University of Massachusetts Amherst
ScholarWorks@UMass Amherst

Open Access Dissertations

9-2009

Sensor Control and Scheduling Strategies for Sensor Networks

Victoria U. Manfredi
University of Massachusetts Amherst, victoria.manfredi@gmail.com

Follow this and additional works at: https://scholarworks.umass.edu/open_access_dissertations

 Part of the Computer Sciences Commons

Recommended Citation
Manfredi, Victoria U., "Sensor Control and Scheduling Strategies for Sensor Networks" (2009). *Open Access Dissertations*. 105.
https://scholarworks.umass.edu/open_access_dissertations/105

This Open Access Dissertation is brought to you for free and open access by ScholarWorks@UMass Amherst. It has been accepted for inclusion in Open Access Dissertations by an authorized administrator of ScholarWorks@UMass Amherst. For more information, please contact scholarworks@library.umass.edu.

SENSOR CONTROL AND SCHEDULING STRATEGIES FOR SENSOR NETWORKS

A Dissertation Presented

by

VICTORIA U. MANFREDI

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2009

Department of Computer Science

© Copyright by Victoria U. Manfredi 2009

All Rights Reserved

SENSOR CONTROL AND SCHEDULING STRATEGIES FOR SENSOR NETWORKS

A Dissertation Presented

by

VICTORIA U. MANFREDI

Approved as to style and content by:

Jim Kurose, Chair

Andrew G. Barto, Member

Deepak Ganesan, Member

Weibo Gong, Member

Don Towsley, Member

Andrew G. Barto, Department Chair
Department of Computer Science

For my mother, father, and brother.
Res ipsa loquitur.

ACKNOWLEDGMENTS

I extend my deepest gratitude to Jim Kurose who has been an outstanding advisor and role model. His lucid insights into complex problems have been a continual inspiration. It has been my great pleasure to work with him.

I am also indebted to my committee members: Don Towsley for his insightful questions, Deepak Ganesan for encouraging my research forays into sensor networks, Andy Barto for providing a machine learning perspective, and Weibo Gong for his enthusiasm for varied research problems. I also thank Robert Hancock, Naceur Malouch, and Mike Zink for lively and interesting research collaborations. I am grateful to Patrick Thiran and Matt Grossglauser for hosting me at EPFL and focusing my research, and to Matt for his valiant efforts at punctuality. I also thank Sridhar Mahadevan for introducing me to the rich world of statistical machine learning.

My life at UMass was made immeasurably easier by the cheerful efficiency of Laurie Connors, Susan Lanfare, Leanne Leclerc, and Tyler Trafford.

I thank the members of the Networks and ALL labs at UMass, and of CASA, for the wonderful research discussions and camaraderie over the years, particularly Yung-Chih Chen, Daniel Figueiredo, Majid Ghaderi, Mohammad Ghavamzadeh, Yu Gu, Bo Jiang, Ramin Khalili, George Konidaris, Patrick Lee, Junning Liu, Eric Lyons, Daniel Menasche, Giovanni Neglia, Sarah Osentoski, Dave Pepyne, Bruno Ribeiro, Guto Rocha, Khash Rohanimanesh, Elisha Rosensweig, Suchi Saria, Ash Shah, Özgür Şimşek, Pablo Serrano, Suddu Vasudevan, Bing Wang, David Westbrook, Pippin Wolfe, Sookhyun Yang, and Chun Zhang. Finally, I thank Lisa Friedland, Ann Guo, Emily Horrell, Geetika Lakshmanan, Natasha Mohanty, Audrey St. John, Frosso Seitaridou, and Elif Tosun for their constant encouragement and support.

ABSTRACT

SENSOR CONTROL AND SCHEDULING STRATEGIES FOR SENSOR NETWORKS

SEPTEMBER 2009

VICTORIA U. MANFREDI

B.A., SMITH COLLEGE

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Jim Kurose

We investigate sensor control and scheduling strategies to most effectively use the limited resources of an ad hoc network or closed-loop sensor network. In this context, we examine the following three problems.

Where to focus sensing? Certain types of sensors, such as cameras or radars, are unable to simultaneously collect high fidelity data from all environmental locations, and thus require some sort of sensing strategy. Considering a meteorological radar network, we show that the main benefits of optimizing sensing over expected future states of the environment are when there are multiple small phenomena in the environment. Considering multiple users, we show that the problem of call admission control (i.e., deciding which sensing requests to satisfy) in the context of a virtualized private sensor network can be solved in polynomial time when sensor requests are

divisible or fixed in time. When sensor requests are indivisible but may be shifted in time, we show that the call admission control problem is NP-complete.

How to make sensing robust to delayed and dropped packets? In a *closed-loop sensor network*, data collected by the sensors determines each sensor's future data collection strategy. Network delays, however, constrain the quantity of data received by the time a control decision must be made, and consequently affect the quality of the computed sensor control. We investigate the value of separate handling of sensor control and data traffic, during times of congestion, in a closed-loop sensor network. Grounding our analysis in a meteorological radar network, we show that prioritizing sensor control traffic decreases the round-trip control-loop delay, and thus increases the quantity and quality of the collected data and improves application performance.

How to make routing robust to network changes? In wireless sensor and mobile ad-hoc networks, variable link characteristics and node mobility give rise to changing network conditions. We propose a routing algorithm that selects a type of routing subgraph (a braid) that is robust to changes in the network topology. We analytically characterize the reliability of a class of braids and their optimality properties, and give counter-examples to other conjectured optimality properties in a well-structured (grid) network. Comparing with dynamic source routing, we show that braid routing can significantly decrease control overhead while only minimally degrading the number of packets delivered, with gains dependent on node density.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

While many sensor networks are resource-constrained due to limited power or bandwidth, certain types of sensors, such as cameras or radars, have additional constraints due to their inability to simultaneously collect high fidelity data from all environmental locations. For such sensors, a sensing strategy is necessary to decide where to focus sensing. In a *closed-loop sensor network*, the data collected by the sensors determines each sensor's future data collection strategy. If the sensed data and subsequent sensor controls must be transmitted over the network, however, packet delays are incurred. These delays constrain the quantity of data received by the time a control decision must be made, and consequently affect the quality of the computed sensor control. Thus, network constraints may further exacerbate existing constraints on the sensing resources. In wireless and mobile ad-hoc networks, changing network conditions due to variable link characteristics or node mobility, can also constrain network resources. This thesis focuses on the design and analysis of network protocols to accommodate the limited resources and changing network conditions present in closed-loop sensor networks and wireless and mobile ad-hoc networks. In this context, we examine the following three problems.

1.1 Where to focus sensing?

We first consider the problem of controlling sensors, such as cameras and radars, that cannot simultaneously collect high fidelity data from all environmental locations. Any effective sensing strategy for such sensors must balance collecting high fidelity

data from some locations (thus implicitly collecting lower fidelity data from other locations), to ensure that the phenomena in those locations are correctly identified, with surveillance scans collecting low fidelity data from all environmental locations, to ensure that all new phenomena are eventually observed. Focusing on the CASA meteorological radar network [19, 62, 101], we compare the advantages of a myopic sensing strategy (i.e., optimizing the radar sensing strategy at each decision epoch over only the current state of the environment) with those of a lookahead sensing strategy (i.e, additionally optimizing the radar sensing strategy over expected future states). We show that the main benefits of considering expected future states of the environment are when there are multiple meteorological phenomena in the environment, and when the maximum radius of any phenomenon is sufficiently smaller than the radius of the radar's footprint. We also show that there is a trade-off between the average quality with which a phenomenon is scanned and the number of decision epochs before which a phenomenon is rescanned. Considering only scan quality, we find that a simple lookahead sensing strategy can achieve approximately the same quality as that of a full lookahead strategy.[1] In contrast to other work on radar control that focuses on hard targets such as airplanes [46], our work focuses on tracking meteorological phenomena and the time frame over which to evaluate control decisions.

Inspired by the GENI Project [27], we next consider the problem of call admission control (i.e., deciding which sensing requests to satisfy) in the context of a virtualized private sensor network (VPSN).[2] Although multiple users may share the sensing resources of a virtualized network, in a VPSN each user works with the abstraction of having its own private network. Unlike traditional virtualized resources such as

[1]See Section 2.3 for a primer on meteorological radars and sensing and Section 2.4.2 for a formal definition of quality.

[2]The GENI project [27] is developing a shared testbed for investigating future internets and is including sensor networks as part of the shared testbed: hence multiple users will compete for sensing resources.

memory, bandwidth, or CPU cycles, however, the sensor requests made by one user may completely or partially satisfy those of another user. We note that while other work has also considered the idea of a virtual sensor network [35], their focus is on how to adaptively select a subset of nodes in the sensor network to construct a "virtual sensor network" for a particular task, rather than virtualizing the sensing resources of the individual sensors. We particularly investigate a model in which a user's sensing strategy can be represented as a temporal sequence of sensor requests distributed across a time interval. We define a sensor request as a request to use a sensor in a particular way, possibly at a particular time, with some associated user utility. The call admission control problem for VPSNs is to select a non-interfering subset of sensor requests with maximum utility from among all of the sensing strategies. We show that the call admission control problem in VPSNs can be solved in polynomial time when sensor requests are divisible or fixed in time. When sensor requests are indivisible but may be shifted in time, we show that the VPSN call admission control problem is NP-complete.

1.2 How to make sensing robust to delayed and dropped packets?

We next consider the problem of transmitting both sensor control and data packets in the presence of network congestion. In a sensor network, congestion can arise due to bursty and high-bandwidth data traffic, combined with wireless links and many-to-one data routing to a sink. Delayed and dropped packets then degrade the performance of the sensing application. In a closed-loop sensor network, the sensed data transmitted through the network may have considerable redundancy in both time and space, making application performance somewhat insensitive to data packet loss and delay. Conversely, performance is typically much more sensitive to loss or delay of sensor control packets, since these packets carry the application's

sensor commands generated in response to received data. We investigate the value of separate handling of sensor control and data traffic, during times of congestion, in a closed-loop sensor network. While previous work, e.g., [50], focuses on the effects of prioritizing network control packets, we focus on the effects of prioritizing sensor control packets. Whereas network control affects what data is transmitted and at what rate, sensor control additionally affects what data is sensed and thus available to be transmitted.

We first show that prioritizing sensor control traffic over data traffic decreases the round-trip control-loop delay, and consequently increases the quantity and quality of the data collected by the network. We then ground our analysis in a storm-tracking application in the context of the CASA radar network [19, 62, 101]. The application measures reflectivity (a measure of the number of scatterers in a unit volume of atmosphere known as a voxel) and tracks storms (i.e., regions of high reflectivity) using a Kalman filter. Considering data quantity, we show that prioritizing sensor control traffic increases the number of voxels, V, that can be scanned given a constant number of reflectivity samples, N_c, obtained per voxel. Considering data quality, we show that prioritizing sensor control traffic increases the number of reflectivity samples, N, that can be obtained per voxel given a constant number of voxels, V_c, to scan. Since as N increases, sensing accuracy improves only as a function of $\sqrt{N}$, the gain in accuracy for the reflectivity estimate per voxel is relatively *small* except when prioritizing sensor control increases N significantly (such as when sensor control packets suffer severe delays). Since prioritizing sensor control traffic also reduces the number of control packets dropped, enabling sensors to execute "correct" rather than default controls, data degradation is mitigated. Considering the performance of the tracking application, we show that during times of severe congestion, not prioritizing sensor control traffic can actually lead to tracking errors accumulating over time.

1.3 How to make routing robust to network changes?

Finally, we consider the problem of routing in bandwidth-constrained networks such as wireless sensor and mobile ad-hoc networks, which are additionally characterized by time-varying network topology. In such an environment, the network must accommodate link changes, providing end-end packet delivery while at the same time incurring low control overhead. Yet this is difficult to do in practice: end-end delivery requires some form of end-end (potentially global) coordination, and frequent changes make adaptation to each and every change costly. Link and mobility characteristics may also be difficult to estimate *a priori*, making proactive or predictive routing approaches difficult to implement in practice. We specifically investigate an approach towards MANET routing, which we refer to as "braid routing," that is robust to changes in link characteristics and network topology.[3] Informally, braid routing operates at two timescales. At the longer time-scale, a routing subgraph (i.e., a braid) is constructed that connects a source and destination. At the shorter time-scale, local forwarding decisions are made to select the "best" next hop out of all possible next hops within the braid routing subgraph.

Unlike many existing "backup routing" approaches that pre-compute disjoint paths, e.g., [43], or partially disjoint paths, e.g., [25], a braid does not impose such requirements on the subgraph. Like approaches such as [25], braid routing performs local adaptation in response to link and topology changes. But unlike approaches that route packets over the entire network topology to achieve robustness (e.g., [92]), the braid subgraph over which packets are forwarded is purposefully constrained to limit control overhead (e.g., for braid construction and state maintenance).

We analytically characterize the reliability (the probability that the source and destination nodes have an instantaneous path, see [12]) of a class of braids, their

[3]We note that the term braided routing originates with [25]. The braid routing we propose in this thesis differs from that of [25] in the structure and usage of the braid (i.e., the routing subgraph).

optimality properties, and counter-examples to conjectured optimality properties in a well-structured (grid) network. Through simulation, we compare the reliability of braid, disjoint-path, and full-network routing in both torus and random networks, and show that while braids incur significantly less overhead, they can also achieve reliability close to that of using the full-network. Finally, we compare the performance of braid routing to that of other MANET routing protocols. Considering the percentage of packets delivered, we show that braid routing can deliver more packets than Ad-hoc On-Demand Distance Vector (AODV) [74] routing without significantly increasing overhead. Considering control overhead, and comparing with dynamic source routing [36], we show that braid routing can significantly decrease control overhead while only minimally degrading the number of packets delivered, with gains dependent on node density. In addition to quantifying the gains and overheads of braid routing, our simulations also illustrate how performance results can change rather dramatically depending on the underlying network model.

1.4 Contributions

We summarize here the contributions of this thesis.

- *Where to focus sensing?* Considering a meteorological radar network, we show that the main benefits of optimizing sensing over expected future states of the environment are when there are multiple small phenomena in the environment. We also show that there is a trade-off between the average quality with which a phenomenon is scanned and the number of decision epochs before which a phenomenon is rescanned. Considering only scan quality, we find that a simple lookahead sensing strategy is sufficient. For multiple users, we show that the problem of call admission control in the context of a virtualized private sensor network can be solved in polynomial time when sensor requests are divisible or

fixed in time. When sensor requests are indivisible but may be shifted in time, we show that the call admission control problem is NP-complete.

- *How to make sensing robust to delayed and dropped packets?* We show that prioritizing sensor control traffic over data traffic during times of congestion in a closed-loop sensor network reduces the number of sensor control packets dropped and thereby mitigates data degradation. Considering tracking performance, we show that during times of severe congestion, not prioritizing sensor control traffic can actually lead to tracking errors accumulating over time.

- *How to make routing robust to network changes?* We propose a routing algorithm that selects a type of routing subgraph (a braid) that is robust to changes in the network topology. We analytically characterize the reliability of a class of braids and their optimality properties, and give counter-examples to other conjectured optimality properties in a well-structured (grid) network. Through simulation, we compare the reliability of braid, disjoint-path, and full-network routing in both torus and random networks, and show that while braids incur significantly less overhead, they can also achieve reliability close to that of using the full-network. Considering the percentage of packets delivered, we show that braid routing can deliver more packets than Ad-hoc On-Demand Distance Vector (AODV) [74] routing without significantly increasing overhead. Considering control overhead, and comparing with dynamic source routing [36], we show that braid routing can significantly decrease control overhead while only minimally degrading the number of packets delivered, with gains dependent on node density.

1.5 Outline of Thesis

The rest of this thesis is organized as follows. In Chapter 2, we examine the benefits of sensing strategies that consider expected future states of the environment. In Chapter 3, we consider the problem of how to mediate among conflicting sensing strategies when there are multiple users. In Chapter 4, we consider how to best use network bandwidth in a meteorological radar network, investigating the value of prioritizing sensor control traffic over data traffic during times of congestion. In Chapter 5, we examine the problem of routing in networks with bandwidth-constraints due to changing network conditions. Finally, in Chapter 6, we summarize the contributions of this thesis and discuss future research directions.

CHAPTER 2

MYOPIC VS. LOOKAHEAD SENSING STRATEGIES

2.1 Introduction

In this chapter, we examine the problem of controlling sensors, such as cameras and radars, that cannot simultaneously collect high fidelity data from all locations in the environment. For such sensors, a sensing strategy is necessary to decide where to focus sensing. Any effective sensing strategy for such sensors must balance collecting high fidelity data from some locations (thus implicitly collecting lower fidelity data from other locations), to ensure that the phenomena in those locations are correctly identified, with surveillance scans collecting low fidelity data from all environmental locations, to ensure that all new phenomena are eventually observed. We focus on a meteorological radar network and compare the advantages of a myopic sensing strategy (i.e., optimizing the radar sensing strategy at each decision epoch over only the current state of the environment) with those of a lookahead sensing strategy (i.e, additionally optimizing the radar sensing strategy over expected future states).

Meteorological radars, such as the National Weather Service NEXRAD system, are traditionally tasked to always scan 360°. In contrast, the Collaborative Adaptive Sensing of the Atmosphere (CASA) Engineering Research Center [48] is developing a new generation of small, low-power but agile radars that can perform sector scanning, targeting sensing when and where the user needs are greatest. Since now all meteorological phenomena cannot be observed all of the time with the highest degree of fidelity, the radar controllers must decide how best to sense.

Given the ability of a network of radars to perform sector scanning, how should sensing be adapted over time? Any sensing strategy must consider, for each scan action, both the expected quality with which phenomena would be observed, and the expected time until phenomena would be first observed (for new phenomena) or rescanned, since not all regions are scanned all of the time under sectored scanning. Another consideration is whether to optimize myopically only over current and possibly past environmental state, or whether to additionally optimize over expected future states.

In this work, we examine the sensing benefits of considering expected future states of the environment in a sensing strategy. We specifically compare four sensing strategies for the CASA meteorological radars. The strategies differ in the amount of information they use to select a scan configuration at each decision epoch. The sit-and-spin strategy of always scanning 360° is independent of any external information. The myopic strategy uses the current environmental state but does not estimate future states when making control decisions. The limited lookahead strategies additionally use the expected environmental state k decision epochs in the future in their decisions. Finally, the full lookahead strategy uses all expected future states by casting the problem as a Markov decision process and using reinforcement learning to estimate the optimal sensing strategy. All sensing strategies, excluding sit-and-spin, work by optimizing the "quality" (a term we will define precisely shortly) of the sensed information about phenomena in the environment, while penalizing long re-scan intervals.

We show that the main benefits of considering expected future states in a radar sensing strategy are when there are multiple meteorological phenomena in the environment, and when the maximum radius of any phenomenon is sufficiently smaller than the radius of the radars (see Section 2.3 for radar and phenomenon definitions). We also show that there is a trade-off between the average quality with which a phe-

nomenon is scanned and the number of decision epochs before which a phenomenon is rescanned. Finally, we show that for some environments, a limited lookahead sensing strategy is sufficient. In contrast to other work on radar control (see Section 2.2), we focus on tracking meteorological phenomena and the time frame over which to evaluate control decisions.

The rest of this chapter is organized as follows. In Section 2.2, we review related work on controlling adaptive sensors. In Section 2.3, we describe the meteorological radar control problem. In Section 2.4, we discuss the sensing strategies we consider. In Section 2.5, we describe our meteorological application. In Section 2.6, we overview our simulation results. Finally, in Section 2.7, we summarize our results.

2.2 Related Work

Work by [46] examines the problem of lookahead scheduling of agile radars on airplanes for detecting and tracking ground targets. They show that lookahead sensing strategies for radar tracking of a ground target outperform myopic strategies. In comparison, we consider the problem of tracking meteorological phenomena using ground radars. Thus, our work differs from [46] in the speed and attributes of the objects being tracked (meteorological phenomena versus ground targets such as cars). Unlike [46], our work also considers coordination among multiple different radars, rather than focusing on a single radar on a plane. [46] uses an information-theoretic measure to define the reward metric and then proposes both an approximate solution to solving the MDP Bellman equations as well as a reinforcement learning-based solution to obtain a lookahead policy. We note that [46] uses an off-policy reinforcement learning algorithm Q-learning, while we use an on-policy algorithm Sarsa(λ). Off-policy algorithms update the action-value function using the currently maximal action, while on-policy algorithms use the action that was actually executed; this has implications for function approximation, see [75, 89] for further information.

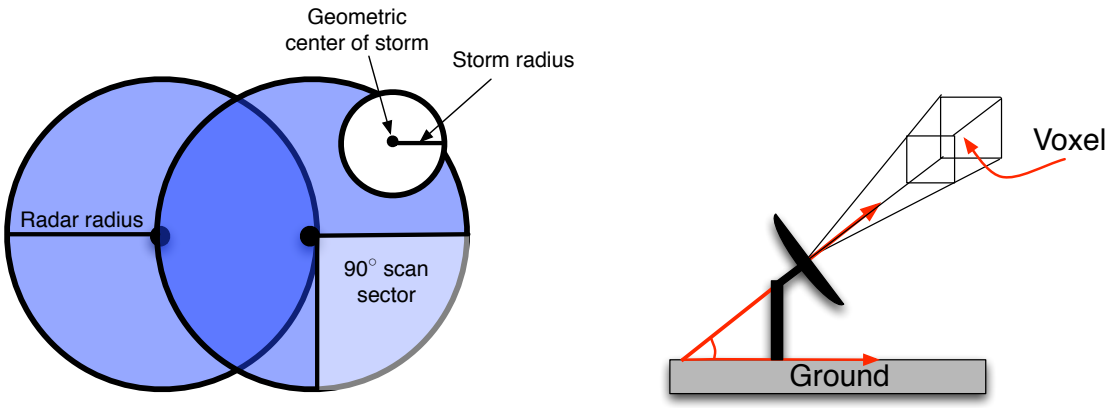
Work by [90] examines the problems of where to target the radar beams and which waveform to use for electronically steered phased array radars. They maintain a set of error covariance matrices and dynamical models for existing targets, as well as track existence probability density functions to model the probability that targets appear. They then choose the scan mode for each target that has both the longest revisit time for scanning a target and error covariance below a threshold. They show that considering the environment two decision epochs ahead outperforms a one-step look-ahead for tracking multiple targets.

