recorded in the non-volatile memory at each mote during 100 runs of a soldier and vehicle each, moving through the network. These detections were then downloaded and time-correlated to recreate the measured influence fields and their probability distribution. It can be seen from Fig. 4.4(a) that these measured influence fields are indeed clearly separable. Fig. 4.4(b) shows the probability distribution of the influence fields estimated by the aggregator based on detections received using the GridRouting protocol in 100 runs of a soldier and a car each. It can be seen that in the presence of false positives and network unreliability, separation between the estimated influence fields of a soldier and a car is lower than in Fig. 4.4(a). However, as calculated earlier, we see that these distributions are non-overlapping meaning that the number of detections received for a vehicle is always greater than that for a soldier. From this data, we also calculate that there exists little variability (7.2%) in network reliability across these runs. Fig. 4.4(c) shows the temporal evolution of influence field estimation at the aggregator. The proximity of traces for individual runs indicates that in addition to reliability, network delays are

also quite predictable, allowing us to choose tight latency bounds at the aggregator. This serves to validate our model that uniform, predictable end-to-end reliability can be obtained by careful design of the routing protocol.

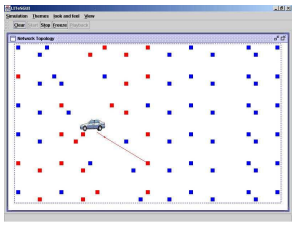


Figure 4.5: Classification and tracking of a car in *A Line In The Sand*

4.4.5 System performance

Finally, we give some performance data for *A Line In The Sand*. By considering the influence field analysis and appropriately tuning the desired network parameters, we were able to achieve the desired classification accuracy of 99%. The accuracy of tracking was higher for a soldier (1-2m) as compared to a vehicle (3-5m) providing further evidence of the claim that reliability and uniformity are dependent on the object type. The system was able to classify and track multiple objects moving concurrently through the network as long as they were separated by a minimum distance threshold. Fig. 4.5 shows a snapshot of the classification and tracking output produced by the system for a car moving through the network.

4.5 Extensions to the Influence Field Approach

In this section, we describe extensions to the influence field to increase the confidence in classification and also decrease the required density of deployment. One

such extension is the use of multiple sensing modalities at each node to estimate multiple influence fields. As an example, consider the case where we wish to distinguish between a motorcycle, a SUV and a truck. The influence field areas for these objects using a magnetometer are $28m^2$, $78m^2$ and $150m^2$ respectively. The minimum density required to distinguish all three objects in the presence of faults, as calculated using the theorems presented earlier, is quite high (2-3m grid spacing). However, the influence field areas of the same objects with respect to an acoustic sensor are $1250m^2$, $700m^2$ and $1250m^2$ respectively. By estimating the influence fields for both these modalities, it is possible to distinguish between these objects with a much lower network density (8-10m grid spacing). The basic idea is to use the estimated acoustic influence field to distinguish a SUV from the other two object types and then use the magnetic influence field to distinguish between a motorcycle and a truck. Thus, using multiple sensing modalities reduces the network density to 6-10% of what was originally required. This simple example serves to demonstrate that estimating multiple influence fields can be used to classify multiple types of objects with higher confidence and lower network density. Multimodal influence field estimation was successfully demonstrated in the *ExScal*[4] network, which is one of the largest sensor deployments to date. *ExScal* used three types of sensors – magnetometers, acoustic and motion sensors – to estimate the influence fields of a person, a car and an ATV and classify and track them accurately.

Another extension to the influence field concept involves communicating not only presence outputs from each node, but some vector of features about the object like peak amplitude, total energy or distance from the object. Such an enhanced influence field can be used to improve confidence in the estimation output.

Regardless of whether the basis chosen for determining the influence field is merely presence as in the case of *A Line In The Sand*, a combination of multiple modalities, or a vector of features, reliable estimation is still an important problem and the same techniques described in this paper can be applied to each of these extensions.

4.6 Related Work

The instrumentation of a militarized zone with distributed sensors is a decades-old idea. Unattended ground sensors (UGS) exist today that can detect, classify, and determine the direction of movement of intruding personnel and vehicles. The Remotely Monitored Battlefield Sensor System (REMBASS) exemplifies UGS systems in use today. REMBASS exploits remotely monitored sensors, hand-emplaced along likely enemy avenues of approach. These sensors respond to seismic-acoustic energy, infrared energy, and magnetic field changes to detect enemy activities. REMBASS processes the sensor data locally and outputs detection and classification information wirelessly, either directly or through radio repeaters, to the sensor monitoring set (SMS). Messages are demodulated, decoded, displayed, and recorded to provide a time-phased record of intruder activity at the SMS.

Like REMBASS, most of the existing radio-based unattended ground sensor systems have limited networking ability and communicate their sensor readings or intrusion detections over relatively long and frequently uni-directional radio links to a central monitoring station, perhaps via one or more simple repeater stations. Since these systems employ long communication links, they expend precious energy during transmission, which in turn reduces their lifetime. For example, a REMBASS sensor node, once emplaced, can be unattended for only 30 days.

In contrast, we design a dense, distributed, and 2-dimensional sensor network-based classification and tracking system using inexpensive sensor nodes. In this model, intrusion data are processed locally at each node, shared with neighboring nodes if an anomaly is detected, and communicated to an exfiltration gateway with wide area networking capability.

The notion of influence of an energy source is used in other science and engineering applications. In some formulations, the distribution of the intensity of the source at various points is considered while modelling its influence. For example, Kellogg et al [6] model the temperature distribution of a heat source across a region as an *influence graph* and use the graph to design algorithms for distributed control. In other formulations, including Zhao et al [74] and ours, the distribution of the intensity is not modelled. Zhao et al [74] define an *influence area* as the number of sensors that detect an object. Our definition of the influence field also captures the shape of the influence field. We are unaware of previous work that has used influence field estimation as a basis for classification and tracking of objects.

The influence field approach should be contrasted to traditional approaches for classification and tracking using Unattended Ground Sensors [31]. The Remotely Monitored Battlefield Sensor System (REMBASS) is a representative example. The approach is centralized, requires complex pattern matching[16] and the sensing and processing devices are expensive, require careful and precise deployment as well as frequent remote monitoring. Meesookho, et al [55] describe a collaborative classification scheme based on exchanging local feature vectors, which imposes a high load on the network. By way of contrast, most of the work on distributed tracking that decreases the load on the network is based on collaborative signal and information processing,

sequential Bayesian filtering, and extended Kalman filtering [74, 21, 50, 51, 54, 75, 26], that require significant node computation.

For the case of node faults, Krishnamachari et al [45] have presented probabilistic decoding mechanisms to detect regions of events in the presence of uncorrelated sensor faults with relatively low probability (around 10%). Our work accommodates the case of uniform nodal failures and we have also presented techniques to handle network faults whose impact is non-uniform across the network such as fading, and network faults whose failure probability grows with the event size such as contention. To the best of our knowledge, the impact of network unreliability in estimating the influence field has not been addressed before, nor has it been addressed in the context of distributed classification and tracking, which has led us to the present work.

Our work also relates influence field to sensor coverage [29, 73] and highlights the important similarities and distinctions between the two concepts. Existing work on sensor coverage gives necessary conditions for detecting an object moving through a network. We are not aware of extensions that deal with the problem of classification.

4.7 Summary

In this chapter, we considered the problem of reliably estimating the influence fields of different target types in a wireless sensor network subject to a variety of faults. We provided mechanical procedures for sensor node density selection as well as algorithmic techniques appropriate for dealing with each fault class. Corroboration of our results and techniques was provided through at-scale experiments.

We showed how reliable estimation was achieved to enable accurate classification and tracking in *A Line In The Sand*. The case study also provided a data point for

the significant impact of network unreliability on network and application design, as well as one for a need for routing protocols in sensor networks to provide uniform reliability.

Our work reveals a notable co-dependence between application design and network design. To achieve the desired estimation reliability, we needed in some cases to use both techniques that affected the network (such as tuning of MAC or routing protocol parameters) and that affected the application (such as tuning the probability of reporting and the rate of temporal aggregation). How to design stable and scalable systems when there are such cyclic dependencies involved is an issue of interest to us.

Although our compositional models allow us to reason about the effects of different types of node and network faults, there are some relevant and more complex fault models that we have not dealt with analytically. One such model, which we dealt with only experimentally in *A Line In The Sand* concerns multi-hop contention and fading errors. In future work, we seek to address this model analytically. We will also incorporate in our analysis consideration of multiple concurrent targets that we dealt with experimentally, towards addressing the gap between existing theory and practice.

CHAPTER 5

DISTRIBUTED VIBRATION CONTROL

Sensor-actuator networks are being prototyped in the control of distributed parameter systems such as flexible structures. A specific example is the vibration control of a fairing during payload launch using embedded MEMS components based sensor-actuator networks [12, 72]. Since MEMS based sensor-actuator devices are potentially cheap, a large number of these devices can be embedded on flexible structures and combinations of these sensors can be used to obtain the required mode vibration information and then the output from these combinations can be used to provide adequate distributed control. Similar applications arise in the control of chemical plants and nuclear reactors.

Distributed control systems have applications in space missions and nuclear plants where degradation of system performance may even compromise human safety. Hence satisfactory performance in the presence of faults is a requirement for these systems. The constraint in most distributed control applications is that of mission critical stability, but achieving this using wireless sensor/actuator networks is a challenge. Sensor-actuator network based control systems typically comprise of embedded sensors and actuators, microprocessor-based controllers (central or distributed) and an underlying network that provides information processing services to the controllers

such as controller group synchronization, communication, (re)parameterization, reconfiguration, etc. Each of the above subsystems are subject to faults: there are hardware faults and these will increase when subject to harsh and unpredictable environments, there are faults in the underlying software and middleware services such as information loss, delay and corruption, and there are configuration faults which given the scale of these networks this will increase even more.

We performed a series of experiments to analyze the effect of potential faults on a vibration control system. We carried out our experiments on the Boeing Open Experimental Platform, which is a simulation framework intended to capture the vibroacoustic damping problem on a satellite launch vehicle.

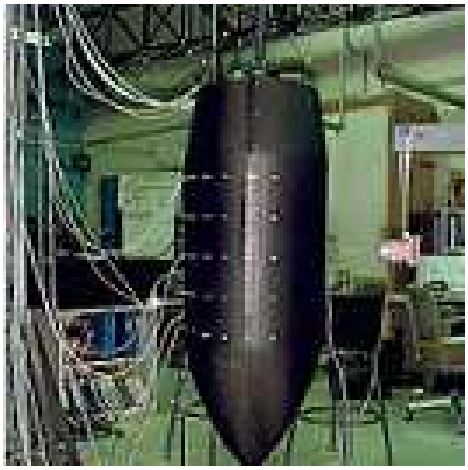


Figure 5.1: Fairing Shaped Payload Installed with Sensors and Actuators

In Fig. 5.1 we have shown a fairing shaped payload that is installed with sensors and actuators on its surface that is used to control the vibrations during payload launch. The simulated environment of the Boeing OEP application includes a fairing

plant model which is a simulation of the fairing structure, a hierarchical control application [33, 34] and a fault injection framework. The 100 node system is partitioned into several groups. Each group acts to damp a particular mode of vibration in the fairing. The number of nodes assigned to each mode depends on the frequency of that mode and the energy required to achieve the dampening. There can be as many as 20 modes to address.

We identified potential component and network level faults for the fairing control application [43]. Using a fault injection framework we evaluated the effect of these faults on the control performance of the Boeing OEP. Specifically, the fault types that we considered are: node fail-stop, node crash, nodes behaving randomly (in terms of sensing and actuation), nodes debonding from their surface and network faults.

From our experiments [43], we note that the underlying hardware faults can result in arbitrary behavior of sensor and actuators that can cause substantial degradation of performance. Our observations motivate the need for designing reliable control scheme that maintain stability and performance in the presence of arbitrary component faults. One of the methodologies for the design of fault-tolerant control systems involves real-time fault detection, isolation and control system reconfiguration [11, 39, 61, 44, 13]. An appropriate action is taken after the diagnosis of the faults. This method still leaves the following challenges. The hardware itself can be faulty causing the actuators to fail-stop and offer no control or debond from their surface causing them to offer incorrect control. It is sometimes also not feasible to integrate the fault detection, diagnosis and reconfiguration in dynamical systems particularly when the available reaction time is limited. The underlying fault detection service is itself vulnerable to faults in the middleware and software services. This leads us to consider a Byzantine

model for the actuator faults. A Byzantine actuator can produce an arbitrary control input to the plant at all times. The behavior is non-deterministic and it can even be the worst possible value at all times. Assuming this worst case scenario, we focus on designing systems that maintain asymptotic stability in the presence of Byzantine actuators that apply arbitrary control input to the plant.

Problem statement

> *Assuming that a bounded number of network actuators can exhibit incorrect (and potentially arbitrary) behavior, how can distributed control be designed to be provably stable?*

In this chapter, we model the system S to be marginally stable, linear time-invariant and multi-variable with m sensor-actuator pairs. We apply distributed local output feedback control to stabilize the system. The question then arises as to how can the control system be guaranteed to be stable when a fraction of the actuators in the network are Byzantine. *Byzantine insensitivity* implies that stability of the system is guaranteed despite a fraction of the actuators being Byzantine.

Given a maximum of k Byzantine actuators, we first determine a necessary number of actuators to guarantee asymptotic stability. Then for a 2 dimensional system, we determine a sufficient number of redundant actuators and determine conditions on placement of the actuators that will guarantee asymptotic stability of the control system. We demonstrate our approach using a beam vibration control application as a case study.

In Section 5.1 we describe the system and fault model and provide a sufficient condition for the stability of the system in the absence of faults. In Section 5.2, we first design a reliable control scheme using redundant colocated actuators and then design a reliable control scheme where the redundant actuators are not colocated and

the redundancy is further decreased. In Section 5.3, we demonstrate our methodology using a beam vibration control application [7, 64] as a case study. In Section 5.4, we discuss related work. We present a summry in Section 5.5.

5.1 System and Fault Model

In this section we describe the system and fault model and derive sufficient conditions for the asymptotic stability of the system without faults.

5.1.1 System Model

Consider a marginally stable linear time-invariant multivariable system S with m sensor-actuator pairs, described by the following equations and control law.

$$\dot{x} = Ax + Bu \tag{5.1}$$

$$y = Cx \tag{5.2}$$

where x is an n-dimensional state vector $[x_1, x_2, \cdots, x_n]^T$, u is an m-dimensional actuator vector, B is an $n \times m$ dimensional matrix and the individual sensor-actuator pairs are colocated. We assume that the system is controllable and observable from individual locations. Since S is marginally stable, A has eigenvalues on the imaginary axis. Since the individual pairs of sensors and actuators are colocated, we have the following condition.

$$B = C^T \tag{5.3}$$

Starting at any state, without any control being applied the system maintains its energy as it is marginally stable. We apply the following local on-off output feedback control law to stabilize the system.

$$u_i = \alpha \times sign(y_i), \quad i = 1....m \tag{5.4}$$

where α is less than zero.

Further u_i equals zero when y_i is zero. Thus a correct actuator can have 3 possible control values 0, $-\alpha$ and α. We choose $|\alpha|$, the magnitude of the actuator force, to be the maximum force that an actuator can apply and assume that this is the same across all actuators. □

5.1.2 Asymptotic Stability Without Faults

We now analyze and prove the stability properties of S in the absence of faults.

Theorem 5.1.1. *If $m \geq n$ and the matrix B is of rank n, the system S is asymptotically stable.*

Proof. We use the Lyapunov approach to prove stability. Now, let us define function V as

$$V = x^T M x \tag{5.5}$$

where M is a symmetric, positive definite $n \times n$ matrix. The Lyapunov derivative can then be written as

$$\dot{V} = x^T (A^T M + MA)x + 2x^T MBu \tag{5.6}$$

Since A is marginally stable, we can transform A to be skew symmetric and $A^T + A$ equals zero. Thus M can be the identity matrix.

$$\dot{V} = 2x^T Bu \tag{5.7}$$

Let B_i denote the i^{th} column of matrix B. For the system described in Eq. 5.1, we have

$$\begin{aligned}
\dot{V} &= 2 \times \alpha(\sum_{i=1}^{m}(x^T).(B_i) \times sign(y_i)) && (5.8)\\
&= 2 \times \alpha(\sum_{i=1}^{m}(x^T).(C_i^T) \times sign(y_i)) && (5.9)\\
&= 2 \times \alpha(\sum_{i=1}^{m}(y_i) \times sign(y_i)) && (5.10)\\
&= 2 \times \alpha(\sum_{i=1}^{m}|(x^T).(B_i)|) && (5.11)
\end{aligned}$$

Note that we can use the magnitude of the dot product $(x^T).(B_i)$ because we see from Eq. 5.10 that $(y_i) \times sign(y_i)$ is always positive. Since m is at least equal to n and B is of rank n, the state x can be orthogonal to at most $n-1$ actuators. Hence the Lyapunov derivative is strictly negative. Thus the system is asymptotically stable.

□

5.1.3 Fault Model

We now describe the fault model acting on system S. We start with the definition of a *Byzantine* actuator.

Definition 15 (Byzantine actuator)**.** A Byzantine actuator q is one that can generate arbitrary value of u_q in the range $-\alpha$ to α at all times.

We note that a Byzantine actuator behavior also captures the case of an actuator fail-stopping ($u_q = 0$, and an actuator debonding form its surface thereby applying a fraction of the control force ($0 \leq u_q \leq \alpha$). In our fault model, k out of the m actuators are Byzantine in system S.