Within sensor networks, [57] examines the use of game theory and reinforcement learning to allocate resources in a sensor network. They focus on actions, using reinforcement learning to learn the profit associated with different actions, rather than the profit associated with different state-action pairs. Besides sensor networks, other reinforcement learning applications in large state spaces include robot soccer [85], helicopter control [67] and planetary rovers [99].

2.3 Primer on Meteorological Radars and Sensing

In this section, we give a primer on meteorological radars and the model of meteorological phenomena, specifically storms, that we use.

Figure 2.1(a) illustrates two radars whose footprints are overlapping, and shows an example 90° scan sector in the right radar. The radius of a radar refers to the farthest distance from the radar for which it is still possible to obtain useful measurements, and thus bounds the extent of the radar footprint. A radar operates by sending out pulses at a given rate as it sweeps through the sector it is scanning. From the radar pulses transmitted, reflectivity values are estimated for each voxel (a unit volume of atmosphere, see Figure 2.1). Reflectivity is a measure of the number of scatterers (such as water droplets or insects) in a voxel. For a given time duration, the smaller



(a) (b)

Figure 2.1. Radar and storm cell definitions. (a) Top view of two radars. (b) Side view of radar.

the sector scanned, to some minimum sector size, the better the estimated reflectivity value for a voxel (since the radar can transmit more pulses per voxel, see [20]).

Meteorological algorithms use reflectivity values to identify meteorological phenomena such as storms or tornados. For instance, a storm corresponds to a region of high reflectivity. The meteorological phenomena that we focus on in this work are storms. Figure 2.1(a) shows our storm cell model, comprising a circle with some radius. For specific storm cell parameters, see Section 2.6.1.

2.4 Meteorological Radar Control Problem

As described in the previous section, meteorological radar sensing characteristics are such that the smaller the sector that a radar scans (until a minimum sector size is reached), the higher the quality of the data collected, and thus, the more likely it is that phenomena located within the sector are correctly identified [20]. We define a *radar configuration* to be the start and end angles of the sector to be scanned by an individual radar for a fixed interval of time. We define a *scan action* to be a set of radar configurations (one configuration for each radar in the meteorological

radar network). We define a *sensing strategy* to be an algorithm for choosing scan actions over time. Suppose that we have a network of radars, with fixed locations and possibly overlapping footprints. *The meteorological radar control problem is to determine a sensing strategy for the radars.*

In the rest of this section, we first discuss our performance metrics, allowing us to formally define the meteorological radar control problem. We then describe the quality functions used by the different sensing strategies.

2.4.1 Performance Metrics

We evaluate the performance of different sensing strategies using three metrics: quality, re-scan interval, and cost. *Quality* measures how well a phenomenon is observed; quality depends on the amount of time a radar spends sampling a voxel in space, the degree to which a meteorological phenomena is scanned in its (spatial) entirety, and the number of radars observing a phenomenon; higher quality scans are better. We define quality formally in Section 2.4.2. *Re-scan interval* is the number of decision epochs before a phenomenon is either first observed or rescanned; we would like this value to be below some threshold. *Cost* is a metric that combines quality with the re-scan interval, and that additionally considers whether a phenomenon was never scanned. We define cost formally in Section 2.5.4. *The meteorological radar control problem is formally the problem of how to dynamically select scan actions over time to maximize quality while minimizing the re-scan interval.*

2.4.2 Quality Function for Scanning a Phenomenon

The quality function for scanning a phenomenon under a given scan action was proposed by radar meteorologists in [48, 71, 72]. Specifically, a scan action S_r specifies a radar configuration s_r for each radar r in the radar network under consideration. Hence S_r is a set of radar configurations where $|S_r|$ corresponds to the number of

radars in network. The quality $U_p(p, S_r)$ of scanning a phenomenon p using scan action S_r can be computed as follows,

$$\begin{aligned} U_p(p, s_r) &= F_c\left(c(p, s_r)\right) \times \left[\beta F_d\left(d(r, p)\right) + (1-\beta) F_w\left(\frac{w(s_r)}{360}\right)\right] \\ U_p(p, S_r) &= \max_{s_r \in S_r}\left[U_p(p, s_r)\right] \end{aligned} \tag{2.1}$$

where

$$\begin{aligned} w(s_r) &= \text{size of sector } s_r \text{ scanned by } r \\ a(r, p) &= \text{minimal angle that would allow } r \text{ to cover } p \\ c(p, s_r) &= \frac{w(s_r)}{a(r, p)} = \text{coverage of } p \text{ by } r \text{ scanning } s_r \\ h(r, p) &= \text{distance from } r \text{ to geometric center of } p \\ h_{max}(r) &= \text{range of radar } r \\ d(r, p) &= \frac{h(r, p)}{h_{max}(r)} = \text{normalized distance from } r \text{ to } p \\ \beta &= \text{tunable parameter} \end{aligned}$$

$U_p(p, S_r)$ is the maximum quality obtained for scanning phenomenon p over all possible radars and their associated radar configurations s_r. $U_p(p, s_r)$ is the quality obtained for scanning phenomenon p using a specific radar r and radar configuration s_r. The functions $F_c(\cdot)$, $F_w(\cdot)$, and $F_d(\cdot)$ from [48, 71, 72] are plotted in Figure 2.2. F_c captures the effect on quality due to the percentage of the phenomenon covered; to usefully scan a phenomenon, at least 95% of the phenomenon must be scanned. F_w captures the effect of radar rotation speed on quality; as rotation speed is reduced, quality increases. F_d captures the effects of the distance from the radar to the geometrical center of the phenomenon on quality; the further away the radar center is from the phenomenon being scanned, the more degraded will be the scan quality due

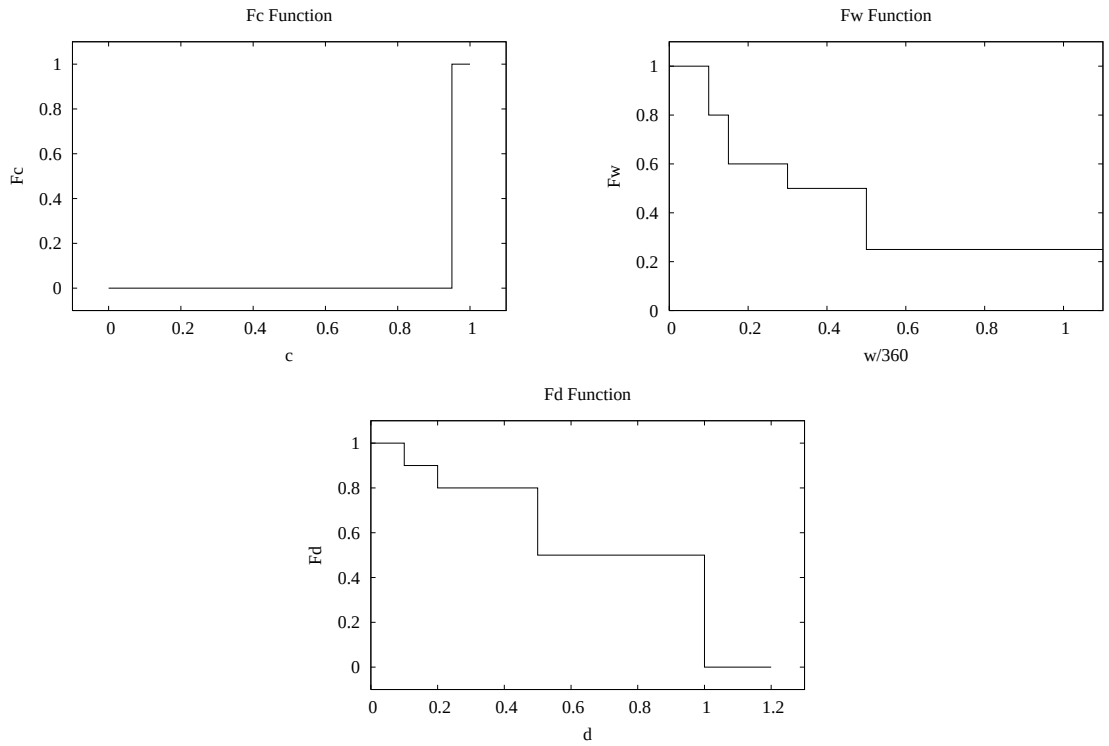


Figure 2.2. The $F_c(\cdot)$, $F_w(\cdot)$, and $F_d(\cdot)$ step functions from [48, 71, 72] used by the U_p and U_s quality functions, from [48, 71, 72].

to attenuation. Due to the F_w function, the quality function $U_p(p, s_r)$ outputs the same quality for scan angles of 181° to 360°.

Whenever a phenomenon p is scanned with scan action s_r, we can compute the quality for the scanned phenomenon, $u_p = U_p(p, s_r)$. If the phenomenon is not rescanned at the next decision epoch, however, we decay the quality u_p over time. Specifically, we assume that for those previously observed phenomena that the radars do not scan at decision epoch t, that the associated qualities are decayed by a fixed amount κ_p,

$$\begin{aligned} \text{If } u_p \geq \kappa_p \qquad & u_p = u_p - \kappa_p \\ \text{else} \qquad & u_p = 0 \end{aligned} \tag{2.2}$$

Our goal is to ensure that all phenomena continue to have high quality values associated with them, even after the phenomena are initially observed.

2.4.3 Quality Function for Scanning a Sector

While the previous section defines the quality function for scanning a phenomenon under a given scan action, this section defines the quality function for scanning a sector, again as defined in [48, 71, 72]. Defining a function that indicates the quality with which a sector has been scanned gives a way to evaluate how well each area of the environment has been observed, regardless of whether there are currently any phenomena in the area. More specifically, the quality $U_s(r_i, s_r)$ for scanning a subsector i of radar r using configuration s_r is,

$$U_s(r_i, s_r) = F_w\left(\frac{w(s_r)}{360}\right) \tag{2.3}$$

where the functions $w(s_r)$ and $F_w(\cdot)$ are as defined previously.

As with the phenomenon quality, we decay the quality $u_s = U_s(r_i, s_r)$ for scanning a subsector i of radar r when that subsector is not immediately rescanned at the next decision epoch. Now we assume that for those sectors that the radars do not scan at decision epoch t, that the associated qualities are decayed by a fixed amount κ_s,

$$\begin{aligned} \text{If } u_s \geq \kappa_s \qquad & u_s = u_s - \kappa_s \\ \text{else} \qquad & u_s = 0 \end{aligned} \tag{2.4}$$

As with the phenomenon quality, we would like to ensure that all scan sectors continue to have high quality values associated with them.

2.5 Sensing Strategies

We have now defined the quality functions U_p and U_s. The quality function that is actually optimized, however, depends on the individual sensing strategy. In this section, we describe the different sensing strategies that we consider.

Generally, any effective radar sensing strategy must balance scanning small sectors (thus implicitly *not* scanning other sectors), to ensure that phenomena are correctly identified, with scanning a variety of sectors, to ensure that no phenomena are missed. Intuitively, a sensing strategy that scans sectors, rather than always 360°, is only preferable when the quality gained for scanning a sector is greater than the quality lost for not scanning another sector.

We specifically compare the performance of four radar sensing strategies: (i) sit-and-spin, (ii) myopic, (iii) limited look-ahead, and (iv) full look-aheaad. The strategies differ in whether they consider only current or also expected future states of the environment when selecting scan actions. For example, suppose a storm cell is about to move into a high-quality multi-doppler region (i.e., the area where multiple radar footprints overlap). By considering expected future states, a lookahead strategy can anticipate this event and have all radars focused on the storm cell when it enters the multi-doppler region, rather than expending resources (with little "reward") to scan the cell before it enters this region. We now describe each of the sensing strategies in more detail.

2.5.1 Sit-and-spin Sensing Strategy

In the sit-and-spin strategy, all radars always scan 360°. This is our baseline sensing strategy and corresponds to how meteorological radars, such as those in the National Weather Service NEXRAD system, are traditionally tasked to scan.

2.5.2 Myopic Sensing Strategy

In the myopic sensing strategy, only the current state of environment is considered. We compute the myopic quality, represented by $U_m(S_r|T_r)$, for different sets of radar configurations S_r with the following equation based on the U_p quality function defined in Section 2.4.

$$U_m(S_r|T_r) \quad = \quad \sum_p U_p(p, S_r|T_r) \tag{2.5}$$

The optimal set of radar configurations is given by $S_r^* = \text{argmax}_{S_r} U_m(S_r|T_r)$. To account for the decay of quality for unscanned sectors and phenomena, and to consider the possibility of new phenomena appearing, we restrict S_r to be those scan actions that ensure that every sector has been scanned at least once in the last T_r decision epochs. T_r is a tunable parameter whose purpose is to satisfy the meteorologists' request, as specified in [73], that all sectors be scanned, for instance by a 360° scan, at most every 5 minutes.

2.5.3 Limited Lookahead Sensing Strategy

In the limited look-ahead strategy we consider a limited number of expected future states of the environment when deciding how to scan. We examine both a 1-step and a 2-step look-ahead sensing strategy. Although we do not have an exact model of the dynamics of different phenomena, to perform the look-ahead we estimate the future attributes of each phenomenon using a separate Kalman filter. For each filter, the true state $\mathbf{x}$ is a vector comprising the (x, y) location and velocity of the phenomenon, and the measurement $\mathbf{y}$ is a vector comprising only the (x, y) location. The Kalman filter assumes that the state at time t is a linear function of the state at time $t-1$ plus some Gaussian noise, and that the measurement at time t is a linear function of the state at time t plus some Gaussian noise. In particular, $\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + N[\mathbf{0}, \mathbf{Q}]$ and $\mathbf{y}_t = \mathbf{B}\mathbf{x}_t + N[\mathbf{0}, \mathbf{R}]$.

Following work by [60], we initialize each Kalman filter as follows. The $\mathbf{A}$ matrix reflects that storm cells typically move to the north-east. The $\mathbf{B}$ matrix, which when multiplied with $\mathbf{x}_t$ returns $\mathbf{y}_t$, assumes that the observed state $\mathbf{y}_t$ is directly the true state $\mathbf{x}_t$ plus some Gaussian noise. The $\mathbf{Q}$ matrix assumes that there is little noise in the true state dynamics. Finally, the measurement error covariance matrix $\mathbf{R}$ is a function of the quality U_p with which phenomenon p was scanned at time t. We discuss how to compute the σ_t's in Section 2.6. We use the first location measurement of a storm cell $\mathbf{y}_0$, augmented with the observed velocity, as the the initial state $\mathbf{x}_0$. We assume that our estimate of $\mathbf{x}_0$ has little noise and use $.0001 * I$ for the initial covariance $\mathbf{P}_0$. The Kalman filter parameters are thus given by,[1]

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} .0001 & 0 & 0 & 0 \\ 0 & .0001 & 0 & 0 \\ 0 & 0 & .0001 & 0 \\ 0 & 0 & 0 & .0001 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} \sigma_t & 0 \\ 0 & \sigma_t \end{bmatrix}$$

We then compute the k-step look-ahead quality for different sets of radar configurations S_r with,

$$U_K(S_{r,1}|T_r) \quad = \quad \sum_{k=1}^{K} \phi^{k-1} \sum_{i=1}^{N_p} U_p(p_{i,k}, S_{r,k}|T_r)$$

where N_p is the number of phenomena in the environment in the current decision epoch, $p_{i,0}$ is the current set of observed attributes for phenomenon i, $p_{i,k}$ is the k-step set of predicted attributes for phenomenon i, $S_{r,k}$ is the set of radar configurations for the kth decision epoch in the future, and ϕ is a tunable discount factor between

[1]We note that with these parameters the Kalman filter model can be further simplified, see Section 4.4.3.

0 and 1. The optimal set of radar configurations is $S^*_{r,1} = \text{argmax}_{S_{r,1}} U_K(S_{r,1}|T_r)$. As with the myopic sensing strategy, to account for the decay of quality for unscanned sectors and phenomena, and to consider the possibility of new phenomena appearing, we restrict S_r to be those scan actions that ensure that every sector has been scanned at least once in the last T_r decision epochs.

2.5.4 Full Lookahead Sensing Strategy

Finally, we consider a full look-ahead sensing strategy, optimizing over all expected future states of the environment. To obtain the full look-ahead sensing strategy, we formulate the meteorological radar control problem as a Markov decision process (MDP) and use reinforcement learning as a solution technique. While a POMDP (partially observable MDP) could be used to model the environmental uncertainty, due to the cost of solving a POMDP with a large state space [66], we choose to formulate the meteorological radar control problem as an MDP with quality (or uncertainty) variables as in an augmented MDP [49].

2.5.4.1 MDP Formulation

Our MDP formulation of the meteorological radar control problem is as follows.

- **State space.** We define S to be the observed state of the environment. The observed state comprises the observed number of storm cells, the observed x, y velocities of each storm cell, and the observed dimensions of each storm cell given by x, y center of mass and radius. To model the uncertainty in the environment, we additionally define as part of the state quality variables u_p and u_s based on the U_p and U_s quality functions defined in Equations 2.1 and 2.3. u_p is the current quality $U_p(\cdot)$ of each observed storm cell, and u_s is the current quality $U_s(\cdot)$ of each 90° subsector, starting at 0°, 90°, 180°, or 270°.

- **Action space.** We define A to be the set of actions available to the radars. This is the set of radar configurations for a given decision epoch. We restrict each radar to scanning subsectors that are a multiple of 90°, starting at 0°, 90°, 180°, or 270°. Thus for each radar there are 13 possible actions, and so with N radars there are 13^N possible actions at each decision epoch.

- **Transition function.** The transition function $T(S \times A \times S) \rightarrow [0, 1]$ encodes the *observed* environment dynamics: specifically the appearance, disappearance, and movement of storm cells and their associated attributes. For meteorological radar control, the next state really is a function of not just the current state but also the action executed in the current state. For instance, if a radar scans 180° rather than 360°, then any new storm cells that appear in the un-scanned areas will not be observed. Thus, the new storm cells that will be observed depend on the scanning action of the radar.

- **Cost function.** The cost function $C(S, A, S) \rightarrow \mathcal{R}$ encodes the goals of the radar sensing network. C is a function of the error between the true state and the observed state, whether all storms have been observed, and a penalty term for not rescanning a storm within T_r decision epochs. More precisely,

$$C = \sum_{i=1}^{N_p^o} \sum_{j=1}^{N_d} |d_{ij}^o - d_{ij}| + (N_p - N_p^o) P_m + \sum_{i=1}^{N_p} I(t_i) P_r \tag{2.6}$$

where N_p^o is the observed number of storms, N_d is the number of attributes per storm, d_{ij}^o is the observed value of attribute j of storm i, d_{ij} is the true value of attribute j of storm i, N_p is the true number of storms, P_m is the penalty for missing a storm, t_i is the number of decision epochs since storm i was last scanned, P_r is the penalty for not scanning a storm at least once within T_r decision epochs, and $I(t_i)$ is an indicator function that equals 1 when $t_i \geq T_r$.

The quality with which a storm is observed determines the difference between the observed and true values of its attributes.

2.5.4.2 Learning Algorithm

We use the linear Sarsa(λ) reinforcement learning algorithm [89] as a solution technique to solve the MDP we defined for the meteorological radar control problem. Dynamic programming could also be used to solve the MDP, since we have access to a transition function and a cost function. We choose to use reinforcement learning instead of dynamic programming, however, since our transition and cost functions consider only a small part of the possible meteorological states (e.g., only storm cells, and focusing only on certain storm characteristics), and since we would ultimately like to learn over real traces of radar data, with phenomena features extracted from the data, rather than using a model.

In Table 2.1 we show the Sarsa(λ) algorithm. The algorithm estimates the action-value function $Q_{s,a}$, representing the expected value of taking action a in state s, by keeping track of the actual sequence of costs received as actions are taken. The heart of the algorithm is lines 20-22 in Table 2.1. In particular, based on the cost received for taking action a in state s, an error δ is computed and the weights are updated. The intuition here is that there is an old estimate for the value of taking action a in state s, represented by the action-value $Q_{s,a}$. There is also a new estimate given by the immediate cost just received for taking action a in state s plus the expected value of taking action a' in state s' (where s' is the next state to which we transition). The error δ is then the difference between these two estimates.