We will prove that the system remains asymptotically stable even when the Byzantine actuators behave in the worst possible way at all times. This is described below. Let $u_{ci}(t)$ be the correct actuator value at any time t for actuator i. Let $u_{fi}(t)$ be the corresponding value generated if the actuator is Byzantine. We then have the following conditions.

$$W1: \quad u_{ci}(t) \neq 0 \quad \Rightarrow u_{fi}(t) = -u_{ci}(t) \tag{5.12}$$

$$W2: \quad u_{ci}(t) = 0 \quad \Rightarrow u_{fi}(t) = \pm\alpha \tag{5.13}$$

Note: If the system S is in equilibrium and is acted upon by a Byzantine actuator, then the system is subject to perturbation and the energy of the system increases. We do not consider this case in our fault model. We are interested in maintaining the asymptotic stability of S in the presence of Byzantine actuators.

5.2 Reliable Control System Design

In this section, we design two reliable control schemes that maintain asymptotic stability of the System S in the presence of Byzantine actuators.

5.2.1 Reliable Control System Using Redundant Colocated Actuators

In this scheme we place multiple actuators at each location. Thus the effect of each redundant actuator on the control stays the same.

Theorem 5.2.1. *A sufficient condition to tolerate k Byzantine actuators at each location and guarantee asymptotic stability in the system S is to have $2k+1$ actuators at each of the m locations, where $m \geq n$ and the B matrix formed by the m distinct locations is of rank n.*

Proof. Since there are $2k+1$ actuators at each location, the Lyapunov derivative in Eq. 5.11 can be written as follows

$$\dot{V} = 2 \times \alpha(\sum_{i=1}^{m}((2k+1) \times |(x^T).(B_i)|)) \tag{5.14}$$

First of all, we see from Eq. 5.14 that if the actuators are not Byzantine, the redundant actuators still keep the energy derivative negative. We now analyze the effect of Byzantine actuators at each location. Without loss of generality let us consider the q^{th} location and assume that k actuators at this location are Byzantine. We consider the 2 conditions $W1$ and $W2$, described in the fault model.

When condition $W1$ of the fault model applies, the energy derivative term corresponding to the q^{th} actuator location can be written as follows.

$$\dot{V}_q = 2 \times \alpha((k+1) \times |(x^T).(B_q)| - (k) \times |(x^T).(B_q)|) \tag{5.15}$$

$$= 2 \times \alpha(|(x^T).(B_q)|) \tag{5.16}$$

Thus we see that the energy derivative corresponding to the q^{th} location still stays negative. This can similarly proved for all locations.

Now consider condition $W2$. If $u_{cq}(t) = 0$, it implies that $y_q(t) = 0$, i.e the local output is zero. Thus the current state $x(t)$ is orthogonal to the vector C_q. Since the actuators are colocated, the current state $x(t)$ is also orthogonal to the vector B_q. Thus, the term $x^T.(B_q)$ is equal to zero no matter what force the Byzantine actuator applies.

Hence the system S with $2k + 1$ actuators at each of the m locations, is asymptotically stable in the presence of k Byzantine actuators at each location. □

Note that this scheme tolerates k Byzantine actuators per location. If the expected reliability ratio of the actuators are known, then we can design for the number of actuators required at each location.

Given a reliability ratio for the actuators (greater than 0.5), denoted as ρ, we can choose a k such that the system is reliable against Byzantine faults.

$$\frac{k}{(2k+1)} > (1-\rho) \tag{5.17}$$

Note However it should be pointed out that placing the redundant actuators at the same location may not be feasible in all control systems. Moreover the redundancy rapidly increases as n increases because the actuators are replicated at each location.

We now describe a reliable control scheme where the colocation of redundant actuators is not required and given that k actuators are Byzantine we add redundant controllers to the system as a whole thus decreasing the redundancy required.

5.2.2 Reliable Control System Without Using Colocated Actuators

We first state the minimum number of actuators to be added to the system S which ensures that the energy derivative of Eq. 5.11 is less than zero at all times.

Lemma 5.2.2. *For the energy derivative of Eq. 5.11 to be less than zero at all times in the presence of k Byzantine actuators, we require $m >= 2k + n$*

Proof. Let the number of actuators in the system be $2k + n - 1$. The state x can be orthogonal to at most $n - 1$ actuators. Let all of these be non-Byzantine actuators. Thus the energy derivative terms corresponding to these actuators is zero. There are $2k$ actuators left. Without loss of generality assume that in the presence of any k Byzantine actuators belonging to set of $2k$ actuators, the energy derivative is less than zero. Then for the same state x, if the remaining set of actuators had been Byzantine the energy derivative would be greater than zero. Thus we need at least $2k + n$ actuators for the energy derivative of Eq. 5.11 to be less than zero at all times in the presence of k Byzantine actuators. □

However, finding an actuator configuration that satisfies such a lower bound for any k and n is a complex problem. We now focus our attention on second order systems, i.e $n = 2$ and show that $3k+1$ is an upper bound on the number of actuators needed.

Definition 16 (m-uniform configuration system)**.** An *m-uniform configuration* of the system is the actuator configuration of the system in which each of the m columns of the B matrix has the same amplitude and are uniformly distributed in the state

space of n dimensions such that the column vectors of B are pairwise equi-angular and the angle between consecutive pairs of vectors is equal to $\frac{\pi}{m}$.

A second order system with 4-uniform and 7-uniform configuration is depicted in Fig. 5.2. For simplicity, Let α be equal to 1. Thus given the actuator locations, each actuator vector can either be equal or opposite to the direction shown depending on the current state of the system.

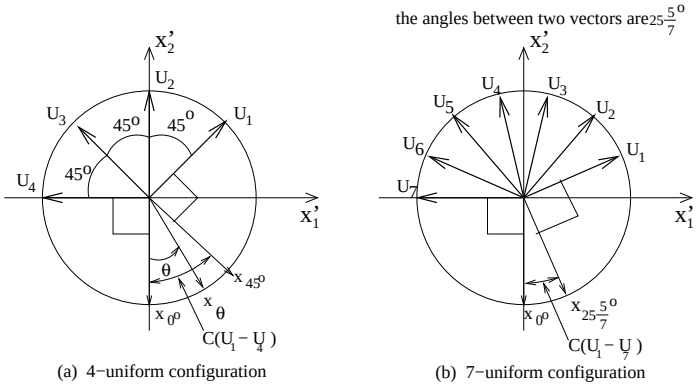


Figure 5.2: 4-uniform and 7-uniform configurations for the second-degree system

Let a unit state vector x_θ form an angle θ with the vertical axis as shown in the figure. When an actuator is behaving correctly, the actuator vector would be such that its dot product with the state vector is less than or equal to zero. This is because the system is observable from each location and each actuator applies control in a direction opposite to that of the local output. The inner product would be equal to zero when the actuator vector is orthogonal to the current state.

Thus, the 4-uniform configuration shown in Fig. 1(a) is the proper actuator configuration when θ is between 0° and 45°. In this configuration, the four normal actuators

$U_1, \cdots, U_4$ keep the energy derivative negative when θ lies anywhere between 0° and 45°. Note that the dot product of the state vector with each actuator vector is less than or equal to zero. If θ is between 45° and 90°, the first actuator changes its direction so that $\bar{U_1}$ is the new actuator vector. Thus the whole configuration is rotated by 45° in the clockwise direction. Thus in an $m - uniform$ configuration, if all the actuators are correct, then the actuator vectors remain pairwise equi-angular at all times.

Therefore while showing that a particular $m - uniform$ configuration is sufficient to guarantee asymptotic stability in the presence of Byzantine faults, it is enough to consider the case that the unit state vector x_θ is located in the *basic range* $[0.0^\circ, 45.0^\circ]$. In general, the basic range of m-uniform configuration system is $[0.0, \frac{\pi}{m}]$. Also note that it is enough to consider unit state vectors because all the actuator vectors are of same magnitude and the total dot product depends only on the angle.

In an m-uniform system of second-degree($m = 3k+1$), let $S(k, \theta)$ denote the set of k Byzantine faulty actuators such that, for a unit state vector x_θ, the corresponding energy derivative becomes maximized among all possible k subsets of actuators. Let $ED(k, \theta)$ be the corresponding energy derivative.

For example, in the 4-uniform configuration of the system, $S(1, 0^\circ)$ and $S(1, 45^\circ)$ are $\{U_2\}$ and $\{U_3\}$, respectively.

$$\begin{aligned} ED(1, 0^\circ) &= x_{0^\theta} \cdot (U_1 - U_2 + U_3 + U_4) \\ &= (\cos 135^\circ - \cos 180^\circ + \cos 235^\circ) \\ &= -0.4142 \end{aligned}$$

Likewise, $ED(1, 45^\circ)$ turns out to be equal to -0.4142. Thus, in the boundary angles of the basic range $[0.0^\circ, 45.0^\circ]$, the system is asymptotically stable due to the negative values of $ED(1, 0^\circ)$ and $ED(1, 45^\circ)$.