Due to the continuous-valued state variables, such as a storm's location, we use function approximation and approximate $Q_{s,a}$ as a linear combination of basis functions. To obtain the basis functions, we use tile coding [33, 87, 88]. Tile coding works by partitioning the state space into a set of tiles. For example, suppose our

Initialization:
    $F \leftarrow$ set of all features
    $A \leftarrow$ set of all actions
    $w_{f,b} = 0,\ e_{f,b} = 0, \forall f \in F, \forall b \in A$
    $s =$ initial state
    $a =$ initial action *(E.g., scan 360°)*

Repeat until error δ is sufficiently small
    Update eligibility traces:
        $F_s \leftarrow$ set of on features for state s
        $e_{f,b} \leftarrow \lambda e_{f,b}, \forall f \in F, \forall b \in A$
        $e_{f,a} \leftarrow e_{f,a} + 1, \forall f \in F_s$
    Environment step:
        Take action a, observe cost c and next state s'
    Choose next action:
        $F_{s'} \leftarrow$ set of on features for state s'
        $Q_{s',b} \leftarrow \sum_{f \in F_{s'}} w_{f,b}, \forall b \in A$
        With probability $1 - \epsilon$: $a' \leftarrow \arg\min_b Q_{s',b}$
        With probability ϵ: $a' \leftarrow$ random action
    Learn:
        $\delta = c - Q_{s,a} + \gamma Q_{s',a'}$
        $w_{f,b} = w_{f,b} + \alpha \delta e_{f,b}, \forall f \in F, \forall b \in A$
    Update current state and action:
        $a = a',\ s = s'$

Table 2.1. Linear Sarsa(λ) reinforcement learning algorithm. Adapted from [85, 88, 89]. $Q_{s,a}$ is the action-value for state s and action a. $w_{f,a}$ is the set of weights used to linearly combine the basis functions obtained for features f and action a. α is the learning rate: it represents the rate at which the weights are updated. γ is the discount factor: it represents how much importance is placed on the future versus the present. Eligibility traces are used to incorporate state history when later updating the action-values: the more recently and frequently a state was visited, the higher will be the value of its eligibility trace.

state space consists only of one state variable, the x-location of the storm cell. Then in the simplest case, we would choose some number of bins into which to partition the values that x-location can take on. This would result in one single-dimensional tiling. If we had multiple variables, we could also tile the cross-product of variables to get multi-dimensional tilings. Tilings allow us to extract features from the state as follows. A given assignment of values to the state variables (i.e., a given state) maps to a unique "on" tile in each tiling. This gives a binary vector for each tiling, with a 1 for the feature (tile) that is on and 0's for the remaining features. In this way

we obtain a basis function from each tiling. Because of this function approximation, however, the error δ is not used to directly update the action-value function, but is instead used to update the weights. By appropriately adjusting the weights, the reinforcement learning algorithm learns how best to linearly combine the basis functions and thus approximate the action-value function $Q_{s,a}$.

Given the action-value function $Q_{s,a}$, then the best action to execute in a state is the action with the lowest action-value for the state. If the cost received for taking an action in a state is a function of the difference between the true state of the environment and the observed state, then implicitly the value for taking an action in a state represents the action that will best let the true state be observed. Unlike the k-step look-ahead Kalman filter algorithms, however, the linear Sarsa(λ) algorithm does not directly predict the next state of the environment.

2.6 Meteorological Application

In this section, we describe the meteorological application that we use to evaluate the different sensing strategies. We consider up to four overlapping radars, arranged as in Figure 2.3, with 10 and 30km radii as in [48, 93]. Following [48], we use a 30-second decision epoch. In the rest of this section, we first describe the storm cell model we use to model meteorological phenomena. We then describe the radar model we use to determine how well a scan action is able to observe a storm cell.

2.6.1 Storm Cell Model

Due to a limited amount of real storm track data, we use the following storm cell model to generate traces of storm cell movement through the environment. In our storm cell model, we assume that a new storm cell can appear anywhere within the radar footprints and that a maximum number of cells can be present on any decision epoch. When the (x, y) center of a storm cell is no longer within range of any radar,

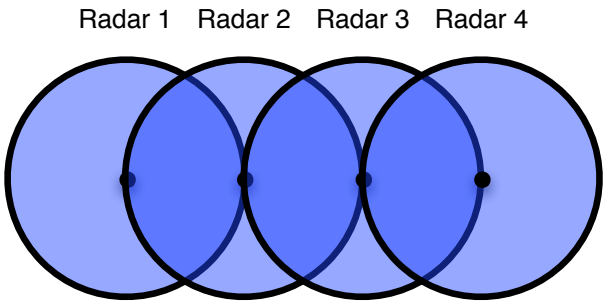


Figure 2.3. Radar setup for simulations.

the cell is removed from the environment. We derive the maximum storm cell radius from [79], which uses 2.83km as "the radius from the cell center within which the intensity is greater than e^{-1} of the cell center intensity." A storm cell's radius can then range from 1 to 4 km.

To determine the distribution of storm cell velocities, we use 39 real storm cell tracks obtained from the National Severe Storms Laboratory courtesy of Kurt Hondl and the WDSS-II software [31]. Each track is a series of $(latitude, longitude)$ coordinates. We first compute the differences in latitude and longitude, and in time, between successive pairs of points. This gives us data on the latitude and longitude velocities. We then fit the latitude velocity data with a Gaussian distribution, and fit the longitude velocity data with another Gaussian distribution. Given that the length of a latitude degree at 40° latitude equals 111.04 km and the length of a longitude degree at 40° latitude equals 85.39 km, we obtain that the latitude (or x) velocity has mean 9.1 km/hr and standard deviation of 35.6 km/hr and that the longitude (or y) velocity has mean 16.7 km/hr and standard deviation of 28.8 km/hr. To obtain a storm cell's (x, y) velocity, we sample the Gaussian distributions.

To simulate the environment transitions we use a stochastic model of rainfall in which storm cell arrivals are modeled using a spatio-temporal Poisson process, see [79, 15]. To determine the number of new storm cells to add during a decision epoch, we sample a Poisson random variable with rate $\lambda\eta\delta a\delta t$ with $\lambda = 0.075$ storm

cells/km^2 and $\eta = 0.006$ storm cells/minute from [79]. From the radar setup we have $\delta a = r(N+1)2r$, where N is the number of radars and r is the radar radius. From the 30-second decision epoch we have $\delta t = 0.5$ minutes. New storm cells are uniformly randomly distributed in the $r(N+1)$ km $\times$ $2r$ km region and we uniformly randomly choose new storm cell attributes from their range of values. This simulates the true state of the environment over time.

2.6.2 Radar Model

The following simplified radar model determines how well the radars observe the true environmental state under a given set of radar configurations. If a storm cell p is scanned using a set of radar configurations S_r, the location, velocity, and radius attributes are observed as a function of the $U_p(p, S_r)$ quality defined in Section 2.4.2. $U_p(p, S_r)$ returns a value u between zero and one. Then the observed value of the attribute is the true value of the attribute plus some Gaussian noise distributed with mean zero and standard deviation $(1-u)V^{max}/\rho$. V^{max} is either the average value of the attribute (in the case of the storm velocity attributes) or the maximum value of the attribute (in the case of the storm location attributes); ρ is a scaling factor that allows us to adjust the noise variability. As $1/\rho$ increases, the amount of Gaussian noise added to the true state to obtain the observations also increases. For example, when $1/\rho = 0.1$, the standard deviation of the Gaussian noise can be at most 10% of the average storm velocity. Since u depends on the decision epoch t, for the k-step look-ahead sensing strategy we also use $\sigma_t = (1-u_t)V^{max}/\rho$ to compute the measurement error covariance matrix, R, in our Kalman filter.

We distinguish the true environmental state known only to the simulator from the observed environmental state used by the sensing strategies for several reasons. Although radars provide measurements about meteorological phenomena, the true attributes of the phenomena are unknown. Poor overlap in a dual-Doppler area, scan-

Variable	Meaning	Value
β	Weighting term in quality function	.5
κ_p	Decay rate of phenomenon quality	.25
κ_s	Decay rate of sector quality	.25
α	Learning rate for Sarsa(λ)	.0005
ϵ	Exploration rate for Sarsa(λ)	.01
γ	Discount factor for Sarsa(λ)	.9
λ	Eligibility decay for Sarsa(λ)	.3
ϕ	Discount factor for k-step strategy	.75
T_r	# of decision epochs within which storm should be re-scanned	5
P_r	Penalty for not re-scanning storm within T_r decision epochs	200
P_m	Penalty for not observing a storm	15.5667

Table 2.2. Parameter settings for variables. Additionally, for Sarsa(λ) we use a single tiling for each state variable: we use a granularity of 1.0 for the (x, y) location and radius tilings, while we use a granularity of 0.1 for the (x, y) velocity, phenomenon confidence, and radar sector confidence tilings. To obtain the penalty $P_m = 15.5667$ for each unobserved storm cell we assume that any unobserved storm cell is observed with quality 0, hence $u = 0$, and then sum over $(1-u)V^{max}/\rho$ for all attributes. Using $P_r = 200$ ensures that if a storm cell has not been rescanned within the appropriate amount of time, this part of the cost function will dominate.

ning a subsector too quickly or slowly, or being unable to obtain a sufficient number of elevation scans will degrade the quality of the measurements. Consequently, models of previously existing phenomena may contain estimation errors such as incorrect velocity, propagating error into the future predicted locations of the phenomena. Additionally, when a radar scans a subsector, it obtains more accurate estimates of the phenomena in that subsector than if it had scanned a full 360°, but less accurate estimates of the phenomena outside the subsector.

2.7 Simulation Results

In this section, we compare the performance of the different sensing strategies from Section 2.5 using the meteorological application described in the previous section. The simulation parameters are summarized in Table 2.2. All sensing strategies are always compared over the same true environmental state. We first examine the performance of the limited lookahead sensing strategies. We then examine the performance of

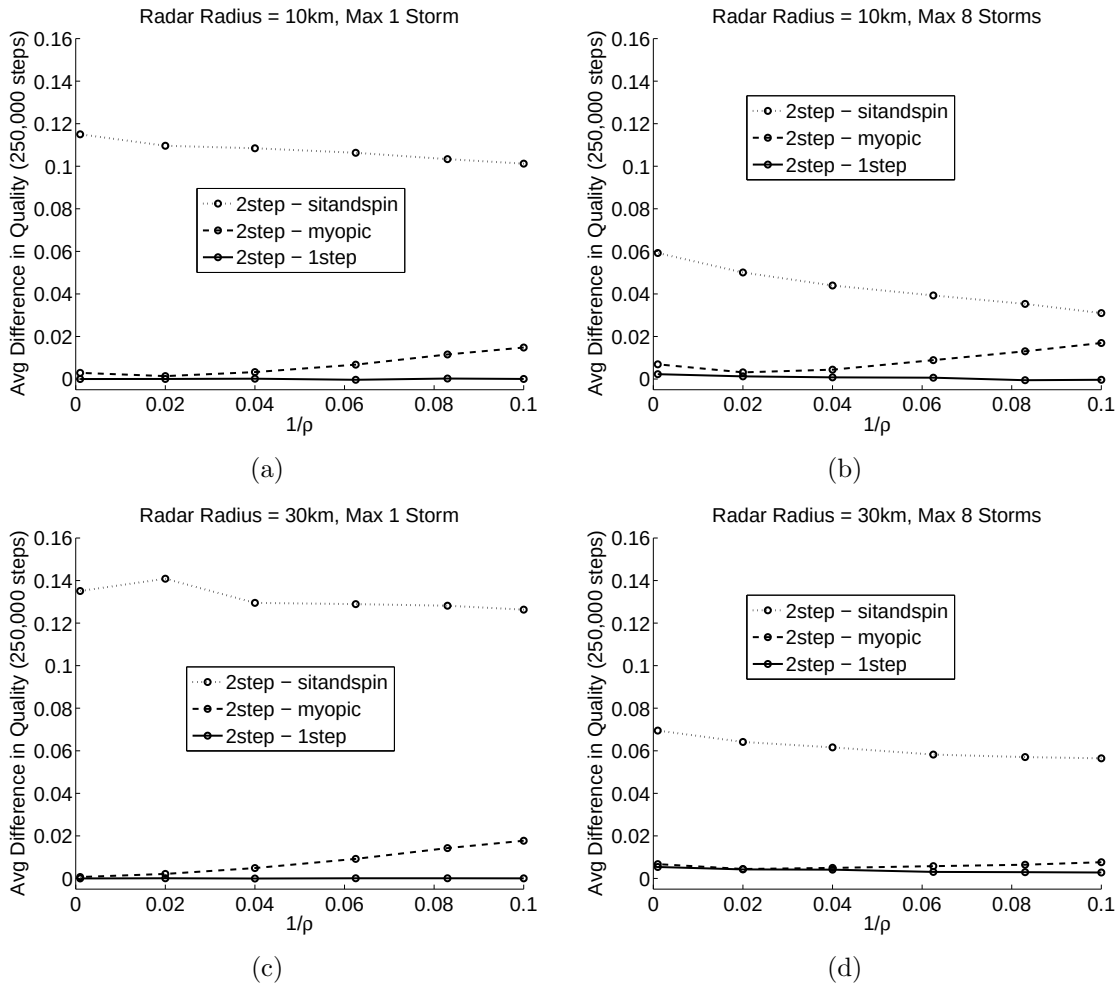


Figure 2.4. Comparing the sensing strategies based on quality two radars (two radar scenario).

the full lookahead strategy. Finally, we examine the scaling behaviour of the sensing strategies.

2.7.1 Limited Lookahead Performance

Scan Quality. Figure 2.4 shows the average difference in per-storm quality between the 2-step lookahead strategy and the sit-and-spin, myopic, and 1-step lookahead strategies. For both 10 km and 30 km radii, the sit-and-spin strategy has the lowest scan quality relative to the 2-step, the myopic strategy has the next lowest relative quality, and the 1-step strategy has the highest (or the same) relative quality. As

the maximum number of storms in the environment increases from one to eight, the scan quality also increases for the sit-and-spin sensing strategy: since there are now more possible storms, it is more likely that a storm cell is close to a radar. Thus, the F_d term of the quality function, see Section 2.4.2, is more likely to be large. Finally, notice that decreasing the radar radius decreases the differences in quality of the different sensing strategies, although the overall trends remain the same. For instance, with a 10 km radius, in Figures 2.4 (a) and (d), the 1-step quality is essentially the same as the 2-step quality. We hypothesize that this is a consequence of the large maximum storm cell radius, 4 km, relative to the 10 km radar radius: larger scan sectors will be needed to fully cover any storm, thereby decreasing the scan quality. This indicates that there may be some maximum number of storms above which it is best to sit-and-spin. In summary, Figure 2.4 indicates that the 2-step lookahead sensing strategy can slightly outperform the 1-step lookahead sensing strategy in how well it scans storms, and more significantly outperforms the sit-and-spin and myopic sensing strategies. The performance gains depend in part on the number of storms present in the environment, and on the size of maximum storm cell radius relative to the radar radius.

Re-scan interval. Next, Figure 2.5 shows the cumulative distribution functions (CDFs) of the number of decision epochs before a storm cell is first observed or re-scanned for the sit-and-spin, 1-step lookahead, and 2-step lookahead sensing strategies. Figures 2.5(a) and (b), show that regardless of the size of the radar radius, if there is at most one storm in the environment, and $1/\rho = 0.001$ (i.e., little measurement noise), the 1-step lookahead and 2-step lookahead sensing strategies re-scan storms with approximately the same frequencies. Figures 2.5(a) and (b), also show that the size of the re-scan interval is typically slightly smaller for a 30 km radius than for a 10 km radius. We hypothesize that this is a consequence of the 4 km storm cell radius: since there is a maximum of one storm cell in the environment, a sector

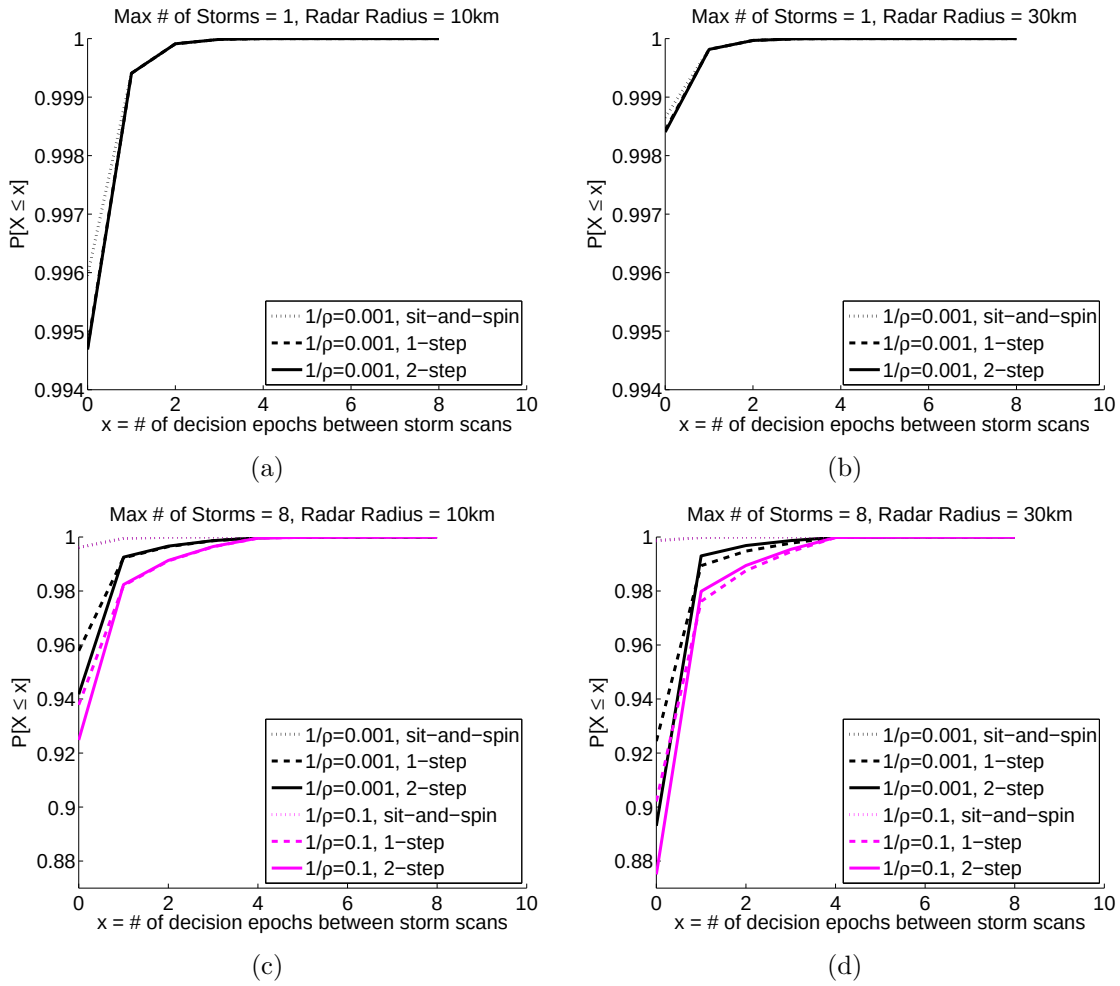
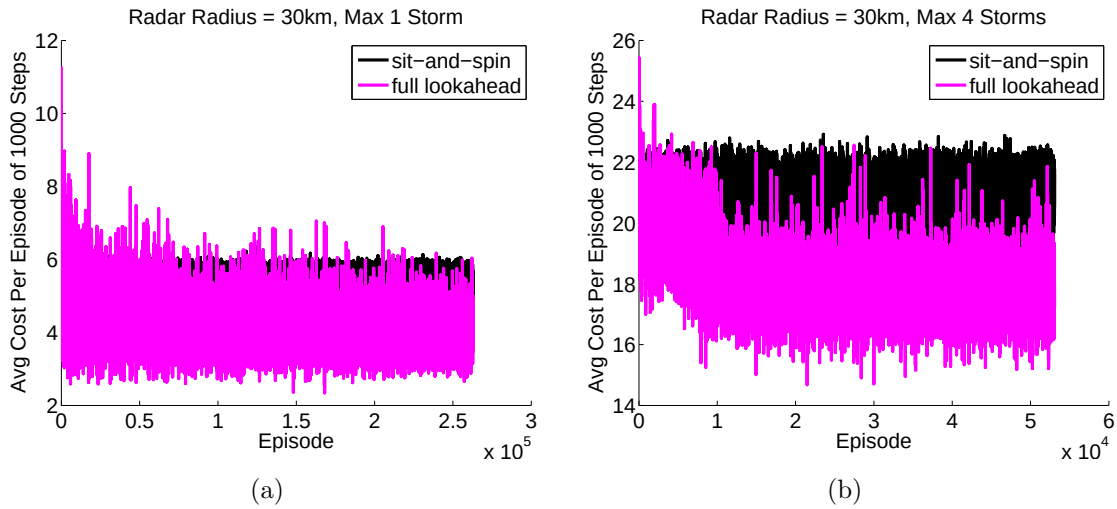


Figure 2.5. Comparing the sensing strategies based on re-scan interval (two radar scenario).

scan of a 30 km radius is more likely to cover at least 95% of the storm. Thus with a 30 km radius, a sensing strategy is less likely to attempt to scan a storm and fail (i.e., scan less than 95% of a storm).

Figures 2.5(c) and (d) then show that when there are at most eight storm cells in the environment, that the 1-step lookahead sensing strategy re-scans more storms within zero decision epochs (i.e., immediately) than does the 2-step lookahead strategy. This is shown by the higher values taken on by the 1-step CDF for $x = 0$. Figures 2.5(c) and (d) also show that the size of the re-scan interval is typically smaller for a 10 km radius than for a 30 km radius, unlike the situation when there



(a) (b)

Figure 2.6. Example convergences of the full lookahead sensing strategy, Sarsa(λ).

is at most one storm cell in the environment. We hypothesize that this is again a consequence of the 4 km storm cell radius: the size of the sector scans will be larger with a 10 km rather than a 30 km radius, consequently, multiple storms will more likely be covered, thereby decreasing the re-scan time. Note that for the sit-and-spin CDF, $P[X \leq 1]$ is not 1; due to noise, for example, the measured location of a storm cell may be (expected) outside of any radar footprint and consequently the storm cell will not be observed.

2.7.2 Full Lookahead Performance

Figure 2.6 shows example convergence profiles for the Sarsa(λ) reinforcement learning algorithm used to learn a full lookahead sensing strategy. When there are a maximum of four storms, we restrict the full lookahead strategy to scanning only 180° or 360° sectors to reduce the time needed for convergence.

Scan quality. Figure 2.7(a) again examines scan quality, now showing the average difference in per-storm scan quality between the full lookahead sensing strategy and the sit-and-spin and 2-step lookahead strategies. When $1/\rho = 0.001$, the full lookahead strategy has the same or higher relative quality than does sit-and-spin, but

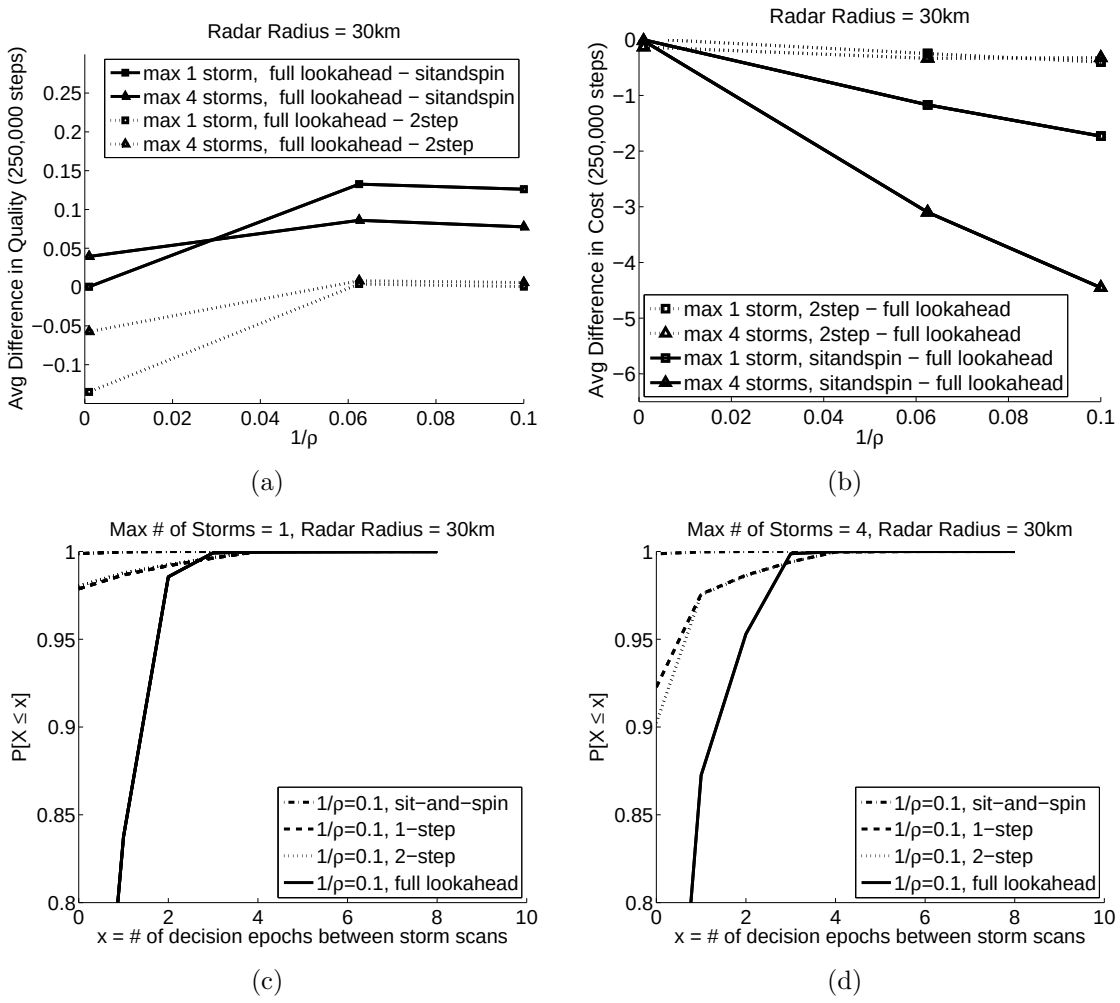


Figure 2.7. Comparing the sensing strategies based on (a) scan quality, (b) cost, and (c), (d) re-scan intervals (two radar scenario).

significantly lower relative quality (0.05 to 0.15) than does the 2-step. This reflects in part the difficulty of learning to perform as well as or better than Kalman filtering. Additionally, the full lookahead strategy is learning to minimize cost, not maximize quality.[2] Examining the learned strategy when there is at most one storm and little

[2]Recall that cost is a function of quality, the re-scan interval, and a penalty for not observing or re-scanning a storm within the re-scan interval. While the full lookahead strategy minimizes cost, the limited lookahead strategy instead maximizes quality but is constrained to re-scan all scan sectors within the re-scan interval. Unlike for the limited lookahead strategy, however, the cost penalty is needed for the full lookahead strategy so that it learns how frequently storms should be rescanned. As shown in Figure 2.7(c) and (d) the full lookahead strategy actually learns to re-scan storms more frequently than is dictated by the re-scan interval. One way to directly compare the

observation noise (i.e., $1/\rho = 0.001$), the full lookahead strategy learns to simply sit-and-spin, since sector scanning confers little benefit. As the observation noise increases, the relative difference increases for the sit-and-spin strategy, and decreases for the 2-step lookahead strategy.

Cost. Next, Figure 2.7(b) shows the average difference in cost (defined as in Equation 2.6) between the full lookahead sensing strategy and the sit-and-spin and 2-step lookahead strategies for a 30 km radar radius. The full lookahead sensing strategy has the lowest average cost.

Re-scan interval. Finally, Figures 2.7(c) and (d) examine the the size of the re-scan intervals for the full lookahead sensing strategy. Figures 2.7(c) and (d) show that, as a consequence of the penalty for not scanning a storm within $T_r = 5$ decision epochs, while the full lookahead sensing strategy may rescan fewer storm cells within one, two, or three decision epochs than do the other sensing strategies, it also scans almost all storm cells within four decision epochs.

Overall, depending on the environment in which the radars are deployed, there are decreasing marginal returns for considering more than one or two future expected states. The 2-step and full-lookahead sensing strategies perform similarly in part because storms move relatively slowly. More generally, the performance gains from considering expected future states of the environment depend on the speed and predictability of storm movement over a given time scale, the number of storms in the environment, and the maximum radius of a storm relative to the radar radius. Instead, the primary value of the full lookahead reinforcement learning strategy here for the radar control problem is balancing multiple conflicting goals, i.e., maximizing scan quality while minimizing the size of the re-scan interval. Additionally, implementing the full lookahead sensing strategy using reinforcement learning in a real

---

limited and full lookahead strategies in terms of cost would be to integrate learning into the limited lookahead strategy by considering a limited lookahead version of the full lookahead strategy.

meteorological radar network requires addressing the differences between the offline environment in which the learned strategy is trained, and the online environment in which the strategy is deployed. Given the slow convergence time for Sarsa(λ) (on the order of days), training solely online is likely infeasible, although the time complexity could be mitigated by using hierarchical reinforcement learning methods and semi-Markov decision process. Some online training could be achieved by treating 360° scans as the true environment state. Then when unknown states are entered, learning could be performed, alternating between 360° scans to gauge the true state of the environment and exploratory scans by the reinforcement learning algorithm.

2.7.3 Scaling Behaviour

Figure 2.8 examines the scaling behaviour of the sensing strategies. For these results we restrict the radars to scanning 180° sectors and 360° due to the cost of computing the limited lookahead strategies. Figures 2.8 (a) and (b) show the average per-storm quality as the number of radars in the network increases. For a maximum of one storm in the environment, Figure 2.8(a) shows that there are gains in quality going from one radar to two radars in the environment for all sensing strategies and then the quality levels off with an increasing number of radars. This is because now it is more likely that a storm will be close to a radar, with at least two radars, and so the F_d term of the quality function, see Section 2.4.2, is more likely to be large. Figure 2.8(a) also shows that the 1-step and 2-step lookahead strategies achieve the same average quality when there is at most one storm in the environment.

For a maximum of eight storms in the environment, Figure 2.8(b) shows that the quality increases as the number of radars increases, although the size of the gains is decreasing. This is because the maximum number of storms is fixed, but the number of radars in the network is increasing. When there are many storms but few radars, scan sectors must typically be larger (since there are now more storms

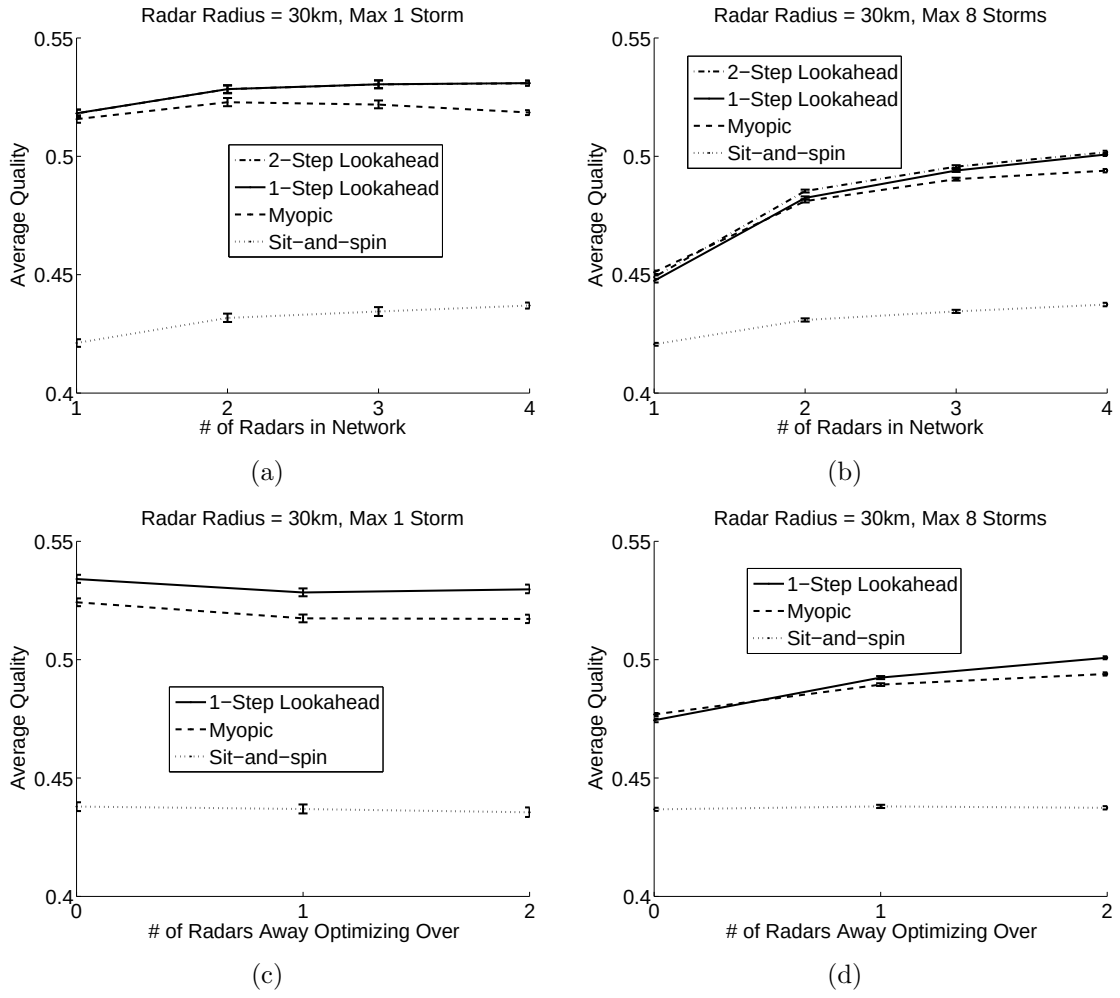


Figure 2.8. Scaling behaviour of sensing strategies. The simulation results in (a) and (b) vary the number of radars, while the results in (c) and (d) are for four radars. Radars are arranged as in Figure 2.3. Error bars are over 1500 runs; each run is 500 decision epochs long.

in each radar's footprint), thereby decreasing the quality with which each storm is individually scanned. Also observe that with fewer than four radars in the network, the two-step lookahead strategy achieves slightly higher average per-storm quality than the one-step lookahead strategy; for four radars, the one-step and two-step lookahead strategies achieve essentially the same quality.[3]

Next, Figures 2.8 (c) and (d) show the average per-storm quality as the number of radars over which each sensing strategy optimizes increases. For example, when the number of radars away that each sensing strategy optimizes over is zero, each radar only considers itself when computing its sensing strategy. When the number of radars away is one, each radar considers itself plus all radars one hop away when computing its sensing strategy. When the number of radars away is two, each radar considers itself plus all radars two hops away: with four radars in the network arranged as in Figure 2.3, this corresponds to optimizing over all radars. If a radar appears in multiple sub-sets over which to optimize, we determine the radar's sensing strategy in one subset, and then condition on that radar's sensing strategy in the other sub-sets. Note that Figures 2.8(a) and (b) show that the 1-step and 2-step lookahead strategies achieve the same average quality. Consequently, due to the computational cost, we compare with only the 1-step lookahead sensing strategy, not the 2-step strategy, in Figures 2.8(c) and (d).

For a maximum of one storm in the environment, Figure 2.8(c) shows that regardless of the number of radars over which strategies optimize, for a given sensing

---

[3]While the results shown in Figure 2.8 are for N radars arranged in an $N \times 1$ strip, we also consider arranging the radars in a 2×2 grid. For a 2×2 radar arrangement, we find that the average quality achieved by each sensing strategy is slightly higher than that achieved for the 1×4 arrangement. This is because it is even more likely with the 2×2 arrangement, versus the 1×4 arrangement, that a storm will be close to a radar and so will increase the F_d term of the quality. Depending on the strategy, the increases range from ~ 0.007 to ~ 0.016 for a maximum of one storm and from ~ 0.0067 to ~ 0.012 for a maximum of eight storms. Like the results in Figure 2.8 for a 1×4 radar arrangement, we also find for a 2×2 arrangement that there is little difference between the quality achieved by the 1-step and 2-step lookahead strategies for both a maximum of one storm and a maximum of eight storms.

strategy, the average per-storm quality is very similar. For a maximum of eight storms in the environment, Figure 2.8(d) shows for the myopic and one-step lookahead strategies that there are decreasing gains in quality when optimizing over more radars. For example, the gain in quality for the 1-step lookahead strategy going from optimizing over one radar away to two radars away is about half that of going from optimizing over zero radars away to one radar away. Hence, as the network size increases, sensing strategies do not necessarily need to optimize over all radars in the network.

2.8 Summary

In this chapter, we compared the performance of myopic and lookahead sensing strategies to address the problem of meteorological radar control. We showed that the main benefits of using a lookahead sensing strategy are when there are multiple meteorological phenomena in the environment, and when the maximum radius of any phenomenon is sufficiently smaller than the radius of the radars. More generally, the performance gains from considering expected future states of the environment depend on the speed and predictability of storm movement over a given time scale. We also showed that there is a trade-off between the average quality with which a phenomenon is scanned and the number of decision epochs before which a phenomenon is rescanned. Overall, considering only scan quality, we find that a simple lookahead sensing strategy can perform as well as a full lookahead strategy in our simulation scenarios. To additionally consider the size of the re-scan interval (or to optimize over multiple metrics of interest), a full lookahead strategy is useful.

One interesting direction for future work is to compute an upper bound on the quality that can be achieved for a given storm track trace and re-scan interval. This could potentially be done by using a limited lookahead sensing strategy and assuming deterministic storm movements.

CHAPTER 3

CALL ADMISSION CONTROL IN VIRTUALIZED PRIVATE SENSOR NETWORKS

3.1 Introduction

In Chapter 2, we examined the benefits of considering current and expected future states of the environment when adapting sensing strategies over time. In this chapter, we investigate how to mediate among different sensing strategies within the same sensor network, corresponding to different users that may make possibly conflicting requests of the sensing resources. We focus on limited lookahead sensing strategies, where users make (expected) sensor requests over some finite time horizon.

Consider again sensors with sensing constraints, such as cameras or radars that cannot simultaneously collect high fidelity data from all environmental locations. When such sensors are shared among multiple users, further sensing constraints are imposed on the sensors. In the CASA radar network [19, 62, 101], for instance, the radars are shared by meteorologists, emergency managers, and academic researchers although they have different needs. For example, once a tornado has been observed, an emergency manager may have no further need to observe the tornado's development and may wish to scan elsewhere, while an academic researcher may still be interested in further scans of the tornado. Thus, for such constrained sensors, not only are there sensing trade-offs when there is only a single user of the sensor network, but there are further trade-offs when there are multiple users.

Inspired by the GENI Project [27], we specifically consider the problem of call admission control (i.e., deciding which sensing requests to satisfy) in the context of

a virtualized private sensor network (VPSN).[1] Although multiple users may share the sensing resources of a virtualized network, in a VPSN each user works with the abstraction of having its own private network. Unlike traditional virtualized resources such as memory, bandwidth, or CPU cycles, however, the sensor requests made by one user may completely or partially satisfy those of another user. While other work has also considered the idea of a virtual sensor network [35], their focus is on how to adaptively select a subset of nodes in the sensor network to construct a "virtual sensor network" for a particular task, rather than virtualizing the sensing resources of the individual sensors.

We investigate a model in which a user's sensing strategy can be represented as a temporal sequence of sensor requests distributed across a time interval of length T. We define a sensor request as a request to use a sensor in a particular way, possibly at a particular time, with some associated user utility. Suppose that we have a set of sensing strategies, corresponding to different users, to schedule on the same set of sensors. *The call admission control problem in VPSNs is to select a non-interfering subset of sensor requests with maximum utility from among all of the sensing strategies.*

We show that the call admission control problem in VPSNs can be solved in polynomial time when sensor requests are divisible or fixed in time. When sensor requests are indivisible but may be shifted in time, we show that the VPSN call admission control problem is NP-complete, but that polynomial-time approximation schemes are possible.

The remainder of this chapter is structured as follows. In Section 3.2, we overview related work on virtual sensor networks and scheduling. In Section 3.3, we discuss

[1]The GENI project [27] is developing a shared testbed for investigating future internets and is including sensor networks as part of the shared testbed: hence multiple users will compete for sensing resources.

what we mean by a VPSN and formally define the call admission control problem in VPSNs. In Section 3.4, we present theoretical results and outline exact and approximate algorithms for the call admission control problem. Finally, in Section 3.5, we summarize our results.

3.2 Related Work

Although other work has also considered the idea of a virtual sensor network [35], their focus is on how to adaptively select a subset of nodes in the sensor network to construct a "virtual sensor network" for a particular task, rather than virtualizing the sensing resources of the individual sensors. Like our work, [54] also considers data sharing among users in a meteorological radar network. The work of [54], however, focuses on how to transmit data that has already been collected to maximize user utility, while our work focuses on what data should be collected. The idea of amortizing bandwidth usage over multiple users interested in the same data also arises in the multicast literature [18, 45, 97], but again, our focus is on data collection rather than data transmission.

Other work has looked at scheduling tasks in parallel and distributed systems [22, 21], real-time systems [77], and real-time control systems [81]. Such scheduling problems are usually NP-complete [21, 77, 84]. Project scheduling, where tasks have precedence constraints and start-time dependent costs, is solvable in polynomial-time when there are no resource constraints [63]. With resource constraints, project scheduling is also NP-hard [64]. The scheduling work most closely related to ours is that of interval scheduling [3, 44, 83] and is discussed in more detail in Section 3.4.

3.3 Virtualized Private Sensor Networks

In this section, we define what we mean by a virtualized private sensor network and formally define the call admission control problem in VPSNs.

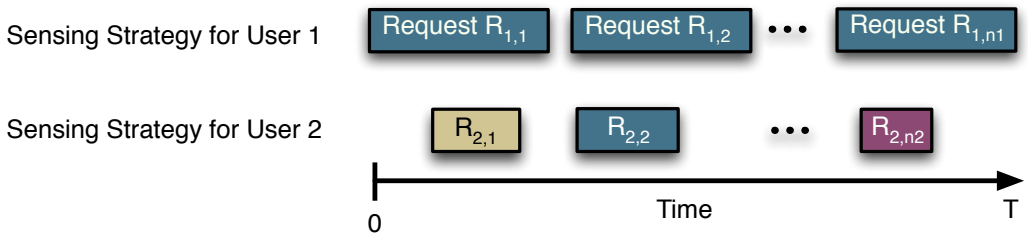


Figure 3.1. Notation for the call admission control problem. Requests that are the same colour, are requesting to use the sensor in the same way.