It is seen that for any m-uniform configuration, when the state vector is at the boundary of the basic range, one of the actuator vectors is orthogonal to the state vector and offers no control. Thus if $ED(k, 0^\circ)$ and $ED(k, \frac{\pi}{m})$ are both negative, the system is asymptotically stable in the presence of k Byzantine faults. We now write down the expressions for $ED(k, 0^\circ)$ and $ED(k, \frac{\pi}{m})$ in any m-uniform configuration.

$$\phi \;=\; \pi/m = \pi/(3k+1) \tag{5.18}$$

$$ED(k,\phi) \;=\; \sum_{i=1}^{k} \cos(\frac{\pi}{2} + i \cdot \phi) - \sum_{i=k+1}^{2k} \cos(\frac{\pi}{2} + i \cdot \phi) + \sum_{i=2k+1}^{3k+1} \cos(\frac{\pi}{2} + i \cdot \phi)$$

$$ED(k,0) \;=\; \sum_{i=0}^{k-1} \cos(\frac{\pi}{2} + i \cdot \phi) - \sum_{i=k}^{2k-1} \cos(\frac{\pi}{2} + i \cdot \phi) + \sum_{i=2k}^{3k} \cos(\frac{\pi}{2} + i \cdot \phi)$$

$$ED(k) \;=\; min(ED(k,0), ED(k.\phi)) \tag{5.19}$$

Upon numerical analysis of $ED(k)$ for a large spectrum of values for k from 1 to 1000, it turns out to be that all values of $ED(k)$ are negative as shown in the figure below. Thus an m-uniform configuration of actuators is sufficient to guarantee asymptotic stability of a second order system in the presence of k Byzantine actuators when $m = 3k + 1$.

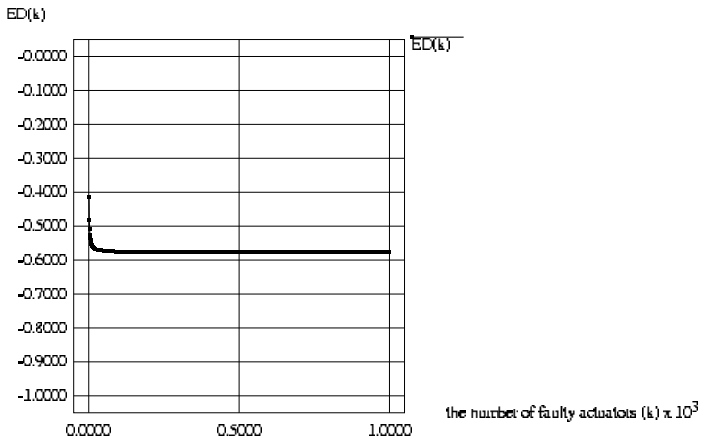


Figure 5.3: The maximum energy derivative $ED(k)$ in m-uniform configuration system

Remark Note that the case of $n = 2$ and $k = 1$, where we need 4 actuators to guarantee asymptotic stability satisfies the lower bound $2k + n$. Further, an upper bound on the redundancy required to tolerate k faults for higher dimension systems can be found in a related technical report [47].

5.3 Application to Beam Vibration Control System

We now apply our reliable control system designs on a local output feedback control scheme to a beam vibration control system. Given is a uniform beam of unit length, unit mass, and unit stiffness factor, that is restricted by pins at both ends and subjected to an initial disturbance. The beam has no dampening factor so that it may vibrate endlessly. The beam has colocated velocity sensors and actuators to reduce the vibration. For simplicity, we consider two fundamental modes of vibration.

The two fundamental vibration modes, denoted as M_1 and M_2, are derived [56] as follows:

$$M_1 \quad : \quad 1.4142 \sin \pi z, \quad \lambda_1 = \omega_1^2 = 97.41 \tag{5.20}$$

$$M_2 \quad : \quad 1.4142 \sin 2\pi z, \quad \lambda_2 = \omega_2^2 = 1558.55 \tag{5.21}$$

where $z \in [0.0, 1.0]$ denotes the position in the beam spatial axis and λ_i and ω_i, $i = 1, 2$, represent the eigenvalues and the frequencies of i-th modes, respectively.

Since each mode is governed by a second-degree differential equation, the state vector for the system contains four variables $x = [x_1, x_2, x_3, x_4]^T$. $x_1(x_2)$ and $x_3(x_4)$ denote the vertical displacement and velocity of first (second) vibration mode, respectively. Then, the system matrix A in Eq. 5.1 is denoted as

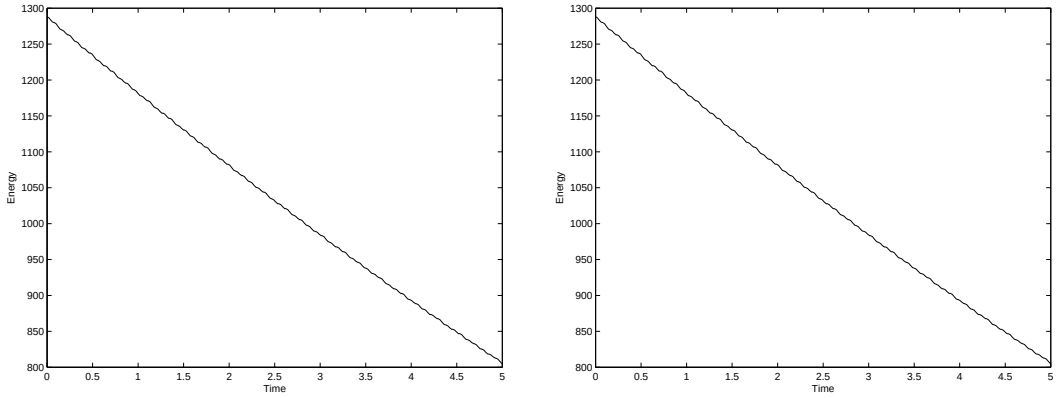
$$A = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -97.41 & 0 & 0 & 0 \\ 0 & -1558.55 & 0 & 0 \end{pmatrix}$$

Note that we use a velocity feedback control. So the control input does not have any effect on the states $x1$ and $x2$. The actuation is used to control the velocity states $x3$ and $x4$. We will assume that the beam cannot be deformed permanently. Thus when the velocity of the beam comes to zero, the displacement is also zero. Thus in this specific example although the number of states is 4, the control affects only the 2 velocity states.

We first show that using 2 sensor-actuator pairs that form a B matrix of rank 2, we can asymptotically stabilize the system. We choose the following B matrix.

$$B \quad = \quad \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1.4142 \\ 1.3066 & 1 \end{pmatrix}$$

The Fig. 3(a) shows the energy of the system staring from an arbitrray initial state going down to zero in the absence of faults. The energy of the system at time t is calculated as $x^T(t) \times X(t)$, where $x(t)$ is the state of the system.



(a) 1 actuator at each location, no faults

(b) 3 actuators at each location of which 1 is Byzantine

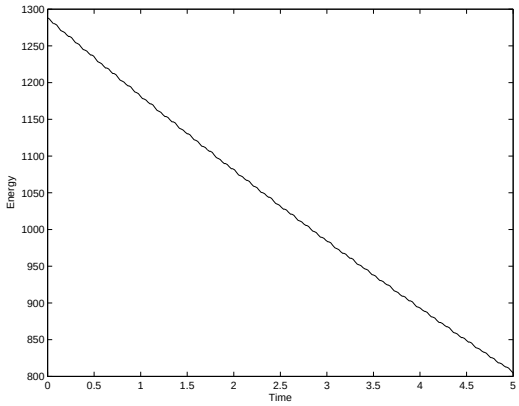
Figure 5.4: Energy of the Beam Vibration System

We now show that 1 Byzantine actuator at each location can be tolerated and asymptotic stability can be maintained by having 3 actuators at each location. The Fig. 3(b)shows the energy of the system staring from an arbitrray initial state when one actuator at each location is Byzantine.

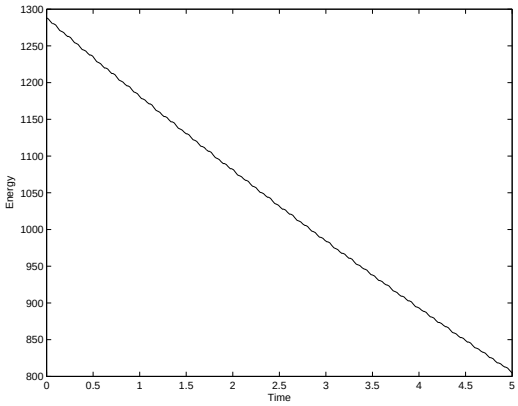
We now show that when $k = 1$, we can asymptotically stabilize the system using 4 actuators that are distributed according to the $4 - uniform$ configuration. We choose 4 pairs of colocated sensors and actuators such that the columns of B matrix have equal magnitude and successive column vectors are seperated by an angle $\frac{\pi}{4}$.

$$B = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -0.5754 & 0.1715 & 0.8179 & 0.9852 \\ -0.8179 & -0.9852 & 0.5754 & 0.1714 \end{pmatrix}$$

The following graphs show the states of the $4 - uniform$ configuration system staying asymptotically stable in the presence of no actuator faults and one actuator failing.