3.3.1 What do we mean by a virtualized private sensor network?

Consider a sensor network shared among multiple users. Users may have certain data that they would like collected by the network. As with running processes on a compute cluster, however, users do not necessarily care how their data is collected as long as the relevant data actually is collected. Consequently, the sensing resources can be virtualized, leading us to the idea of a virtualized private sensor network (VPSN). In a VPSN, each user has a virtual slice of the sensing resources available on the sensors. Although multiple users may share the sensing resources of the virtualized network, each user works with the abstraction of having its own private network. One benefit of virtualization here is to reduce the complexity of sharing the sensor resources among multiple users.

3.3.2 Call Admission Control Problem

We now formally define the call admission control problem in VPSNs. Suppose that there are N users of a VPSN and that each user has an associated sensing strategy. As shown in Figure 3.1, we define the sensing strategy for user i to be a temporal sequence of n_i sensor requests distributed over a time interval of length T.[2]

[2]The user sensing strategies that we consider in this chapter correspond to the finite lookahead sensing strategies of Chapter 2. In particular, the time interval of length T over which sensor requests are distributed corresponds to the finite horizon of a lookahead strategy. The sensor requests correspond to the current and expected radar sensing actions under consideration by a lookahead sensing strategy.

Informally, we define a sensor request to be a request that a sensor sense in a particular way (e.g., scan a specified 90° radar sector), possibly during a particular interval of time, and with some associated user utility. For example, a user's sensing strategy may be for a radar to scan a particular storm once every minute, with sensor requests corresponding to storm scans at the appropriate points in time. More formally, sensor request j of user i's sensing strategy, given by R_{ij}, has an associated start time t_{ij}^s and finish time t_{ij}^f. Since the sensor requests made by one user may satisfy those of another user, we define the utility $u_{ij}^{i'}$ for satisfying sensor request j of user i for all users $1 \leq i' \leq N$. *The call admission control problem in VPSNs is to select a non-interfering subset of sensor requests with maximum utility from among all of the sensing strategies.*

Whether the call admission control problem in VPSNs can be optimally solved efficiently depends on the constraints imposed on how and when utility may be received for executing a sensor request. As illustrated in Figure 3.2, the primary constraints on the utility function are as follows.

1. *Are the sensor requests of the sensing strategies divisible, i.e., must all of a sensor request be executed to receive utility?* For example, if the user requests that multiple elevations of a storm be scanned, utility is still received even when not all elevations are scanned. Alternatively, if the user requests that only a single elevation of a storm be scanned, there is no utility for scanning only part of the storm within that elevation.

2. *Can the sensor requests specified by a sensing strategy be shifted in time, i.e., must a sensor request be executed at the requested start time to receive utility?* For example, if a phenomenon is expected at a particular location at a particular time, and the sensor request is to scan that location at that time, there is no utility to scanning the location at another time. Alternatively, if the sensor

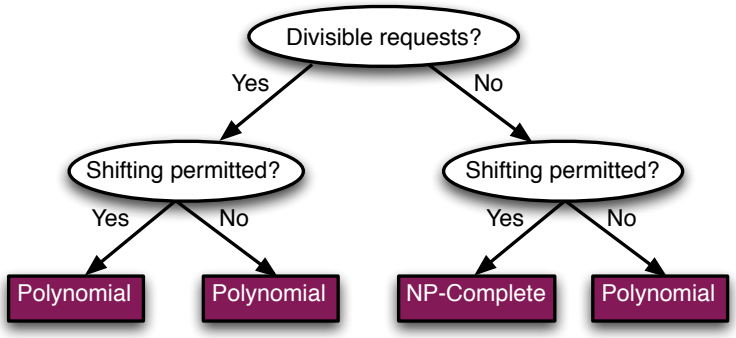


Figure 3.2. Computational complexity of the call admission control problem in VPSNs assuming different constraints on the utility functions.

request is a surveillance scan, the user will still receive utility if the surveillance scan is executed at a time other than the time requested.

3.4 Theoretical Results

We now examine the complexity of the call admission control problem in VPSNs under the different utility constraints discussed in Section 3.3.2; our results are summarized in Figure 3.2. As defined, the call admission control problem in VPSNs corresponds to the problem of interval scheduling [3, 44, 56, 83], and so we can leverage results from the interval scheduling literature to solve certain versions of the call admission control problem in VPSNs.

3.4.1 Indivisible Sensor Requests, No Shifting

Suppose that sensor requests are indivisible and that sensor requests are fixed in time. Then the call admission control problem in VPSNs can be solved in polynomial time by leveraging results from the interval scheduling literature [3]. Specifically, in the version of interval scheduling addressed in [3], a set of intervals is given where each interval represents a job to execute on a machine and has a fixed start time, end time, and utility. The goal is to select the set of non-interfering intervals that maximize utility. Sensor requests can be directly mapped to intervals. We then

Input

N sensing strategies:

Each sensing strategy $1 \leq i \leq N$ is comprised of n_i sensor requests, $R_{i1}, \ldots R_{in_i}$

Each sensor request R_{ij}, $1 \leq j \leq n_i$, has start time t_{ij}^s, finish time t_{ij}^f, and utility u_{ij}

Initialization

Identify all maximal sets of interfering sensor requests, $q_1, \ldots, q_r$, from sensing strategies

Set max_q =maximum$\{|q_1|, \ldots, |q_r|\}$

Algorithm

Create directed graph G:

Add nodes and arcs representing the maximal sets $q_1, \ldots, q_r$

Add nodes $v_0, \ldots, v_r$

Add arcs (v_i, v_{i-1}) with zero cost and infinite capacity, $i = 1, \ldots, r$

For each sensor request R_{ij}, $1 \leq i \leq N$, $1 \leq j \leq n_i$

If R_{ij} is in sets $q_k, \ldots, q_{k+1}$, add arc (v_{k-1}, v_{k+1}) with cost u_{ij} and capacity 1

For each maximal set q_k, $1 \leq k \leq r$

If $|q_k| < max_q$, add arc (v_{k-1}, v_k) with zero cost and capacity $(max_q - |q_k|)$

Formulate and solve minimum cost flow problem in graph G:

Select source v_0 and sink v_r

Fix flow size to be $(max_q - 1)$

Solution is least cost path from v_0 to v_r that can accommodate flow

Output

For each sensor request R_{ij}

If minimum cost flow uses arc representing sensor request R_{ij}, do not execute R_{ij}

Otherwise, execute R_{ij}

Table 3.1. Algorithm to solve the call admission control problem in VPSNs when sensor requests are indivisible and cannot be shifted in time; adapted from the algorithm for interval scheduling found in [3].

define $u_{ij} = \sum_{i'=1}^{N} u_{ij}^{i'}$ to be the utility of satisfying sensor request j of user i. These u_{ij} utilities can be directly mapped to the interval utilities. Thus, we can use the polynomial-time algorithm given in [3] to solve the call admission control problem in VPSNs when sensor requests are indivisible and fixed in time.

We present the algorithm from [3] in Table 3.1. As discussed in [3], the time complexity of the algorithm is $O(n^2 \log n)$, where n is the number of jobs. In the context of our call admission control problem, by summing over all sensor requests from all sensing strategies, we have $n = \sum_{i=1}^{N} n_i$. Dependencies among sensor requests can be dealt with as follows. Treat each set of dependent sensor requests as a single sensor

request by modifying how maximal sets of interfering sensor requests are identified in line 7 in Table 3.1. Specifically, we would say that a sensor request r interferes with a sensor request r', if either r or any sensor request on which r depends overlaps in time with either r' or any sensor request on which r' depends.

3.4.2 Divisible Sensor Requests, No Shifting

Now suppose that sensor requests are divisible but that requests are still fixed in time. Then this version of the call admission control problem in VPSNs can be formulated as an integer programming problem. We first sort the start times, t_{ij}^s and finish times, t_{ij}^f, of all $n = \sum_{i=1}^{N} n_i$ sensor requests in increasing order to obtain the sequence of times $t_1, \ldots, t_{2n}$ where t_1 is the earliest start time and t_{2n} is the latest finish time. We then find the maximum utility interleaving of sensor requests by finding the sensor request that maximizes the utility achieved during each time interval t_k to t_{k+1} with,

$$\text{Maximize} \quad \sum_{k=1}^{2n-1} \sum_{i=1}^{N} \sum_{j=1}^{n_i} x_{ij}^k \sum_{i'=1}^{N} \frac{u_{ij}^{i'}}{\Delta t_k}$$

where

$$x_{ij}^k = \begin{cases} 1 & \text{if request } R_{ij} \text{ is executed during time interval } t_k \text{ to } t_{k+1} \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta t_k = \begin{cases} t_{ij}^f - t_k & \text{if finish time } t_{ij}^f \text{ of request } R_{ij} \text{ is less than } t_{k+1} \\ t_{k+1} - t_k & \text{otherwise} \end{cases}$$

The time complexity of finding the maximum utility interleaving of sensor requests can then be computed as follows. Sorting the start times of the sensor requests takes time $O(n \log n)$, or can be done in linear time if the requests are already sorted within each sensing strategy. Finding the maximum utility sensor request within each

time interval takes time $O(nN)$. Since there are at most $2n$ time intervals, the time complexity is $O(n^2N)$.

3.4.3 Indivisible Sensor Requests, Shifting

Suppose now that the sensor requests are indivisible, but that the time at which a sensor request is started may be shifted in time, with the last request being executed by time T. We assume that the utility of a request is independent of its starting time. We can then formulate this version of the call admission control problem in VPSNs as the following integer programming problem,

$$\begin{aligned} \text{Maximize} \quad & \sum_{i=1}^{N} \sum_{j=1}^{n_i} x_{ij} \sum_{i'=1}^{N} u_{ij}^{i'} \\ \text{subject to} \quad & \sum_{i=1}^{N} \sum_{j=1}^{n_i} (t_{ij}^{f} - t_{ij}^{s}) x_{ij} \leq T \end{aligned}$$

where

$$x_{ij} = \begin{cases} 1 & \text{if request } R_{ij} \text{ is executed} \\ 0 & \text{otherwise} \end{cases}$$

This formulation of the call admission control problem corresponds to the 0-1 knapsack problem [41]. We thus show that this version of the call admission control problem in VPSNs is NP-complete as follows. First, given a specified subset of the sensor requests to execute, we can check whether the requests are feasible (i.e., do not overlap in time) by comparing the start and finish times of each sensor request with that of every other request in polynomial time. Consequently, this version of the call admission control problem is in NP.

We then use the 0-1 knapsack problem [41] to perform the reduction. An instance of the knapsack problem consists of a knapsack with integer capacity C, and a set of N objects with integer weights $w_1, w_2, \ldots, w_N$ and values $v_1, v_2, \ldots, v_N$. The goal is

to select the set of objects with maximum value whose total weight does not exceed the capacity of the knapsack. We reduce an instance of the knapsack problem to an instance of this version of the call admission control problem in VPSNs in polynomial-time as follows. We first set $T = C$. We then convert each knapsack object i directly to a sensing strategy (comprising only one sensor request) where the weight of object i becomes the time duration occupied by the sole sensor request of the new sensing strategy. The utility for executing the sole request of sensing strategy i is then v_i, regardless of the time at which the request is started. Thus, this version of the call admission control problem in VPSNs is NP-complete. To approximately solve this version of the call admission problem, one of the polynomial time approximation schemes for the single-dimensional knapsack problem [41] could be used.

We note that this version of the call admission control problem in VPSNs can also be formulated as an interval scheduling problem in which the intervals have a range of starting times, see [5, 83]; [5] also gives a pseudo-polynomial time algorithm.

3.4.4 Divisible Sensor Requests, Shifting

Finally, suppose that the sensor requests are divisible and that the time at which a sensor request is started may be shifted in time, with the last request being executed by time T. We assume that the sensor requests are arbitrarily divisible and that the utility associated with each new sub-request equals the utility of the original request times the fraction of time of the original request that the new request occupies. Then this version of the call admission control problem in VPSNs is exactly the fractional knapsack problem [41] where rather than maximizing utility per unit volume, we are maximizing utility per unit time. Since the fractional knapsack problem can be solved in polynomial time [41], this version of the call admission control problem in VPSNs can be solved in polynomial time. The intuition for the polynomial time algorithm is as follows. First, order requests by their utility per unit time. Then, select requests

occupying up to time T in total, starting with those requests with highest utility per unit time, and successively selecting requests with lower and lower utility per unit time. Finally, the last request selected may not necessarily be fully executed.

We note that if the utility of a sensor request were dependent on the starting time of the request, unlike what we assumed here, we do not expect that that the call admission control problem in VPSNs could be solved in polynomial time.

3.5 Summary

In this chapter, we examined how to accommodate different sensing strategies within the same sensor network, corresponding to different users who may make possibly conflicting requests of the sensing resources. We specifically considered the problem of call admission control (i.e., deciding which sensing requests to satisfy) in the context of a virtualized private sensor network. We showed that the call admission control problem in virtual private sensor networks can be solved in polynomial time when sensor requests are divisible or fixed in time. When sensor requests are indivisible but may be shifted in time, we showed that the call admission control problem in virtualized private sensor networks is NP-complete.

In future work, rather than assuming that all sensor requests are known *a priori*, we are interested in online versions of the call admission control problem where new sensor requests appear over time. Here there is related literature on online interval scheduling [44, 56]. We are also interested in decentralized methods to solve the call admission control problem. For example, rather than requiring the call admission controller to have global knowledge (particularly knowledge of all user utility functions), the controller could instead iterate back and forth with users to converge upon an acceptable solution (e.g., as is possible for routing [24]). Finally, there are interesting trade-offs between maximizing the utility of the sensor requests executed and

user fairness: one way to address the problem of fairness would be to require users to pay to have their requests satisfied.

CHAPTER 4

SEPARATION OF SENSOR CONTROL AND DATA TRAFFIC

4.1 Introduction

In Chapters 2 and 3, we focused on sensing strategies for sensor networks with constraints on the sensing resources. In this chapter, we examine how network constraints such as limited bandwidth due to network congestion impact sensing performance. In a sensor network, congestion can arise due to bursty and high-bandwidth data traffic, combined with wireless links and many-to-one data routing to a sink. Suppose then that data collected by the sensors must be transmitted over the network to a control center which computes new sensor controls (such as the radar scan actions in Chapter 2 or the user sensor requests in Chapter 3) based on the received data. If the control center receives insufficient data by the time a new sensor control must be computed, for instance due to network congestion, how does that impact the quality of the computed sensor control, the subsequent data collected under that control, and future sensor controls?

We examine these questions in the context of a *closed-loop sensor network*. In a closed-loop sensor network, sensors send data to a control center and sensor controls flow back to the sensors. The sensed data transmitted through the closed-loop sensor network may have considerable redundancy in both time and space making application performance somewhat insensitive to data packet loss and delay. Conversely, performance is typically much more sensitive to loss or delay of sensor control packets, since these packets carry the application's sensor commands generated in response

to received data. As delayed and dropped packets degrade the performance of the sensing application, network constraints may further exacerbate existing sensing constraints. Consequently, there are potential advantages to separate handling of sensor control and data traffic.

In this chapter, we specifically investigate the value of separate handling of sensor control and data traffic, during times of congestion, in a closed-loop sensor network. We first show that prioritizing sensor control traffic over data traffic decreasses the round-trip control-loop delay, and consequently increases the quantity and quality of the data collected by the sensor network. We then ground our analysis in a closed-loop meteorological sensor network [19, 62, 100], focusing on a storm-tracking application running over a network of X-band radars. The storm-tracking application measures reflectivity in the atmosphere and tracks storms (i.e., regions of high reflectivity) using a Kalman filter as in [59]; reflectivity is a measure of the number of scatterers in a unit volume of atmosphere known as a voxel. The reflectivity data are transmitted from the radars over a shared wireline [62, 100] or wireless [19] network to a control center that periodically generates radar targeting (sensor control) commands based on features detected in the data.

To evaluate the utility of separate handling of sensor control and data traffic in our storm-tracking application, we compare the performance of aggregate FIFO for both sensor control and radar data packets with that of priority forwarding of sensor control packets. Considering data quantity, we show that prioritizing sensor control traffic increases the number of voxels, V, that can be scanned given a constant number of reflectivity samples, N_c, obtained per voxel. Considering data quality, we show that prioritizing sensor control traffic increases the number of reflectivity samples, N, that can be obtained per voxel given a constant number of voxels, V_c, to scan. Since as N increases, sensing accuracy improves only as a function of $\sqrt{N}$, the gain in accuracy for the reflectivity estimate per voxel is relatively *small* except when prioritizing

sensor control increases N significantly (such as when sensor control packets suffer severe delays). Since prioritizing sensor control traffic also reduces the number of control packets dropped, enabling sensors to execute "correct" rather than default controls, data degradation is mitigated. Considering the performance of the tracking application, we show that during times of severe congestion, not prioritizing sensor control traffic can actually lead to tracking errors accumulating over time.

In one sense, our results mirror those in [23] regarding differential traffic handling and network-based (rather than sensor-application-based) performance metrics: "that performance is generally satisfactory in a classical best effort network as long as link load is not too close to 100%," and that "there appears little scope for service differentiation beyond the two broad categories of 'good enough' and 'too bad.'" *However, unlike some other network applications, the sensing application can still perform well during times of severe congestion, when sensor control packets are given priority, because (i) sensor controls are not dropped, so sensors execute "correct" rather than default sensor controls, and (ii) sensing accuracy both improves and degrades slowly in the number of sensed data samples obtained.*

While previous work [4, 7, 13, 38, 50] focuses on prioritizing network control packets, our focus here is on prioritizing sensor control packets. Whereas network control affects what data are transmitted and at what rate, sensor control additionally affects what data are actually sensed and thus available to be transmitted. Consider object tracking and suppose that the sensor controller incorrectly asks to sense data from one location in the environment when the object is at a different location. The data that should have been collected from sensing the different location cannot be collected retroactively, as the environmental conditions may have changed (i.e., the object may have moved) during the time that the incorrect location was sensed. Thus, in a closed-loop sensor network, the issue is not just that data may be received late, but that the opportunity to ever sense some data may be missed completely. Other work

in sensor networks has considered service differentiation [6, 39, 91], including during times of congestion [47], but does not specifically look at the effects of prioritizing sensor control nor consider closed-loop sensor networks.

The remainder of this chapter is structured as follows. In Section 4.2, we overview related work. In Section 4.3, we discuss general characteristics of a closed-loop sensor network. In Section 4.4, we describe our meteorological sensing application. In Section 4.5, we present simulation results comparing the performance of FIFO with that of priority forwarding of sensor control packets. Finally, in Section 4.6, we summarize our results.

4.2 Related Work

The notion of separate handling of control and data packets in a network has a long history. The SS7 signaling system [13] that carries control packets in telephone networks is a packet-switched control network that is physically separate from the circuit-switched network carrying voice traffic. In ATM networks, Q2931 signaling packets for virtual circuit management are carried over connections that are logically separated from data traffic [7]. For IP networks, it is possible for operators to configure routers to provide prioritized service for "control" protocols such as BGP or SNMP. In wireless networks, [50] advocates for a separate control channel for controlling access to a shared medium. Proposals for priority handling of TCP acknowledgments [4, 38] can also be considered as providing a different level of service to control packets (ACKs) than data packets. While this previous work focuses on prioritizing network control packets, our focus here is on prioritizing sensor control packets.

Other work, in sensor networks, has considered service differentiation for different classes of traffic. [6] assigns priority levels to packets, forwarding higher-priority packets more frequently over more paths to achieve higher probability of delivery. [39] allocates rates to flows based on the class of traffic being sent and the estimated

load on the network. [91] considers bandwidth reservation for high-priority flows in wireless sensor networks. [47] proposes congestion-aware routing in sensor networks, providing differential service to high priority data traffic versus low-priority data traffic in congested areas of the network. None of these approaches, however, considers the effects of prioritizing only sensor control in a closed-loop sensor network.

Control theory considers the effects of a network within the control loop in the field of "Networked Control Systems" [30]. As in a closed-loop sensor network, data and sensor control are sent over a network. Unlike in a closed-loop sensor network, however, the sensor control and data packets in a Networked Control System are constrained to be the feedback (sensor control) and measurements (data) of a classical control system. Consequently, the ratio of data to control is much smaller than that of a closed-loop sensor network such as our radar network [62, 100]. Since any data packet (i.e., measurement) in a Networked Control System may now be as important as any sensor control packet (i.e., feedback), it is not necessarily beneficial to always give higher priority to sensor control. Instead, packets are scheduled to optimize expected performance using the control equations, for instance by incorporating the error incurred due to network delays directly into the control equations [96], or by optimally dropping selected data measurements during times of overload by analyzing the effect of the resulting missing measurements on the control equations [53]. Networked Control Systems can thus be considered a specific sub-class of the more general closed-loop sensor networks we consider in this work.

4.3 Closed-Loop Sensor Networks

A generic closed-loop sensor network is shown in Figure 4.1: data are streamed from sensors to a control center, while sensor commands flow from the control center back to the sensors. The control center closes the system's main control loop by ingesting data, computing statistics from the data, and selecting each sensor's future

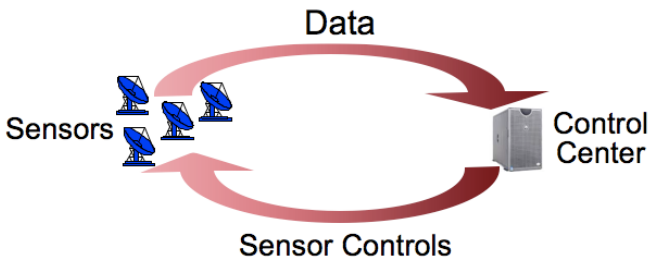
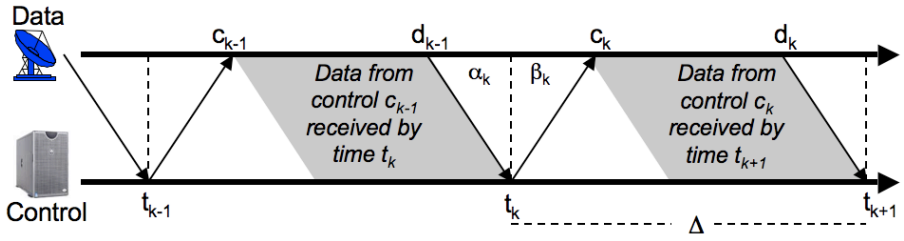


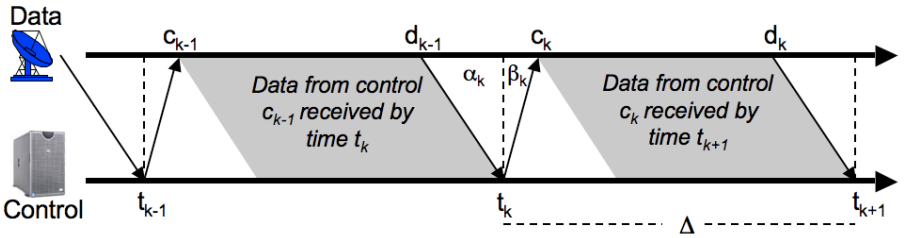
Figure 4.1. A closed-loop sensor network.

data collection strategy based on the statistics. As shown in Figure 4.2, a closed-loop sensor network periodically computes a new sensor control. We assume wireless links for the analysis in Figure 4.2: thus, sensor control and data packets must compete for access to links and so can be modeled as if they share the same queue. At the start of the kth update interval, t_k, the control center issues a command to the sensor specifying how to collect data. After a delay of β_k, the command is received at the sensor. The sensor then begins transmitting back measured data; α_k is the delay of data from the sensor to the control center before the kth control update interval. After time Δ, the sense-and-response cycle then repeats. We assume that the duration, Δ, of each control update interval is fixed, but the length of Δ could also depend on time or another metric.

From Figure 4.2, the sensor control computed at time t_k is based on data that the sensor sent by time $t_k - \alpha_k$. The sensor control is then applied at the sensor at time $c_k = t_k + \beta_k$. When computing sensor control c_{k+1}, we assume that it is preferable to use only the most recent data, obtained with sensor control c_k. Thus, while data obtained under sensor control c_{k-1} could continue to be transmitted by the radars during time d_{k-1} to time c_k, and could additionally be used to compute sensor control c_{k+1}, this data is now out-of-date, and we assume that it is not transmitted. While not considered here, such out-of-date data could also be sent as low priority background traffic.



(a) $\alpha_k \approx \beta_k$



(b) $\alpha_k > \beta_k$

Figure 4.2. Timing of the control loop when (a) $\alpha_k \approx \beta_k$ and (b) $\alpha_k > \beta_k$. We assume wireless links: thus, sensor control and data packets must compete for access to links and so can be modeled as if they share the same queue.

Suppose now that packets are delayed: then α_k and β_k will increase and the total amount of data from control c_k received by time t_{k+1} at the control center will decrease. Since the ratio of data to sensor control traffic is large, it should be possible to provide significantly better performance to sensor control traffic (e.g., lower end-to-end delays and lower loss) with only a minimal performance degradation of the data traffic, as illustrated in Figure 4.2(b) and in keeping with queueing theoretic conservation laws [42]. This then decreases the "round trip" delay for the control loop - the summed delay of sensed data from a sensor to the control center and the delay of sensor control packets back to the sensor. Lower round-trip delays enable the control center to examine more data before making a control decision, resulting in more accurate estimates of the sensed quantity of interest, which should then give better application-level performance. Consequently, prioritizing sensor control in a closed-loop sensor network should produce both *more data* and *better quality data*. We explore these two benefits in more detail below.

4.3.1 More Data

One benefit of prioritizing sensor control is to allow more data to be collected in time to compute the next sensor control. The additional data collected when sensor control is prioritized can, for instance, be collected from additional environmental locations that would not have been sensed if sensor control were not prioritized. In a meteorological sensing network, sensing more environmental locations results in the radar sensing more voxels. Equivalently, in a camera network [2] comprised of pan-tilt-zoom cameras, sensing more environmental locations translates to the camera collecting images from more locations. In both the radar and camera networks, this additional data increases the probability that an object of interest (storm, person, car, etc.) is detected. From p. 35 of [14] "the data processing inequality can be used to show that no clever manipulation of the data can improve the inferences that can be made from the data": i.e., further processing of the data will not produce more information than that contained in the original data. Assuming utility depends on the amount of information contained in the data, we expect that the utility gain from additional data is at most a linear function of the amount of additional data obtained. Note that some minimum amount of (additional) data may be needed before this or any non-zero utility is achieved.

To quantify the amount of additional data obtained when prioritizing sensor control, consider the effect of the length of the control update interval Δ. For FIFO scheduling, data are collected during a time interval of length $\Delta - \alpha_k - \beta_k$, while for priority scheduling, data are collected during a time interval of at most length $\Delta - \alpha_k$. Consequently, priority scheduling has a percentage gain of at most $\beta_k/(\Delta - \alpha_k - \beta_k)$ more time over FIFO. As Δ decreases, the percentage gains from priority scheduling thus increase. As Δ increases, although the total amount of data that is collected will increase, the gains from priority scheduling will decrease.

4.3.2 Better Quality Data

Another use of the additional data obtained by prioritizing sensor control is to improve data quality. In this case, rather than collecting data from more environmental locations, additional data samples would be collected from the same environmental location. In a meteorological sensing network, more data samples from the same environmental location translates to more reflectivity samples per voxel, thereby decreasing the standard deviation of the reflectivity estimate for the voxel, see Section 4.4.2. Equivalently, in an acoustic sensor network, more signals from the same environmental location can be used to perform signal averaging to reduce noise [17], while in a camera sensor network, more images from the same environmental location can be used to perform image averaging to reduce noise [9].

To quantify the gain in data quality when prioritizing sensor control, consider the set of i.i.d. data samples $X_1, \ldots, X_n$ collected during time $t = \Delta - \alpha_k - \beta_k$. Increasing t linearly increases the number of samples n collected. Suppose, however, that we use those samples to compute an unbiased estimator $W(\mathbf{X})$ of some parameter θ (such as reflectivity or temperature). The Cramer-Rao bound [10] says that the standard deviation of $W(\mathbf{X})$ from θ, $\mathrm{SD}_\theta(W(\mathbf{X}))$, can be lower bounded as follows,

$$\mathrm{SD}_\theta(W(\mathbf{X})) \;\geq\; \frac{1}{\sqrt{n\,\mathcal{I}}} \tag{4.1}$$

where $\mathcal{I}$ is the Fisher information, representing the information a sample contains about the estimator $W(\mathbf{X})$. As $\mathrm{SD}_\theta(W(\mathbf{X}))$ decreases only at the rate of $1/\sqrt{n}$, sensing accuracy improves slowly in the number of sensed data values obtained. For our meteorological radar network (and for acoustic [17] and camera [9] sensor networks), the reflectivity standard deviation when averaging over n i.i.d data samples decreases as a function of $\sqrt{n}$.

Even during times of packet loss, not just packet delays, prioritizing sensor control improves data quality. Consider an overloaded network in which packets may be

dropped. Since prioritizing sensor control limits the number of sensor control packets dropped, suppose that only data packets are dropped. As the number of i.i.d. data samples, n, decreases, the standard deviation of any estimator increases only at the rate of $1/\sqrt{n}$. Thus, sensing accuracy also degrades slowly (i.e., the standard deviation increases slowly) as the number of data samples, n, decreases. Additionally, because sensor control packets are not dropped, sensors are able to execute the "correct" sensor controls (instead of executing a default control such as having a radar scan 360° or a camera take low-quality images of all environment locations), thereby obtaining even better quality data. Consequently, the sensing application may still perform well during times of network overload, if data packets, but not sensor control packets, are dropped.

4.4 Meteorological Application

In this section, we describe the networked meteorological remote sensing application, in the context of the CASA radar network [11, 48], that we use to illustrate and quantitatively explore the effect of prioritizing sensor control on data quantity and quality, and on application-level performance. As shown in Figure 4.1, remote X-band radars transmit measured reflectivity values to a control center. The Meteorological Command and Control (MC&C) [100] component at this control center identifies meteorological features from the radar data, reports the features to end-users, and determines each radar's future scan strategy (i.e., the volume of the atmosphere to be scanned by each radar). A 4-node system has been developed and deployed in southwestern Oklahoma as part of the CASA project [11]. As shown in Figure 4.2, the system operates on a Δ=30-second control update interval. We now describe our network and radar meteorology models, and the storm-tracking application running over the network.

4.4.1 Network Model

In this section, we describe how we model the effect of prioritizing sensor control on the data delay, α_k, and sensor control delay, β_k, during congestion, as well as how we model packet drops. To model packet delays, we assume a wireless network where data is sent from radars (sources) to a control center (the sink), and sensor control commands are sent back to the radars from the control center. We analyze the packet delays incurred at a bottleneck link, assuming other network delays are small enough to be ignored. In a wireless sensor network, the bottleneck link might be the last hop node before the sink. We consider two scheduling mechanisms: *(1)* aggregate FIFO service of sensor control and data packets and *(2)* nonpreemptive priority forwarding of sensor control packets. For instance, when using 802.11, priority forwarding over the wireless links could be done as in [69] by assigning queuing, waiting, and back-off times based on priority level.

Figure 4.3(a) shows our queue model of the bottleneck link. We group traffic contending for the bottleneck queue into three flows: sensor *control* traffic destined for some node r, *data* traffic generated by node r, and *other* traffic including data and sensor control traffic either generated by or destined for nodes other than node r. Although data and sensor control packets might always travel in opposite directions in a sensor network, due to the wireless links, data and sensor control packets will still compete against each other for access to the wireless link; consequently, we model the bottleneck link as a single queue. For wired links, data and sensor control packets may end up in the same outgoing queue when there are, e.g., (i) multiple control centers, (ii) multiple sensor applications using the same network, or (iii) asymmetric routing.

Since data and sensor control are generated at deterministic intervals in the CASA network, we assume that sensor control and data packets have deterministic arrivals with rates λ_c and λ_d respectively. "Other" packets arrive according to a two-state

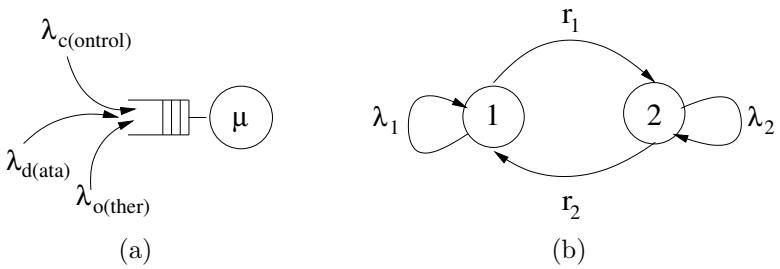


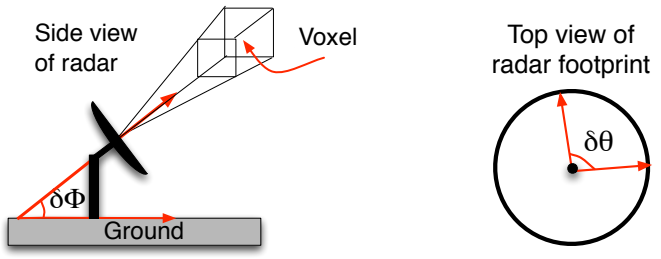
Figure 4.3. (a) Model of the bottleneck queue in the network. (b) The 2-state Markov modulated Poisson process used to model the "other" traffic.

Markov-modulated poisson process, see Figure 4.3(b), where packets arrive on average at rate λ_o; in state 1 packets arrive at rate λ_1, in state 2 packets arrive at rate λ_2, and transitions from states 1 to 2 and from states 2 to 1 occur at rates r_1 and r_2 respectively. To vary the burstiness, as measured using the index of dispersion, see [29], we vary the values of λ_1 and λ_2 while keeping λ_o constant. Finally, we assume that the packet service time is exponentially distributed with rate μ. To compute the delays through the bottleneck queue, we use the ns-2 simulator [1], see Section 4.5.

To model packet drops, e.g., due to overload, we compare the worst-case and best-case scenarios. For the worst-case scenario, we assume that all sensor control packets are dropped and that the default sensor control must be used. For the best case scenario, we assume that no sensor control packets are dropped and that the scan strategy specified by the sensor controls always collect data that is optimal for the application performance metrics defined in the next section.

4.4.2 Radar Meteorology Model

In this section, we describe the radar meteorology model we use to evaluate the effect of prioritizing sensor control on application performance. A radar operates by sending out pulses at a given rate as it sweeps through the sector it is scanning. For a given time duration, the smaller the sector scanned, to some minimum sector size, the better the estimated reflectivity values, since the radar can send more pulses per



(a) (b)

Figure 4.4. Radar definitions. (a) Side view of radar. (b) Top view of radar.

volume of atmosphere, see [20]. Meteorological algorithms use reflectivity values to identify, e.g., storms and tornados. For a more detailed primer on meteorological radars, see Section 2.3. We now describe the application performance metrics of interest.

4.4.2.1 Number of Voxels Scanned

Suppose that the number of pulses, N_c, transmitted per voxel is fixed, where a voxel is a unit volume of atmosphere. Then the simplest metric of interest is the number of voxels, V, that can be scanned during time $\Delta - \alpha_k - \beta_k$, given by,

$$V = \frac{(\Delta - \alpha_k - \beta_k) f_p}{N_c} \tag{4.2}$$

where $f_p = 3$ kHz is the pulse repetition frequency. We ignore here how the voxels are distributed to form a sector scan. As $\Delta - \alpha_k - \beta_k$ increases, the number of voxels V that can be scanned, each with N_c pulses, increases linearly.

4.4.2.2 Reflectivity Standard Deviation

We now relax the assumption that the number of pulses transmitted per voxel is constant. We focus here on the quality of the reflectivity metric estimated from the

data. The number of pulses, N, transmitted per voxel given a constant number of voxels, V_c, and during a time interval of length $\Delta - \alpha_k - \beta_k$ is [20],

$$N = \frac{(\Delta - \alpha_k - \beta_k)\, f_p}{V_c} \tag{4.3}$$

where $f_p = 3$ kHz is the pulse repetition frequency, $V_c = \frac{\delta\theta\,\delta\phi}{\theta\,\phi}$, $\delta\theta$ is the size of the sector scanned in degrees, see Figure 4.4, $\theta = 1.8°$ is the antenna beamwidth, $\delta\phi = 12°$ is the elevation height, and $\phi = 2°$ is the elevation step (i.e., the increase in elevation after a horizontal scan). Following Equation 4.3, as the sector size $\delta\theta$ decreases, more pulses can be transmitted per voxel.

Each pulse transmitted per voxel returns an estimate of the reflectivity for that voxel. Reflectivity is a measure of the number of scatterers in a volume of atmosphere. Averaging over more samples increases the confidence in the estimated reflectivity value. Given N samples for a voxel, the reflectivity standard deviation, $\hat{\sigma}_r$, for the voxel is [20]:

$$\hat{\sigma}_r = 1 + \sqrt{\frac{1}{N}\left(\left(1 + \frac{1}{S_n}\right)^2 + \left(\frac{1}{S_n}\right)^2\right)} \tag{4.4}$$

where S_n is the signal to noise ratio and has a typical value of 10dB. Computing $\sigma_r = 10\log_{10}(\hat{\sigma}_r)$, we obtain the reflectivity standard deviation in decibels (dB). While increasing $\Delta - \alpha_k - \beta_k$ linearly increases the number of samples N collected, the standard deviation of the estimated reflectivity value of the voxel decreases only at the rate of $1/\sqrt{N}$, exemplifying the Cramer-Rao result seen in Section 4.3.

4.4.2.3 Tracking Error

Both the number of voxels scanned and the reflectivity standard deviation evaluate system performance only within a single control update interval, Δ. To capture whether per-interval gains accumulate across multiple intervals, we look to simulations

tracking a storm (i.e., a region of high reflectivity). We first discuss how we convert the standard deviation of reflectivity to the standard deviation of the location of peak reflectivity (i.e., the location of the storm centroid). We then describe how the standard deviation of the location of the storm centroid affects the root mean-squared error (RMSE) when tracking the centroid.

The reflectivity standard deviation, σ_r, depends, through N in Equation (4.3), on the scan sector size, $\delta\theta$, and the time spent scanning the sector, $\Delta - \alpha_k - \beta_k$. An increase in σ_r should translate into an increase in the standard deviation of the location of the peak reflectivity, σ_z. As there are many algorithms for detecting peak reflectivity, and the uncertainty associated with the location depends on the algorithm, we adopt a simple approach here and set the value of σ_z along a radial from the radar as,

$$\sigma_z = \frac{\sigma_r D_r}{30dB} \tag{4.5}$$

where D_r is the distance of the object from the radar and 30dB is a mid-range reflectivity value. This assumes that uncertainty in the reflectivity estimate translates into an equivalent amount of uncertainty in the location of peak reflectivity.

We use the standard deviation in the location of peak reflectivity in the covariance matrix of the Kalman filter used to track storms, described in the next section. For meteorological algorithms, it is not sufficient to scan only the storm centroid. Instead, reflectivity data from the surrounding area (i.e., the entire storm cell) is also needed [100]. Hence our experiments will perform tracking based on the location of the storm centroid, but will also scan the surrounding area, corresponding to the expected storm radius.

4.4.3 Storm-Tracking Application

In this section we describe a storm-tracking application. Due to a limited amount of real storm track data, we use the following model to generate traces of storm movement through the environment. Let $\mathbf{x}_k$ be the true location of the storm centroid at time k and let $\mathbf{y}_k$ be noisy measurements of the location. A Kalman filter [98] assumes that the true location at time k is a linear function of the true location at time $k-1$ plus Gaussian noise, and that the noisy measurements at time k are a linear function of the true location at time k plus Gaussian noise. I.e.,

$$\mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + N[\mathbf{0}, \mathbf{Q}_k] \tag{4.6}$$

$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k + N[\mathbf{0}, \mathbf{R}_k] \tag{4.7}$$

We now describe our Kalman filter model of the movement of a storm centroid. We use $\mathbf{x}_k = [x^1, x^2, x^3, x^4]^T$, where x^1 is the true x-location of the storm centroid, x^2 is the true y-location, x^3 is the true x-velocity, and x^4 is the true y-velocity. For the noisy measurements, we use $\mathbf{y}_k = [y^1, y^2]^T$, where y^1 is the measured x-location and y^2 is the measured y-location. Then,

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & \Delta & 0 \\ 0 & 1 & 0 & \Delta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & q^3 & 0 \\ 0 & 0 & 0 & q^4 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad \mathbf{R}_k = \begin{bmatrix} r^1 & 0 \\ 0 & r^2 \end{bmatrix}_k$$

We obtain the covariance matrix $\mathbf{Q}$ as follows. First, we assume that the latitude and longitude noises are uncorrelated and set the off-diagonal elements of $\mathbf{Q}$ to zero. We also assume that there is no noise in the latitude and longitude locations. We then compute the noise in the latitude and longitude velocities from 39 existing storm

tracks from the National Severe Storms Laboratory courtesy of Kurt Hondl and the WDSS-II software [31]. Each track is a series of $(latitude, longitude)$ coordinates. We first compute the differences in latitude and longitude, and in time, between successive pairs of points. This gives us data on the latitude and longitude velocities. We then fit the latitude velocity data with a Gaussian distribution, and fit the longitude velocity data with another Gaussian distribution. Since the length of a latitude degree at 40° latitude equals 111.04 km and the length of a longitude degree at 40° latitude equals 85.39 km, we obtain, in units of km/hour, that the latitude velocity is $\sim N(9.1, 1268)$ and that the longitude velocity is $\sim N(16.7, 836)$. For example, the latitude velocity is on average 9.0 km/hr with one standard deviation of $\sqrt{1268} = 35.6$ km/hr. Working in seconds, we set $q^3 = 0.0001$ and $q^4 = 0.00006$. While the process noise $\mathbf{Q}$ is not a function of time k, the measurement noise $\mathbf{R}_k$ is, as it depends both on the radar scan strategy at time k, and on the delay given by $\alpha_k + \beta_k$. Thus, at time k, we compute σ_z as in Equation (4.5) and set $r^1 = r^2 = \sigma_z^2$.