(a) Energy (4-uniform configuration) with No Faults



(b) Energy (4-uniform configuration) with 1 Byzantine Actuator

Figure 5.5: Energy of the 4-Uniform Configuration Beam Vibration System

5.4 Related Work

The application of active control to attenuate lower frequencies of vibration in structures has been studied from many years [56, 8]. The earlier systems were mostly centralized in control. However, the extension of this technology to large scale systems motivates the design of control that is distributed [19].

A control system designed to tolerate failures in system components while maintaining closed loop system stability and performance has been defined as a reliable control system [71]. Such systems are also called systems possessing integrity against component failures. Redundancy is a key ingredient in all such reliable control systems. A basic difference between robust control techniques and reliable control is that the former deals with small parameter variations and system model uncertainties while the latter handles more drastic changes in the control system configuration. There exist several reliable control schemes [71, 76, 18, 35, 68, 37] that provide stability in the presence of a set of failed actuators and sensors that are non responsive. However, in this chapter we have designed control schemes that guarantee stability in the presence of malfunctioning actuators which continuously offer detrimental input and thereby can lead the system to instability.

5.5 Summary

In this chapter, we designed two reliable control schemes using a local output feedback control system that maintain asymptotic stability in the presence of Byzantine actuators that continuously generate erroneous control inputs. The first scheme was designed using redundant actuators that were colocated. However, it may not be feasible to collocate actuators in all systems. The second scheme does not require the actuators to be colocated. The other advantage with the second scheme is that the required redundancy is reduced. ut in this scheme the restrictions in the choice of actuator locations increased. The design of the system becomes more complex when the number of state dimensions of the system increases. (Upper bounds for tolerating

Byzantine faults in higher dimension systems can be found in a related technical report [47]). We gave an application of both the control schemes in stabilizing a beam subjected to an initial perturbation.

We plan to extend our results on tolerating faulty actuators to systems that use centralized and decentralized state feedback. An interesting topic for future study is also to design reliable control schemes based on adaptive control laws using state feedback. Extending some heuristic studies in this area [42] to sufficient conditions is a subject of ongoing work.

CHAPTER 6

CONCLUDING REMARKS

6.1 Contributions

This dissertation has addressed the challenge of designing reliable applications (that continue to grow in scale and complexity) using wireless sensor networks (that continue to be resource constrained and unreliable in many ways). In order to address this challenge, we have adopted a joint co-design of application and network layers, and we propose the design and implementation of a network abstraction layer that bridges the gap between the application and the network and provides performance guarantees to enable design of robust applications. our approach is to identify and specify the requirements of the application from the network in terms of suitable network abstractions and then to implement these abstractions. The implementation of the abstractions is performed by either using middleware services that are programs running in the network or by simply designing the network by choosing the right density or placement of nodes.

In this dissertation, we have designed reliable applications for classification, distributed tracking and distributed control using wireless sensor networks. In the context of each of these applications, this dissertation has made specific contributions in terms of application and network design. We list these below.

6.1.1 Snapshot services for wireless sensor networks

We have designed generalized snapshot services for arbitrary sensor networks with N nodes that reside in an f-dimensional space, where each node periodically generates m bits of information. Global state snapshots are a fundamental primitive for wireless networks that sense and control real environments. Applications often require that they be consistent and timely, which makes them potentially costly. Cost reduction is often realized by gathering only a "delta" from previous snapshots. We have explored an alternative form of efficiency by generalizing the notion of a snapshot to satisfy *distance sensitivity* properties, wherein the state of nearby nodes is available with greater resolution, speed, and frequency than that of farther away nodes.

Our algorithms can be formulated to allow delivery at a subset of nodes as opposed to all nodes. They are memory efficient and realizable in networks with irregular density, with arbitrary sized holes, imperfect clustering, and non unit disk radios. We have quantified the maximum rate at which information can be generated at each node so that snapshots are periodically delivered across the network. We have specified the allowable aggregation functions in abstract terms, allowable functions include average, max, min and wavelet functions. For our services, global time synchronization is not required; a local notion of time however is needed to ensure fair scheduling of transmission of nodes.

We have shown how our snapshot service can be used in a distributed pursuer evader tracking application, where one or more pursuers are required to eventually catch all the evaders in a region.

6.1.2 Trail: Sensor network service for distributed object tracking

We have presented a wireless sensor network service, *Trail*, that supports distance sensitive tracking of mobile objects for in-network subscribers upon demand. *Trail* achieves a *find* time that is linear in the distance from a subscriber to an object, via a distributed data structure that is updated only locally when the object moves. Notably, *Trail* does not partition the network into a hierarchy of clusters and clusterheads, and as a result *Trail* has lower maintenance costs, is more locally fault-tolerant and it better utilizes the network in terms of load balancing and minimizing the size of the data structure needed for tracking. Moreover, *Trail* is reliable, and energy-efficient, despite the network dynamics that are typical of wireless sensor networks. *Trail* can be refined by tuning certain parameters, thereby yielding a family of protocols that are suited for different application settings such as rate of queries, rate of updates and network size.

We have shown how *Trail* can be used to support a distributed pursuer evader tracking application, commonly known as an asset protection game. We have evaluated the performance of *Trail* by analysis, simulations in a 90-by-90 sensor network, and experiments on 105 Mica2 nodes in the context of the pursuer-evader control application.

6.1.3 Influence field based classification and tracking using wireless sensor networks

We considered the problem of reliably estimating the influence fields of different target types in a wireless sensor network subject to a variety of faults. We provided mechanical procedures for sensor node density selection as well as algorithmic techniques appropriate for dealing with each fault class. Corroboration of our results and techniques was provided through at-scale experiments. We showed how reliable estimation was achieved to enable accurate classification and tracking in *A Line In The Sand*.

6.1.4 Reliable control system design despite Byzantine actuators

Motivated by the application of wireless sensor networks in distributed control of flexible structure, we designed two reliable control schemes using a local output feedback control system that maintain asymptotic stability in the presence of Byzantine actuators that continuously generate erroneous control inputs. The first scheme was designed using redundant actuators that were collocated. However, it may not be feasible to collocate actuators in all systems. The second scheme does not require the actuators to be collocated. The other advantage with the second scheme is that the required redundancy is reduced. But in this scheme the restrictions in the choice of actuator locations increased. The design of the system becomes more complex when the number of state dimensions of the system increases. (Upper bounds for tolerating Byzantine faults in higher dimension systems can be found in a related Technical report [47]). We gave an application of both the control schemes in stabilizing a beam subjected to an initial perturbation.

6.2 Future Work

This dissertation has identified the need to jointly design the application and network layers so that (1)applications are designed in such a way that they are feasible to implement given the network constraints (2) applications can be adaptive to varying network conditions (3) the network can operate in such a way that the application requirements are met and yet the system is optimal in its energy usage. By providing network abstractions, applications can be designed without directly coupling them with the network and therefore without exposing low level parameters such as those of the MAC layer, thereby simplifying the application design. Some of the future extensions of this work are as follows.

Adaptive Systems: In this dissertation, we have mostly looked at static configurations where given a specification, network abstractions are implemented. Parameterizations for different application settings are done apriori. We would like to consider adaptive system design, where the application and network layers are able to tune dynamically to optimize overall system efficiency. This involves the application layer and the middleware services to simultaneously tune themselves and dynamically agree upon parameters.

Mobile wireless networks: While this dissertation has considered applications of sensor networks where the network is static, of late sensors and actuators are integrated into mobile objects such as cell-phones, PDAs and even humans and animals. Mobility adds to the challenge of reliable application design and poses interesting problems in the design of middleware services. Examples include location services that are not dependent on static anchors and reliable transport services that can tolerate temporary disconnections and slow changing topologies. The location service

is aimed at providing low cost solutions for people/object tracking in indoor environments and also to serve as a localization framework for other middleware services such as routing that can run on top of it. For groups working in disaster relief situations or in remote and inaccessible regions such as forest workers and underground miners, the location service and the reliable transport service will enable sharing basic communication when other communication infrastructure is destroyed or not available.

Control systems: This dissertation has also focused on control applications using wireless sensor networks such as distributed vibration control. WSNs are ideally suited for control of distributed parameter systems because of the scale at which they can operate. Wireless networks also remove the need for cumbersome wiring in industrial and process control applications. But when wireless networks are used for control, reliability, low latency and security will be extremely critical. There exists plenty of related work on network based control, that analyze the stability and performance of control systems in the presence of latency and data losses. These studies have focused on high bandwidth networks. How can these results be ported to wireless networks? More importantly, what applications are feasible using wireless networks? Can control strategies be relaxed to accommodate wireless networks? These are interesting questions related to control systems using wireless networks.

High level specification: One of the contributions in this dissertation has been the simplification of application design. We have achieved it in our work by abstracting away the network details. The idea is that the application designers for wireless sensor networks need not be networking experts. They should not be involved in the design and implementation of the network. A future work in this direction is a high level

specification language for the application that results in translation to network level specification and dynamically connecting to the appropriate abstraction.