We note that due to the $\mathbf{A}$ and $\mathbf{B}$ matrices that we use in the Kalman filter, it is possible to further simplify the Kalman filter update equations from [98] as follows. Simplifying Equations 4.6 and 4.7 we have that,

$$x^1_{k+1} = x^1_k + \Delta x^3_k + w^1_k \tag{4.8}$$

$$x^2_{k+1} = x^2_k + \Delta x^4_k + w^2_k \tag{4.9}$$

$$x^3_{k+1} = x^3_k + w^3_k \tag{4.10}$$

$$x^4_{k+1} = x^4_k + w^4_k \tag{4.11}$$

where $\mathbf{w}_k = [w^1_k, w^2_k, w^3_k, w^4_k]^T$, $\mathbf{v}_k = [v^1_k, v^2_k]$, $\mathbf{w}_k \sim N[0, \mathbf{Q_k}]$, $\mathbf{v}_k \sim N[0, \mathbf{R_k}]$. We can then compute, for instance, the estimate of the first component of the state vector, $x^1_{k|k}$, at time k using Equation 4.16 derived as follows. We use the notation $x^i_{k|k'}$ to indicate that the estimate of the ith component of the state vector at time k is

conditioned on all observations up to and including time k'.

$$\begin{aligned} x^1_{1|0} &= x^1_0 + \Delta x^3_0 + w^1_0 && (Predict) \\ x^1_{1|1} &= x^1_{1|0} + \mathbf{K}^1_0(\mathbf{y}_0 - \mathbf{x}_0) && (Update) \end{aligned} \tag{4.12}$$

$$\begin{aligned} x^1_{2|1} &= x^1_{1|1} + \Delta x^3_1 + w^1_1 && (Predict) \\ x^1_{2|2} &= x^1_{2|1} + \mathbf{K}^1_1(\mathbf{y}_1 - \mathbf{x}_{1|1}) && (Update) \end{aligned} \tag{4.13}$$

$$\begin{aligned} x^1_{k|k-1} &= x^1_{k-1|k-1} + \Delta x^3_k + w^1_k && (Predict) \\ x^1_{k|k} &= x^1_{k|k-1} + \mathbf{K}^1_{k-1}(\mathbf{y}_k - \mathbf{x}_{k-1|k-1}) && (Update) \end{aligned} \tag{4.14}$$

where $\mathbf{x}_0$ is the initial state estimate, which we assume to directly be the observations $\mathbf{y}_0$. $\mathbf{K}_k$ is the Kalman gain at time k, see [98], which does not depend on the state or observations. Thus, the full machinery of the Kalman filter is not needed to model the storm dynamics that we assume. To allow this work to generalize to other storm dynamics, however, we will continue to use the Kalman filter model in this work.

We use the Kalman filter parameters described above in the tracking algorithm used in Section 4.5. As the system is not directly observable, it can be difficult to exactly obtain the covariance matrices in practice. Consequently, to parameterize the Kalman filter used to generate the trajectory of the storm centroid, we use the same parameters as the Kalman filter used for tracking, but we perturb the covariance matrices as follows.

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & q^3 & 0 \\ 0 & 0 & 0 & q^4 \end{bmatrix}, \quad \mathbf{R}_k = \begin{bmatrix} 5r^1 & 0 \\ 0 & 5r^2 \end{bmatrix}_k$$

We now describe the storm-tracking application. Assume that the radar is located at the origin, that the radar radius is 40km (e.g., as with the CASA radars [100]), and that the radar stops tracking a storm when it exits the radar's footprint. We represent a storm as a circle with a 3km radius based on work by [79] on storm cells which gives 2.83km as "the radius from the cell center within which the intensity is greater than e^{-1} of the cell center intensity"; the initial location of the storm centroid is chosen randomly and subsequent movement is governed by Equation 4.6. To compute the measured location of the storm centroid, $\mathbf{y}_k$, we use Equation 4.7, using $\Delta - \alpha_k - \beta_k$ and the procedure described earlier to obtain the parameters for the covariance matrix $\mathbf{R}_k$. To compute the estimated true location $\hat{\mathbf{x}}_k$ from $\mathbf{y}_k$, and to compute the predicted true location $\hat{\mathbf{x}}_{k+1}^-$ and covariance matrix $\mathbf{P}_{k+1}^-$, we use the filtering equations, e.g., see [98]. To compute the area that contains $\hat{\mathbf{x}}_{k+1}^-$ with 99% confidence, we use its covariance matrix $\mathbf{P}_{k+1}^-$. The 99% confidence area is an ellipse centered at the point $(\hat{x}_{k+1}^1, \hat{x}_{k+1}^2)$ whose semi-axes are given by the submatrix $\mathbf{P}_{k+1}^-[1,2;1,2]$. $\hat{x}_{k+1}^1$ and $\hat{x}_{k+1}^2$ are, respectively, the x- and y-locations of the storm centroid and are the first two components of the vector $\hat{\mathbf{x}}_{k+1}$. To account for the storm radius, we expand the confidence ellipse by 3km (since the ellipse gives the area in which the storm centroid will be found 99% of the time, but does not include the storm radius). We compute the radar's next scan sector to be the smallest scan angle that covers the expanded confidence ellipse. The radar then scans this sector for $\Delta - \alpha_{k+1} - \beta_{k+1}$ seconds during the next update interval; the radar scans 360° initially, whenever the true location lies outside of the scanned area, and when $\alpha_k + \beta_k \geq \Delta$.

4.5 Simulation Results

In this section, we use the models described in Section 4.4 to investigate the value of prioritizing sensor control traffic over data traffic in our illustrative closed-loop meteorological sensing network. We first examine the effect of prioritizing sensor

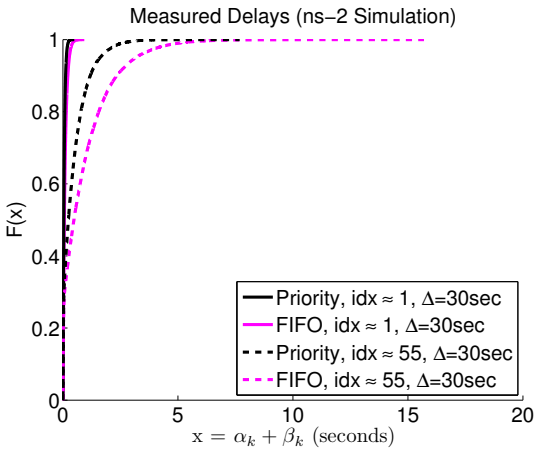
control on data quantity and quality, and then examine the effect on the performance of the tracking application.

4.5.1 Simulation Set-up

Delayed packets. To obtain the control-loop delays we use the ns-2 simulator [1]. We set the queue size to be large enough that no packets are dropped. The data delay is the delay incurred by the last packet that is processed at the bottleneck node by the start of each update interval. The corresponding control delay is the delay of the associated sensor control packet for that update interval. Based on experimental results from the CASA radar testbed, we use $\frac{\lambda_c}{\lambda_c+\lambda_d} = 0.0005$ and $\Delta = 30$ sec, setting $\lambda_c = \frac{1}{30}$ pkts/sec and $\lambda_d = \frac{2000}{30}$ pkts/sec. We also use $\Delta = \{5, 15\}$ sec, setting $\lambda_c = \frac{1}{\Delta}$ pkts/sec while leaving λ_d unchanged; such Δs are feasible for a phased array radar [80]. For the "other" traffic, we set $\lambda_o = \frac{2000}{30}$ pkts/sec, with $\lambda_1 = p\lambda_o$ and $\lambda_2 = (1-p)\lambda_o$, for $p = \{0.5, 0.2, 0.05\}$. We set the transition rates r_1 and r_2 for the Markov modulated Poisson process to each be 1.0 sec on average. Computing the index of dispersion (idx), see [29],

$$idx = 1 + \frac{2(\lambda_1 - \lambda_2)^2 r_1 r_2}{(r_1 + r_2)^2(\lambda_1 r_2 + \lambda_2 r_1)} \tag{4.15}$$

shows that our parameters consider $idx \approx \{1, 25, 55\}$. $idx = 1$ corresponds to a Poisson process while larger values correspond to increased traffic burstiness. Finally, since $\lambda_c + \lambda_d + \lambda_o \approx 133.37$ pkts/sec we set $\mu = 148.5$ pkts/sec achieving a load of about 0.90. Even with $\frac{(\lambda_c+\lambda_d+\lambda_o)}{\mu} < 1$, however, the bursty "other" traffic introduces temporary overload conditions. Using this network model, for each parameter setting, we perform 10 simulation runs, of 100,000 sec each. This gives, for instance, 20,000 update intervals per run for $\Delta = 5$ sec. For each run we obtain a time-varying series of $\alpha_k + \beta_k$ delays. Figure 4.5 shows the delay distributions for $\Delta = 30$ sec; we plot the data from all runs to obtain the CDF for each scheduling mechanism and idx



(a)

Figure 4.5. CDFs of the measured $\alpha_k + \beta_k$ delays.

pair. Although not shown, we also find that on average, the $\alpha_k + \beta_k$ delay for priority scheduling is about half that of FIFO, regardless of Δ (we observe a maximum average delay of ~ 0.9 sec for FIFO with $\Delta = 5$ sec and $idx = 55$). While we expect that increasing λ_o will increase the $\alpha_k + \beta_k$ delays, recall from Section 4.3 that prioritizing sensor control has a percentage gain of at most $\beta_k/(\Delta - \alpha_k - \beta_k)$ more time over FIFO. Consequently the relative performance gain of priority over FIFO should be bounded regardless of λ_o.

Dropped packets. To model packet drops only, we compare the worst-case (all sensor control dropped) and best-case (no sensor control dropped) scenarios. We assume that the sensor control packets always tell the radar to scan 45° and two elevation angles within that sector (i.e., the smallest sector that would be scanned by the CASA radars, and correspondingly, the highest quality data that would be obtained). Consequently N_{45} samples per voxel would be collected in the specified 45° sector. As a result of overload, we assume that a fraction p_{loss} of packets are lost. For both FIFO and priority scheduling, a fraction of the data samples will be lost. Additionally for FIFO, however, since we assume the worst case, all sensor control packets will be lost, and the radars will always use the default strategy of scanning

360° and all six elevation angles (i.e., the largest volume of space that the CASA radars would scan), collecting N_{360} samples per voxel.

4.5.2 Data Quantity and Quality Results

Effect of packet delays on number of voxels scanned. Substituting the delays generated from ns-2 into Equation 4.2, Figure 4.6(a) shows the number of times more voxels scanned under priority scheduling than under FIFO. Figure 4.6(a) shows that as Δ decreases and burstiness increases, the benefits of prioritizing increase: for $\Delta = 5$ sec and $idx = 55$, priority scheduling scans about 1.15 times as many voxels as FIFO.

Effect of packet delays on reflectivity standard deviation. Substituting the delays generated from ns-2 into Equation 4.3, for $\delta\theta = 360°$ we obtain a time-varying series of Ns. The empirical CDFs for N are shown in Figure 4.6(b), using the data from all 10 runs for each CDF: we see that FIFO and priority each achieve about 6× as many pulses for $\Delta = 30$ sec as for $\Delta = 5$ sec and that the total number of pulses gained over FIFO from using priority is independent of Δ. Figure 4.6(c) plots the ratio of each FIFO CDF in Figure 4.6(b) with that of the corresponding priority CDF. Figure 4.6(c) shows that for $idx = 1$ or $\Delta = 30$ sec, FIFO achieves at least 90% as many pulses as priority, more than 95% of the time. Only for $idx = 55$ and $\Delta = 5$ sec (very bursty traffic and a small update interval), does FIFO perform significantly worse (achieving $\sim 80\%$ as many pulses as priority $\sim 80\%$ of the time).

Substituting the time-varying series of Ns into Equation 4.4, we obtain the corresponding series of reflectivity standard deviation σ_r values. Figure 4.6(d) plots the ratio of each priority σ_r CDF with that of the corresponding FIFO σ_r CDF, again using the data from all runs. Due to the $1/\sqrt{N}$ behavior in Equation 4.4, Figure 4.6(d) shows that the gains in N from prioritizing sensor control are diminished: e.g., now for $idx = 55$ and $\Delta = 5$ sec, priority scheduling has at least 90% as much uncertainty

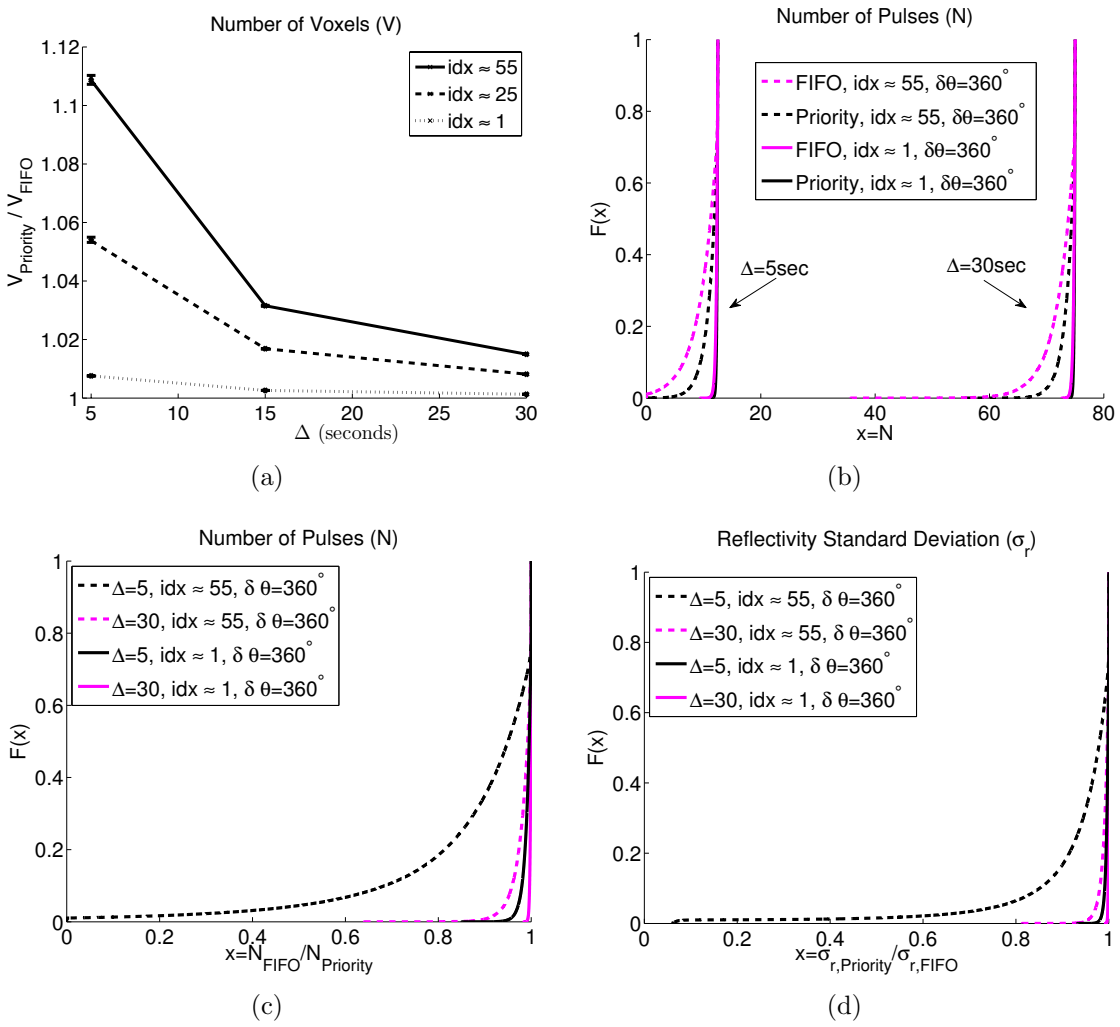


Figure 4.6. (a) Number of times more voxels V scanned under priority scheduling than under FIFO; 95% bootstrap confidence intervals over 10 simulation runs are shown. (b) CDFs of the number of pulses, N. (c) CDFs of the normalized number of pulses. (d) CDFs of the normalized reflectivity standard deviation.

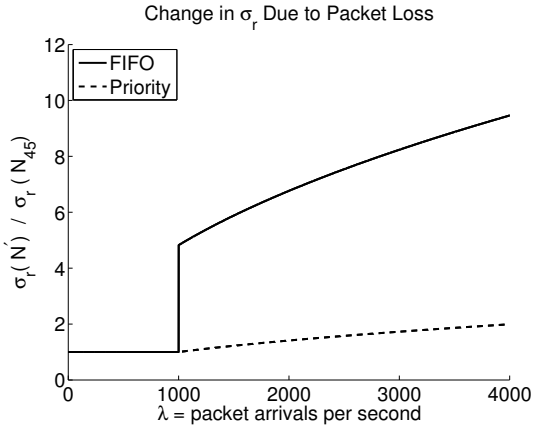


Figure 4.7. Packet loss under different arrival rates. Capacity is 1000 pkts/sec; when arrivals exceed capacity, packets are lost. We assume for these results that when capacity is exceeded, all sensor control packets are lost for FIFO scheduling, but no sensor control packets are lost for priority scheduling. During times of packet loss, however, both FIFO and priority scheduling lose data packets. Note that the data contained in the data packets are reflectivity measurements per scanned voxel, not storm cell location measurements.

as FIFO about 90% of the time. To summarize, Figures 4.5 and 4.6 show while the $\alpha_k + \beta_k$ delays for priority scheduling are about half of those for FIFO, these gains do not translate into equivalent % gains in N or σ_r, and the gains are greater for smaller Δs and more overloaded links.

Effect of packet drops on reflectivity standard deviation. From our simulation setup for dropped packets, FIFO will have $N' = N_{360} \times (1 - p_{loss})$ data samples per scanned voxel reach the control center while priority will have $N' = N_{45} \times (1 - p_{loss})$ data samples per scanned voxel reach the control center. From [20] we use $N_{360} = 750$ and $N_{45} = N_{360} \times 3 \times 8 = 18000$. Figure 4.7 plots the reflectivity standard deviation when N' samples reach the control center (i.e., there is loss), normalized by the reflectivity standard deviation when N_{45} samples reach the control center (i.e., there is no loss). We assume that the network delivers packets at its capacity and that traffic beyond network capacity is lost. Hence, $N' = N_{45}$ for both FIFO and priority when arrivals are less than capacity; when arrivals exceed capacity, $N' = N_{360} \times (1 -$

p_{loss}) for FIFO and $N' = N_{45} \times (1 - p_{loss})$ for priority. Figure 4.7 shows that as the system goes into overload, σ_r degrades gracefully for priority scheduling, as the offered load increases (i.e., the fraction of lost data samples increases). These results show that by prioritizing sensor control, the sensing system is robust to network overload conditions, and suggest that in times of congestion, it is preferable for the end-to-end data transfer protocol to ignore lost data samples, rather than adopting an ARQ protocol for retransmission, that would then increase the data delays to RTT timescales.

4.5.3 Storm-Tracking Results

Effect of packet delays on tracking error. We plug the delays generated from ns-2 into the tracking application in Section 4.4.3. At each control update we compute the next scan sector which becomes the new sensor control. We note that the data packets here that are transmitted to the control center still contain reflectivity samples, not tracking location information; tracking information is extracted from this reflectivity data at the control center.

Figure 4.8 shows the RMSE under FIFO relative to that of priority. The RMSE is computed over the differences between the true, $\mathbf{x}_k$, and estimated true, $\hat{\mathbf{x}}_k$, locations of the storm centroid. As Δ decreases and burstiness increases, Figure 4.8 shows that the benefits of prioritizing increase: for $\Delta = 5$ sec and $idx = 55$, FIFO has a median of about 1.06 times the RMSE of priority scheduling. We also see some outliers: e.g., for $\Delta = 5$ sec and $idx = 55$, when FIFO has about 4.17 times the RMSE of priority. For this outlier run, the average scan angle was about 56° while for the other 9 runs, the average scan angle ranged from 36° to 46°. Hence, once the scan angle (and consequently the measurement noise) is sufficiently large, the Kalman filter less effectively filters out the noise when estimating the true location. As it is not possible by prioritizing sensor control to gain even 4x more data (let alone 4x better

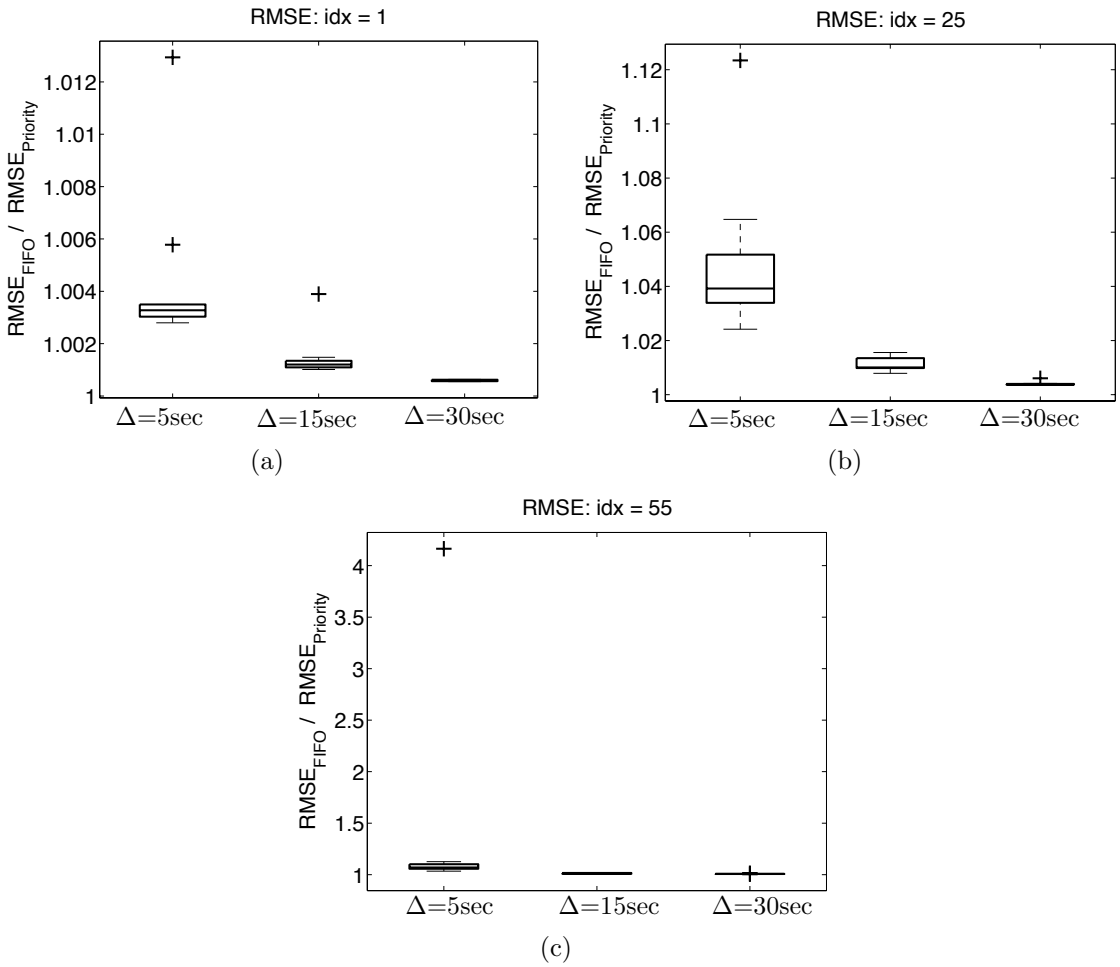


Figure 4.8. RMSE from tracking application for (a) $idx = 1$, (b) $idx = 25$, and (c) $idx = 55$. Boxplots are over 10 runs. Boxes show the median and first and third quartiles; +'s indicate outliers, i.e., data values more than 1.5 times greater (smaller) than the third (first) quartile.

reflectivity standard deviation) within a single update interval, then at least some of the outliers are due to errors accumulating over multiple intervals. Thus, for tracking, it is possible for per-interval performance gains or losses to accumulate across multiple update intervals, unlike with the voxel and reflectivity standard deviation metrics.