Heterogeneity: As wireless sensor networks move into the urban domain, applications have to be developed across varied sensor networks. We characterize the urban sensing problem as one where, in response to a specific requirement (such as tracking and subsequent capture of an assailant), an application needs to be developed where (a) the operations of varied, independent sensor networks are integrated and (b) the information from these networks suitably retrieved, analyzed and fused to meet the application. This points to not only a network abstraction oriented approach for system design, but also standardization of these abstractions. As initial work in this direction, we have recently prosed an architecture for composing applications across heterogeneous sensor networks [48].

APPENDIX A

PROOF OF MAXIMA OF EXPRESSION IN EQ. 3.11

Proposition A.0.1. *Let $f(\theta, \phi)$ be defined as in Eq. A.1.*

$$f(\theta, \phi) = \frac{(sin(\theta) + sin(\phi))}{sin(\theta + \phi)} \tag{A.1}$$

The maximum value of $f(\theta, \phi)$ where $\theta > 0$, $\phi > 0$, and $0 < (\theta + \phi) \leq \alpha$ is $sec(\frac{\alpha}{\sqrt{2}})$ and occurs when $\theta = \phi = \frac{\alpha}{2}$.

Proof. To simplify, we use the following transformation matrix.

$$x = \frac{\theta + \phi}{\sqrt{2}}$$
$$y = \frac{-\theta + \phi}{\sqrt{2}}$$

where, $0 < x \leq \frac{\alpha}{\sqrt{2}}$ and $|y| \leq x$.

Based on the above transformation, $f(\theta, \phi)$ can be written in terms of x, y as follows:

$$g(x, y) = \frac{(sin(\frac{x+y}{\sqrt{2}}) + sin(\frac{x-y}{\sqrt{2}}))}{sin(\sqrt{2} * x)} \tag{A.2}$$

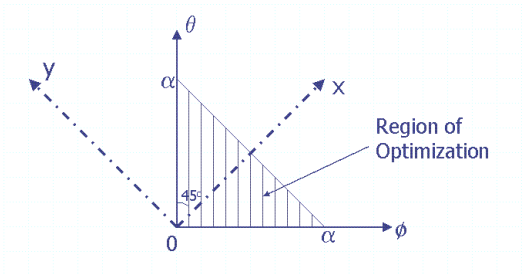


Figure A.1: Finding maxima for $f(\theta, \phi)$

Now we first find the value of y at which function $g(x, y)$ is maximum given a value of x. For this we differentiate $g(x, y)$ partially with respect to y and equate the result to 0.

$$\begin{aligned} & \frac{\partial g(x,y)}{\partial y} = 0 \\ \Rightarrow \quad & \frac{(cos(\frac{x+y}{\sqrt{2}}) - cos(\frac{x-y}{\sqrt{2}}))}{\sqrt{2} * sin(\sqrt{2} * x)} = 0 \\ \Rightarrow \quad & y = 0 \end{aligned}$$

Thus we find that for function $g(x, y)$ at any given value of x, $y = 0$ is the only stationary point since $x > 0$. Differentiating $g(x, y)$ partially twice with respect to y, we note that result is less than 0 when $y = 0$.

$$\frac{\partial^2 g(x,y)}{\partial y^2}\Big|_{y=0} = -\frac{1}{cos(\frac{x}{\sqrt{2}})} < 0$$

Thus for any given value of x, $g(x, y)$ is maximum when $y = 0$.

Also, when $y = 0$, $g(x, y) = \frac{1}{cos(\frac{x}{\sqrt{2}})}$ which increases monotonically when $x > 0$ and since $x \leq \frac{\alpha}{\sqrt{2}}$, the maximum value is $sec(\frac{\alpha}{\sqrt{2}})$ and the maximum occurs when $x = \frac{\alpha}{\sqrt{2}}$, i.e, $(\theta + \phi) = \alpha$.

□

APPENDIX B

TECHNICAL NOTE: ON TERMINATING POINTS FOR TRACKING MOBILE OBJECTS

In this section, we define the notion of *terminating points* for schemes that track mobile objects in a distance sensitive manner in terms of *update* and *find*. We also analyze the tradeoffs with respect to the choice of terminating points.

Let O be a set of mobile objects in a network of size $N \times N$. Mobile objects are of two types: $finder$ objects (O_f) and *mover* objects (O_m). Thus O is a union of disjoint sets O_f and O_m. Let p denote the location of any object P. Let $Tracker$ be a distance sensitive tracking scheme. $Tracker$ maintains a track $track_P$ for every object P that belongs to O_m. $track_P$ is a set of points that contain information pertaining to P. This information could be the actual state of P or simply a pointer following which leads to the actual state of P. The length of $track_P$ is the length of the shortest curve connecting all points in $track_P$. $Tracker$ offers two functions: $find(P, Q)$, that returns state of P to Q, where Q belongs to O_f and P belongs to O_p; and $move(P, p', p)$ that updates $track_P$ when P moves from p' to p. $Tracker$ satisfies property F (*find* distance sensitivity) and property U (*update* distance sensitivity) stated below:

Definition 17 (*find* distance sensitivity)**.** $Tracker$ satisfies property F if the cost of $find(P, Q)$ grows linearly with $dist(p, q)$.

Definition 18 (*update* distance sensitivity)**.** $Tracker$ satisfies property U if the cost of $move(P, p', p)$ grows linearly with $dist(p, p')$.

Note that in this document we consider only *discrete* moves of the objects. When the motion of the object is continuous, a subset of $Tracker$ schemes may have a property that the cost of move is proportional to the distance of move in an amortized sense.

From here on we assume that there is only one *mover* in the network and track for that object is represented as *track* and we drop the identity subscript. We refer to *track* for the *mover* at location x in the network simply as *track* for point x.

Definition 19 (Terminating Set τ)**.** A terminating set τ in $Tracker$ is *a* smallest set of points such that *track* for every point in the network passes through at least one point in τ.

The cardinality of a terminating set τ is denoted as μ_τ. The points contained in a terminating set are called as terminating points of that set. Note that there could also be multiple terminating sets in the network with each set containing an equal but any number of points. In other words there are multiple smallest sets of points such that *track* for every point in the network passes through at least one point in each of those sets. For example in the case of *Trail* and *Stalk* [24], there is only terminating set and this set has exactly one point, namely the center of the network. Thus $\tau = \{C\}$. In *DSIB* [27] and *LLS* [1], each set containing one publish location at the highest level constitutes a terminating set. Thus in those schemes there are

multiple terminating sets but the cardinality of each set is 1. In the simple horizontal vertical double ruling scheme, each horizontal line is a terminating set because every vertical track passes through at least one point in each of those sets.

Lemma B.0.2. *A terminating set τ cannot be an empty set.*

Proof. *track* for each point in the network is at least equal to the point itself. Thus τ cannot be empty. □

In *Trail*, we have chosen a unique terminating set consisting of a unique terminating point. We now analyze the tradeoffs involved when the terminating set is not unique and when a terminating set contains more terminating points. We consider 2 cases: a single terminating set with multiple terminating points and multiple terminating sets each with one terminating point.

Case 1:$\mu_\tau \geq 1$ We note that it is possible to decrease the maximum track length in the network by dividing the network into regions and tracks being maintained with respect to a terminating point in each region. (The disproportionate updates that can be caused when an object keeps switching between boundaries can be avoided by making the regions overlap.) The question then arises as to how small the regions can be and yet maintain *distance sensitivity*. We now analyze the limits for decreasing the track length and its effect on maximum *find* cost.

Given any location f of a finder, let L_f denote the length of the *find* trajectory traversed from the finder location, after which the *track* for any point in the network is found. Thus L_f denotes the worst case *find* cost from location f. Let $\hat{L_f}$ denote the maximum value of L_f in the network.

Lemma B.0.3. *For F to hold, $\hat{L_f} = O(N)$.*

Proof. Note that the maximum distance between any two points in the network is $O(N)$. The result follows. □

Lemma B.0.4. *$\hat{L_f}$ is at least equal to the length of traversing all points in τ.*

Proof. From the definition of a terminating set, τ is a minimum set of points through which tracks from all points pass through. □

Using Lemma $A.6$, we state the following Theorem.

Theorem B.0.5. *The maximum* find *cost in the network is minimized when $\mu_\tau = 1$.*

Proof. In the worst case, a *find* trajectory has to traverse all terminating points in the terminating set. Compared to any configuration of terminating points when $\mu_\tau > 1$, there exists a configuration of the terminating point when $\mu_\tau = 1$, such that $\hat{L_f}$ is lower. □

Using Lemmas $A.5$ and $A.6$, we get the following Lemma.

Lemma B.0.6. *All terminating points in τ must be traversable in $O(N)$.* □

Tha above Lemma imposes a lower bound the maximum size of the regions in the following way. Let a region $\mathcal{R}_t$ be a set of points that choose t as a terminating point. Let $\rho_\mathcal{R}$ denote the distance of the farthest point in the region from t. Let $\hat{\rho_\mathcal{R}}$ denote the maximum distance from a terminating point in any region of the network. We state the following Theorem.

Theorem B.0.7. *In order to preserve F, $\hat{\rho_\mathcal{R}} = \Omega(N)$.*

Proof. Recall that all terminating points in τ must be traversed in $O(N)$. If $\hat{\rho_\mathcal{R}} < \Omega(N)$, then the maximum diameter of any region in the network is less than $\Omega(N)$ in

the $N \times N$ network and therefore all terminating points in τ cannot be traversed in $O(N)$. □

Summary: The maximum track length in the network depends on the size of the largest region. We note the maximum track length can be decreased by dividing the network into regions and having local terminating points per region. However, the size of the largest region can only be a constant order less than the diameter of the network. For instance dividing the network into infinitesimally small regions or even regions of size $O(\sqrt{N})$ will violate F. Also when $\mu_\tau > 1$, *find* has to traverse all the points in τ in the worst case. Thus the maximum *find* cost in the network increases. □

Case 2: Multiple terminating sets: Now we consider the case where there are multiple terminating sets each with one terminating point and analyze the tradeoffs in *find* and *update* cost. If there are multiple terminating sets then it is sufficient for *find* to traverse the terminating point in any such set. In this case there is a likelihood of decreasing the maximum *find* cost when compared to having only set of terminating points because tracks can be *found* by reaching a terminating point in any of the terminating sets. An example of this is the case where all tracks pass through a common set of points as opposed to just one common point. Thus the cardinality of each terminating point set is 1 because all tracks pass through each point, but there are multiple such points.

Similar to case 1, we can show that the maximum *find* cost can be decreased by dividing the network into regions and having one terminating set per region. We state the following Theorems whose proofs are similar to that of case 1.

Theorem B.0.8. *The maximum update cost in the network is minimized when the number of terminating sets is* 1. □

Theorem B.0.9. *In order to preserve U, $\hat{\rho_{\mathcal{R}}} = \Omega(N)$, where $\hat{\rho_{\mathcal{R}}}$ is the maximum distance of any point in the network from the terminating point in its local terminating set.* □

Summary: The maximum *find* cost in the network depends on the size of the largest region. We note the maximum *find* length can be decreased by dividing the network into regions and having local terminating points per region. However, the size of the largest region can only be a constant order less than the diameter of the network. For instance dividing the network into infinitesimally small regions or even regions of size $O(\sqrt{N})$ will violate F. Also when the number of terminating sets is greater than 1, *update* has to traverse the terminating point in all terminating sets in the worst case. Thus the maximum *update* cost in the network increases. □

Choice of unique terminating point: Maintaining a track with respect to local terminating points and a single terminating set could be advantageous if it is more likely that querying object and the object being found are closer and therefore it is unlikely that all terminating points in the set have to be traversed. Similarly maintaining a track with respect to multiple terminating sets could be advantageous if objects are likely to move within bounded regions within a network. In this paper we consider all distances between querying object and tracked object to be equally likely and do not restrict mobility of the objects. Hence we consider only the case where there is a unique terminating point, namely C.

Note that even if the network is divided into a constant number of regions, the concept of maintaining a track with respect to any terminating point is the same

as in *Trail*. Note also that the tracks that are maintained in *Trail* with respect to the terminating point are *tight* with a stretch factor that is less than 1.2 times the shortest distance to the terminating point.

Multiple terminating sets with multiple terminating points: The cases that we have presented can be extended to the case of multiple terminating sets, each with multiple terminating points when there is more knowledge of *find* and *update* patterns within each region.

BIBLIOGRAPHY

[1] I. Abraham, D. Dolev, and D. Malkhi. LLS: A Locality Aware Location Service for Mobile Ad-hoc Networks. In *Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*, pages 75–84, Philadelphia, USA, 2004. ACM.

[2] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita. A Line in the Sand: A wireless Sensor Network for Target Detection, Classification, and Tracking. *Computer Networks, Special Issue on Military Communications Systems and Technologies*, 46(5):605–634, July 2004.

[3] A. Arora, E. Ertin, R. Ramnath, M. Nesterenko, and W. Leal. Kansei: A High Fidelity Sensing Testbed. *IEEE Internet Computing*, 10(2):35–47, March 2006.

[4] A. Arora and R. Ramnath. ExScal: Elements of an Extreme Wireless Sensor Network . In *The 11th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 102–108, HongKong, 2004.

[5] B. Awerbuch and D. Peleg. Online Tracking of Mobile Users. *Journal of the Associsation for Computing Machinery*, 42:1021–1058, 1995.

[6] C. Bailey-Kellogg and F. Zhao. Influence-based model decomposition for reasoning about spatially distributed physical systems. *Artificial Intelligence*, 130:125–166, 2001.

[7] M. J. Balas. Direct output feedback control of large space structures. *Journal of Astronautical Sciences*, 27(2):157–180, 1979.

[8] M. J. Balas. Direct velocity feedback control of large space structures. *Journal of Guidance and Control*, 2(3):252–253, 1979.

[9] S. Bapat, V. Kulathumani, and A. Arora. Analyzing the Yield of ExScal, a Large Scale Wireless Sensor Network Experiment. In *13th IEEE International Conference on Network Protocols*, 2005.

[10] S. Bapat, V. Kulathumani, and A. Arora. Reliable Estimation of Influence Fields for Classification and Tracking in an Unreliable Sensor Network. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS)*, 2005.

[11] M. Blanke, R. I. Zamanabadi, and Bogh. Fault tolerant control systems, a holistic view. *Control Engineering Practice*, 5(5):693–702, 1997.

[12] "Boeing". Challenge problem description for network embedded software technology (nest). Boeing Tech Report, The Boeing Company, St. Louis, MO 63166, April 2002.

[13] C. Bonivento, A. Paoli, and L. Marconi. Fault-tolerant control for a ship propulsion system. In *European Control Conference*, Porto, Portugal, 2001.

[14] D. Braginsky and D. Estrin. Rumor Routing Algorithm for Sensor Networks. In *First ACM Workshop on Wireless Sensor Networks and Applications*, pages 22–31. ACM, 2002.

[15] H. Cao, E. Ertin, V. Kulathumani, M. Sridharan, and A. Arora. Differential Games in Large Scale Sensor actuator Networks. In *Information Processing in Sensor Networks (IPSN)*, pages 77–84. ACM, 2006.

[16] Michael J. Caruso and Lucky S. Withanawasam. Vehicle detection and compass applications using AMR magnetic sensors, AMR sensor documentation. "http://www.magneticsensors.com/datasheets/amr.pdf".

[17] M. Chandy and L. Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Transacttions on Computer Systems*, 3(5):63–75, 1985.

[18] T. Chang and Z. Chen. Reliable control for systems with block diagonal feedback structures. In *American Control Conference*, volume 2, pages 829–833, Chicago, IL, USA, June 2000.

[19] D. Šiljak. *Decentralized Control of Complex Systems*. Academic Press, New York, 1991.

[20] T. Dang, N. Bulusu, and W. Feng. Robust Information Driven Data Compression Architecture for Irregular WSN. In *EWSN*, 2007.

[21] Ashwin D'Costa and Akbar Sayeed. Collaborative signal processing for distributed classification in sensor networks. *The 2nd International Workshop on Information Processing in Sensor Networks (IPSN '03)*, pages 193–208, 2003.

[22] F. Dehne, A. Ferreira, and A. Rau-Chaplin. Parallel fractional cascading on hypercube multiprocessors. In *Computational Geometry Theory Applications*, volume 2, pages 144–167, 1992.

[23] M. Demirbas, A. Arora, and V. Kulathumani. Glance: A Light Weight Querying Service for Sensor Networks. In *International Conference on Principles of Distributed Systems (OPODIS)*, pages 242–257, 2006.

[24] M. Demirbas, A. Arora, T. Nolte, and N. Lynch. A Hierarchy-based Fault-local Stabilizing Algorithm for Tracking in Sensor Networks. In *International Conference on Principles of Distributed Systems (OPODIS)*, pages 143–162, 2004.

[25] S. Dolev, D. Pradhan, and J. Welch. Modified Tree Structure for Location Management in Mobile Environments. In *INFOCOM*, pages 530–537, 1995.

[26] Marco Duarte and Yu-Hen Hu. Vehicle classification in distributed sensor networks. *Journal of Parallel and Distributed Computing*, 2004.

[27] S. Funke, L. Guibas, A. Nguyen, and Y. Wang. Distance Sensitive Information Brokerage in Sensor Networks. In *International Conference on Distrbuted Computing in Sensor Systems DCOSS*, pages 234–251. Springer-Verlag, 2006.

[28] J. Gao, L.J. Guibas, J. Hershberger, and L. Zhang. Fractionally cascaded information in a sensor network. In *IPSN*, pages 311–319, 2004.

[29] R. Hall. *Introduction to the Theory of Coverage Processes*. Wiley, 1988.

[30] T. He, S. Krishnamurthy, and J. Stankovic. VigilNet: An Integrated Sensor Network System for Energy-Efficient Surveillance. *ACM Transactions on Sensor Networks*, 2(1):1–38, 2006.

[31] Mark Hewish. Reformatting fighter tactics. *Jane's International Defense Review*, 2001.

[32] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[33] J. P. How. Local control design methodologies for a hierarchic control architecture. Master's thesis, Dept. of Aeronautics and Astronautics, MIT, Cambridge, 1990.

[34] J. P. How. Local control design methodologies for a hierarchic control architecture. *Journal of Guidance, Control and Dynamics*, 15(3):654–663, 1992.

[35] S. Huang, J. Lam, and B. Chen. Local reliable control for linear systems with saturating actuators. In *Proceedings of 41st IEEE Conference on Decision and Control*, pages 4154–4159, Las Vegas, Nevada, Dec 2002.

[36] C. Intanogonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed Diffusion for Wireless Sensor Networking. *IEEE Transactions on Networking*, 11(1):2–16, 2003.

[37] J. Jiang and Q. Zhao. Design of reliable control systems possessing actuator redundancies. *AIAA Journal of Guidance, Control and Dynamics*, 23(4):709–718, 2000.

[38] B. Karp and H. T. Kung. Greedy Perimeter Stateless Routing for Wireless Networks. In *Proceedings of Mobile Computing and Networking, MobiCom*, pages 243–254, 2000.

[39] G. Karsai, G. Biswas, T. Pasternak, and S. Narasimhan. Fault-adaptive control: a CBS application. In *8th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, Washington DC, April 2001.

[40] D. Kempe, J. M. Kleinberg, and A. J. Demers. Spatial gossip and resource location protocols. In *ACM Symposium on Theory of Computing*, pages 163–172, 2001.

[41] S. Kim. Sensor networks for structural health monitoring. Master's thesis, University of California, Berkeley, 2005.

[42] Y. M. Kim, A. Arora, and V. Kulathumani. Local distributed control of linear systems despite byzantine faults. OSU Tech Report OSU-CISRC-7/03-TR42, The Ohio State University, Columbus, Ohio 43210, July 2003.

[43] Y. M. Kim, A. Arora, and V. Kulathumani. On Effect of Faults in Vibration Control of Fairing Structures. In *Fifth International Conference on Multibody Systems, Nonlinear Dynamics and Controls (MSNDC)*, 2005.

[44] X. Koutsoukos, F. Zhao, H. Haussecker, J. Reich, and P Cheung. Fault modelling for monitoring and diagnosis of sensor-rich hybrid systems. In *Proc. IEEE Conference on Decision and Control*, Orlando Fl, 2001.

[45] Bhaskar Krishnamachari and Sitharama Iyengar. Distributed bayesian algorithms for fault-tolerant event region detection in wireless sensor networks. *IEEE Transactions on Computers*, 53(3):241–250, 2004.

[46] V. Kulathumani, M. Demirbas, and A. Arora. Trail: A Distance Sensitive WSN Service for Distributed Object Tracking. In *European Conference on Wireless Sensor Networks*, 2007.

[47] V. Kulathumani, Y. M. Kim, and A. Arora. Reliable control system design despite byzantine actuators. OSU Tech Report OSU-CISRC-3/05-TR13, The Ohio State University, Columbus, Ohio 43210, March 2005.

[48] V. Kulathumani, M. Sridharan, A. Arora, and R. Ramnath. Weave: An Architecture for Composing Urban Sensing Applications across Multiple Sensing Fabrics. In *Workshop on Mobile Devices and Urban Sensing (MODUS)*, 2008.

[49] S. Kulkarni and U. Arumugam. TDMA service for Sensor Networks. In *ICDCS*, volume 4, pages 604–609, 2004.

[50] Dan Li, Kerry Wong, Yu Hu, and Akbar Sayeed. Detection, classification and tracking of targets in distributed sensor networks. *IEEE Signal Processing Magazine*, 19(2):17–29, 2002.

[51] J. Liu, J. Reich, and F. Zhao. Collaborative in-network processing for target tracking. *Journal on Applied Signal Processing*, 2002.

[52] X. Liu, Q. Huang, and Y. Zhang. Combs, Needles, Haystacks: Balancing Push and Pull for Discovery in Large-Scale Sensor Networks. In *ACM Conference On Embedded Networked Sensor Systems (Sensys)*, pages 122–133. ACM, 2004.

[53] C. Lu, G. Xing, O. Chipara, C.-L. Fok, and S. Bhattacharya. A Spatiotemporal Query Service for Mobile Users in Sensor Networks. In *IEEE International Conference on Distributed Computing Systems*, pages 381–390. IEEE Computer Society, 2005.

[54] Donal McErlean and Shrikanth Narayanan. Distributed detection and tracking in sensor networks. *36th Asilomar Conf. Signals, Systems and Computers*, 2002.

[55] "C. Meesookho, S. Narayanan, and C. S. Raghavendra". "collaborative classification applications in sensor networks". In *"Second IEEE Sensor Array and Multichannel Signal Processing Workshop"*, "2002".

[56] L. Meirovitch. *Dynamics and Control of Structures*. John Wiley and Sons, New York, 1990.

[57] V. Mittal, M. Demirbas, and A. Arora. Loci: Local clustering service for large scale wireless sensor networks. Technical Report OSU-CISRC-2/03-TR07, The Ohio State University, 2003.

[58] V. Naik and A.Arora. Harvest: A reliable bulk data collection service for large scale wireless sensor networks. Technical Report OSU-CISRC-4/06-TR37, The Ohio State University, 2006.

[59] J. Nash. Noncooperative games. *Annals of Mathematics*, 54:286–295, 1951.

[60] H. Park, J. Burke, and M. Srivastava. Design and Implementation of a Wireless Sensor Network for Intelligent Light Control. In *6th International Conference on Information Processing in Sensor Networks, IPSN/SPOTS*, 2007.

[61] R. J. Patton, P. Frank, and R. Clark. *Fault Diagnosis in Dynamic Systems: Theory and Applications*. Prentice Hall, New Jersey, 1989.

[62] G. Pei, M. Gerla, and T.-W. Chen. Fisheye State Routing in Mobile Adhoc Networks. In *ICDCS Workshop on Wireless Networks*, pages 71–78, 2000.

[63] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A geographic hash table for data-centric storage. In *First ACM Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 12–21. ACM, 2002.

[64] N. Sandell, P. Varaiya, M. Athans, and M. Safanov. Survey of decentralized control methods for large scale systems. *IEEE Transactions on automatic Control*, 23(2):108–128, 1978.

[65] Rik Sarkar, Xianjin Zhu, and Jie Gao. Hierarchical Spatial Gossip for MultiResolution Representations in Sensor Networks. In *IPSN*, pages 311–319, 2007.

[66] S. Servetto. Sensing lena - massively distributed compression of sensor images. In *IEEE ICIP*, 2003.

[67] J. Shin, L. Guibas, and F. Zhao. A Distributed Algorithm for Managing Multi-Target Indentities in Wireless Ad Hoc Networks. In *International Workshop on Information processing in Sensor Networks IPSN*, pages 223–238, 2003.

[68] H. Shousong and H. Weili. Decentralized output feedback fault-tolerant control for uncertain large-scale systems. In *Proceedings of IEEE International Conference on Industrial Technology*, pages 20–23, Dec 1994.

[69] B. Sinopoli, C. Sharp, L. Schenato, S. Schaffert, and S. Sastry. Distributed Control Applications within Sensor Networks. In *Proceedings of the IEEE*, volume 91, pages 1235–46, Aug 2003.

[70] Vanderbilt University. JProwler, Discrete Event Simulator for Wireless Networks. http://www.isis.vanderbilt.edu/Projects/nest/jprowler/index.html.

[71] R. J. Veillette, J.V.Medanic, and W.R.Perkins. Design of reliable control systems. *IEEE Transactions on Automatic Control*, 37:290–304, 1992.

[72] S. Wang and E. J. Davison. On the stabilization of decentralized control systems. *IEEE Transactions on Automatic Control*, 18(5):473–478, 1973.

[73] Honghai Zhang and Jennifer C. Hou. On deriving the upper bound of alpha-lifetime for large sensor networks. *Proceedings of ACM Mobihoc 2004*, 2004.

[74] F. Zhao, J. Liu, J. Liu, L. Guibas, and J. Reich. Collaborative signal and information processing: An information directed approach, 2003.

[75] Feng Zhao, Jaewon Shin, and James Reich. Information-driven dynamic sensor collaboration for tracking applications. *IEEE Signal Processing Magazine*, 2002.

[76] Q. Zhao, Y. Weng, and J. Jiang. Design of reliable control systems with multiple sensors. In *Proceedings of the 11th Canadian Conference on Electrical and Computer Engineering*, volume 1, pages 225–228, Toronto, Canada, May 1998.