4.6 Summary

In this chapter, we examined the value of prioritizing sensor control traffic over data traffic during times of network congestion in closed-loop sensor networks. Ground-

ing our analysis in a meteorological radar network, we showed that prioritizing sensor control traffic decreases the round-trip control-loop delay, and thus increases the quantity and quality of the collected data and improves application performance.

One interesting direction for future work is to reduce the amount of sensor data that must be transmitted over the network. This could be done by summarizing or compressing the data, or by changing the sensing strategy so that less data is actually collected. For instance, it may not make sense to collect data if there is insufficient bandwidth to transmit the data to a control center. We also assumed that when sensor control packets are dropped, that the default sensing strategy was to scan 360°. Thus, another direction for future work is to instead assume that radars have some intelligence and are not solely reliant on the control center for sensor controls.

CHAPTER 5

ROBUST ROUTING IN AD HOC NETWORKS

5.1 Introduction

In Chapter 4, we examined how limited network bandwidth impacts the performance of the sensing application. In this chapter, we consider the problem of routing in networks with bandwidth constraints due to changing network conditions. We focus on wireless and mobile ad-hoc networks (MANETs), which are distinguished by time-varying link characteristics and network topology. In such dynamic environments, the network must accommodate the changes, providing end-end packet delivery while at the same time incurring low control overhead. Yet this ideal is difficult to meet in practice: end-end delivery requires some form of end-end (potentially global) coordination, and frequent changes make adaptation to each and every change costly. Link and mobility characteristics may also be difficult to estimate *a priori*, making proactive or predictive routing approaches difficult to implement in practice.

We specifically consider the problem of *robust* routing in MANETs. By "robust" we mean that although a particular routing configuration may not be optimal for a single *specific* configuration (e.g., specific network topology and link characteristics), it will perform well over a larger set of likely network configurations: i.e., it is robust to changes without requiring global recomputation. The issues of local versus global adaptation to link/topology changes, and the timescale(s) at which this adaptation occurs (and the concomitant overhead incurred) are central to the MANET routing problem. The approach to MANET routing explored in this chapter is based on the intuition that a global routing configuration should be determined at a coarse time-

scale (e.g., periodically, every T time units), with local adaptation to link or topology changes occurring at a finer time-scale within the current global configuration.

We examine here an approach towards MANET routing, which we refer to as "braid routing," that is robust to changes in link characteristics and network topology.[1] Informally, braid routing operates at two timescales. At the longer time-scale, a routing subgraph (i.e., a braid, defined formally in Section 5.3.2) is constructed that connects a source and destination. At the shorter time-scale, local forwarding decisions are made to select the "best" next hop out of all possible next hops. Unlike many existing "backup routing" approaches that pre-compute disjoint paths, e.g., [43], or partially disjoint paths, e.g., [25], a braid does not impose such requirements on the subgraph. Like approaches such as [25], braid routing performs local adaptation in response to link and topology changes. But unlike approaches that route packets over the entire network topology to achieve robustness (e.g., [92]), the braid subgraph over which packets are forwarded is purposefully constrained to limit control overhead (e.g., for braid construction and state maintenance). The tradeoff between the control overhead incurred (which depends on the width of the braid and the interval at which the braid is re-computed), and packet delivery performance is of principal concern to us.

We analyze braid routing from several different perspectives in order to fully explore and understand its properties. We analytically characterize the reliability (the probability that the source and destination nodes have an instantaneous path, see [12]) of a class of braids, their optimality properties, and counter-examples to conjectured optimality properties in a well-structured (grid) network. Through simulation, we compare the reliability of braid, disjoint-path, and full-network routing in both torus and random networks, and show that while braids incur significantly less overhead,

[1] We note that the term braided routing originates with [25]. The braid routing we propose in this chapter differs from that of [25] in the structure and usage of the braid (i.e., the routing subgraph).

they can also achieve almost the same reliability as using the full-network. Finally, we investigate the performance of braid routing versus other MANET routing protocols. Considering the percentage of packets delivered, we show that braid routing can deliver more packets than Ad-hoc On-Demand Distance Vector (AODV) [74] routing without significantly increasing overhead. Considering control overhead, and comparing with dynamic source routing [36], we show that braid routing can significantly decrease control overhead while only minimally degrading the number of packets delivered, with gains dependent on node density. In addition to quantifying the gains and overheads of braid routing, our simulations also illustrate how performance results can change rather dramatically depending on the underlying network model.

The remainder of this chapter is structured as follows. In Section 5.2, we discuss related work on routing. In Section 5.3 we describe the reliability metric we use as a measure of robustness, formally define what we mean by a braid, and describe how we use a braid for routing. We then present analytical results, in Section 5.4, and simulation results, in Section 5.5, evaluating braid performance in terms of reliability. In Section 5.6, we present simulation results evaluating the utility of the braid for routing. Finally, in Section 5.7, we summarize our results.

5.2 Related Work

A variety of other work has considered the use of disjoint routes in ad hoc networks, including [52, 61, 70, 78]. In addition to the overhead cost of finding disjoint paths, if any link in a path breaks then the path itself breaks. Detection and recovery from failures is also expensive since it cannot be carried out locally. These considerations have thus motivated research on the use of non-disjoint paths.

Considering non-disjoint paths, backup routing [51] reinforces the path selected by AODV [74] by allowing nodes that overhear AODV control messages to become part

of the routing subgraph, to be used only when links on the AODV path break. [82] proposes duct routing in mobile packet radio networks, allowing nodes neighbouring the primary path to be used. When sending packets to the ith node along the primary path, one of either the ith node or one of its neighbours will hear the transmission first. The first node that hears the transmission will forward the packet to the $(i+1)$st node; the other nodes will overhear the forwarding transmission and refrain from transmitting. For underwater networks, [68] proposes a geo-routing mesh using only nodes within a given distance from the vector from the source or current forwarding node to the sink. Finally, braided multipaths are proposed in [25] to protect against node failure. A braided multipath consists of the primary path plus an additional path for each node i on the primary path that does not use node i, possibly reusing parts of the primary path. We note that [82] (when all nodes neighbouring the primary path are used) and [51, 68] build routing subgraphs which structurally correspond to what we will describe in Section 5.4 as a 1-hop braid. Our work generalizes that of [51, 68, 82] since we consider k-hop braids, with $k \geq 1$. Our work also differs from [51, 68, 82] in how we use a braid, since we focus on leveraging the braid structure to decrease control overhead by updating routes less frequently. Our work differs from [25] in the construction and structure of the routing subgraph.

For changing network topology, [94] show for a class of graphs that it is possible to maintain paths whose lengths are within a constant factor of the shortest path while limiting overhead. Focusing on reliability, [65] argues for the reliability benefits of using non-disjoint paths in wireless mesh networks, showing gains over disjoint paths. Also focusing on reliability, [26] considers the problem of finding the most reliable subgraph for routing. Due to the #P-hardness of this problem, they propose a method to approximately compute reliability and leverage known contact probabilities between node pairs to select a routing subgraph. Finally, [86] proposes a routing algorithm that first selects the most reliable path and then locally reinforces those

links whose probability of being up is lower than a threshold. Unlike [26, 86], our work focuses on identifying properties of a routing subgraph which make it reliable, and then efficiently identifying such a subgraph without actually computing reliability.

5.3 What do we mean by robust?

Any protocol used in a wireless network or MANET must typically adapt to the changing network structure. If there are frequent network changes, however, then adapting to every change can be costly in terms of control overhead. Ideally, however, a protocol would not need to adapt to every change (thereby reducing control overhead), but would still perform "well enough." More generally, we say that the solution specified by a protocol is *robust* if the solution performs well over many scenarios. In the rest of this section, we first define what we mean by robust routing and describe a type of routing sub-graph that we propose, which we call a braid, to provide such robustness. We then describe how we use a braid for routing.

5.3.1 Robustness in Terms of Reliability

Informally, we say that a routing subgraph is *robust* if there is at least one path available between a source and destination with high probability, even as links appear or disappear in the network. Consider the four-node network in Figure 5.1: as time passes, indicated by the T values, links may fail or re-appear. Figure 5.1 shows that if the shortest path is chosen as the routing subgraph, there is a path from the source to the destination 50% of the time, while the two shortest disjoint paths have a path 75% of the time, and the entire network has a path 100% of the time. Thus, routing over the entire network here ensures that a path is available even as links fail, without needing to re-compute the subgraph. Generally, however, we would prefer not to route over the entire network as this incurs high control overhead. Instead, we would like

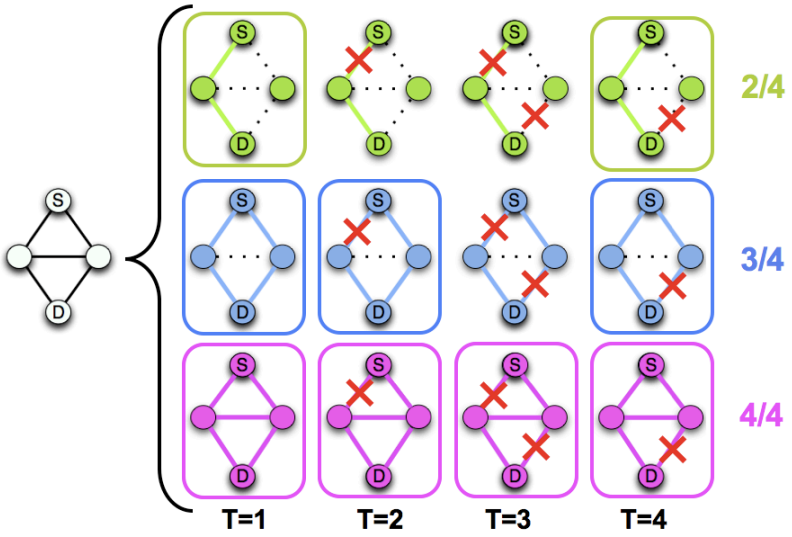


Figure 5.1. Example network with source S and destination D. Top row: selecting the shortest path as the routing sub-graph. Middle row: selecting the two shortest disjoint paths as the routing subgraph. Bottom row: selecting the entire network as the routing subgraph. Crosses indicate link failures.

to select some small number of additional nodes to include in the routing sub-graph to provide robustness to changes in network topology.

We formalize our definition of routing sub-graph robustness using reliability theory. Consider a graph $G = (V, E)$ with IID edges up with probability p, and specify source and destination nodes. From reliability theory [12], the 2-terminal reliability $R(G, p)$ is the probability that there exists an instantaneous path between the source and destination in G,

$$R(G,p) \quad = \quad \sum_{i=0}^{|E|} N_i p^i (1-p)^{|E|-i} \tag{5.1}$$

where N_i is the number of pathsets with i edges, and $p^i(1-p)^{m-i}$ is the probability that a pathset with i edges is up. A pathset is defined as a subset of the edges in G for which there is a path between the specified source and destination nodes. Thus, $p^i(1-p)^{|E|-i}$ is the probability that a subgraph with i edges, and containing a path from the source to the destination, is up. Taking the product of $p^i(1-p)^{|E|-i}$ and the

number of pathsets with i edges, N_i, gives the reliability contribution from pathsets with i edges. Summing over all $1 \leq i \leq |E|$ considers all possible pathset sizes. Thus, Equation 5.1 computes 2-terminal reliability by summing over the probabilities of the different way a path could exist between the source and the destination. We will use 2-terminal reliability to evaluate the robustness of different routing subgraphs and provide intuition about what types of graphs are "highly" reliable.

5.3.2 The k-Hop Braid

For a given source and destination, the most robust routing subgraph is the subgraph that has maximum 2-terminal reliability while using at most a specified amount of overhead. Computing reliability exactly, however, is generally $\#P$-complete [12], as is solving the corresponding optimization problem [26]. For all-terminal reliability (the probability that a graph is connected), [40] gives a randomized fully polynomial time approximation scheme. [40] shows that for very reliable graphs only small cuts are likely to fail and there are only a polynomial number of such cuts; otherwise Monte Carlo simulation may be used to compute reliability. The approach in [40] could presumably be used to approximate 2-terminal reliability, but this does not efficiently solve the optimization problem, nor lend itself easily to theoretical comparisons of the reliability of different subgraphs. Thus, in this work, we focus on identifying structural properties of graphs that make them reliable, and efficiently finding subgraphs with such properties.

Consider Equation (5.1): in the small p limit, reliability is dominated by terms from shorter paths, and so the shortest path (i.e., most reliable path) is an appropriate part of the routing subgraph. Conversely, [12, 76] give an alternative expression for reliability as a polynomial in $q = 1 - p$ with source, s, and destination, d, as follows,

$$R(G, p) \quad = \quad 1 - \sum_{C_i \in \mathcal{C}} P(E_i) \tag{5.2}$$

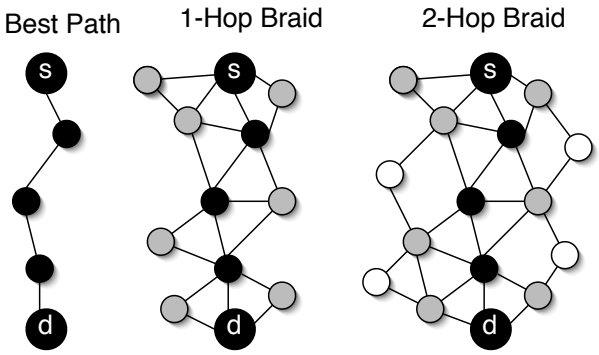


Figure 5.2. Example best path, 1-hop braid, and 2-hop braid between a source (s) and destination (d).

$$P(E_i) \;=\; q^{|C_i|}\left[1 - \sum_{C_j \in L(C_i)} \frac{P(E_j)}{q^{|C_i \cap C_j|}}\right]$$

where C_i is the set of edges in minimal cut i (partitioning s and d), $\mathcal{C}$ is the set of all C_i, and, informally, $L(C_i)$ is the set of minimal cuts lying entirely between node s and edges in cut C_i. In the small q limit, the unreliability $1 - R(G, p)$ is dominated by the smallest cuts. Thus, in this limit, a good routing subgraph will have a large minimum cut: i.e., the subgraph should widen uniformly along the shortest path. We use this analysis to propose a type of routing sub-graph that we call a "braid", shown in Figure 5.2: a k-hop braid comprises the "best" path (e.g., most reliable or shortest path) between a source and destination, plus all nodes within k hops of nodes on the best path. If there is a tie among several paths for the best path, there are several options. One option is to select the best path randomly from among the tied paths. Another option is to select the path around which the best braid (i.e., the k-braid with highest reliability for some k) can be built; we leave this for future work.

5.3.3 Braid Routing Algorithm

In this section, we describe how to use our proposed braid sub-graph to perform routing. Our braid routing algorithm contains the following steps:

Global adaptation every T time-steps:

1. Identify shortest path in network
2. Build k-hop braid around shortest path
3. Perform local forwarding within braid

Rather than re-build the braid routing sub-graph every time a link breaks, we only re-construct the braid every T time-steps. If a link failure occurs, a braid can potentially adapt to the change by locally routing packets around the failure, with the scope of the forwarding constrained by the braid. Consequently, network-wide flooding of control messages to alert nodes about the link failure and to repair the route need not be (immediately) incurred. We will focus on a simple, single-copy local forwarding mechanisms in Sections 5.6.1 and 5.6.2, but other ways to implement local forwarding include flooding and backpressure routing [92]. While the braid sub-graph changes every T timesteps, local forwarding decisions are computed by nodes every timestep. *The key idea is that by using a braid for routing, routes can be re-computed less frequently, and so less control overhead will be incurred.*

5.4 Theoretical Analysis

In this section, we characterize the reliability properties of braids, concentrating on well-structured grid networks. Our goal is to analyze how well a braid performs with respect to 2-terminal reliability given a fixed number of nodes or links in the subgraph. These results then provide insight into more general network topologies, which are analytically intractable.

5.4.1 1-Hop Braids

A 1-hop braid comprises the shortest path between a source and a destination plus all nodes within 1-hop of nodes on the shortest path. For the idealised network model

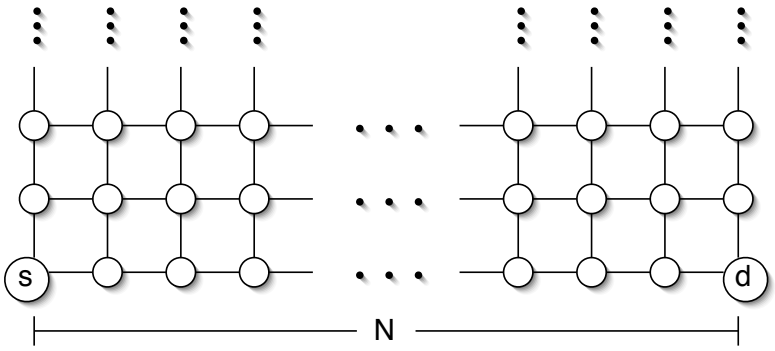


Figure 5.3. Model used in Section 5.4, comprising source (s) and destination (d) on a line in a bounded half-plane grid.

shown in Figure 5.3, we show in Lemma 1 that a reliable routing subgraph should widen uniformly along the shortest path, as in a 1-hop braid. Informally, Lemma 1 says that when incrementally adding nodes (one or two at a time), adding all nodes one hop away from the shortest path before adding any nodes that are two hops away maximizes reliability.

Lemma 1: *Assume the network structure in Figure 5.3: the source, s, and destination, d, are connected by a shortest path, P, comprising N nodes; links are IID and up with probability* $0 < p < 1$. *Let G be the sub-graph formed by P plus* $0 < n < N$ *additional 1-hop nodes, (where a k-hop node is a node k hops away from P). Using one additional 1-hop node (and its associated edges) that is also adjacent to another 1-hop node, increases the reliability of G strictly more than does using any two additional 2-hop nodes (and their associated edges).*

Proof: Figure 5.4(a) shows the general structure of the graphs we consider. Suppose we can add either one of the grey nodes or the black node. Adding only one of the grey nodes does not affect the reliability from s to d, as no additional (disjoint or non-disjoint) paths will be created from s to d. Adding the black node increases the probability of getting from nodes d_0 and d_1 to node d, and so will increase the probability of getting from node s to node d.

$$
\begin{aligned}
P(d|s) &= P(d|s_0s_1)P(s_0s_1|s) + P(d|s_0\bar{s}_1)P(s_0\bar{s}_1|s) + P(d|\bar{s}_0s_1)P(\bar{s}_0s_1|s) \\
\\
P(d|s_0s_1) &= P(d|d_0d_1)P(d_0d_1|s_0s_1) + P(d|d_0\bar{d}_1)P(d_0\bar{d}_1|s_0s_1) + P(d|\bar{d}_0d_1)P(\bar{d}_0d_1|s_0s_1) \\
P(d|s_0\bar{s}_1) &= P(d|d_0d_1)P(d_0d_1|s_0\bar{s}_1) + P(d|d_0\bar{d}_1)P(d_0\bar{d}_1|s_0\bar{s}_1) + P(d|\bar{d}_0d_1)P(\bar{d}_0d_1|s_0\bar{s}_1) \\
P(d|\bar{s}_0s_1) &= P(d|d_0d_1)P(d_0d_1|\bar{s}_0s_1) + P(d|d_0\bar{d}_1)P(d_0\bar{d}_1|\bar{s}_0s_1) + P(d|\bar{d}_0d_1)P(\bar{d}_0d_1|\bar{s}_0s_1) \\
\\
P(d_0d_1|s_0s_1) &= P(d_0d_1|q_0q_1)P(q_0q_1|s_0s_1) + P(d_0d_1|q_0\bar{q}_1)P(q_0\bar{q}_1|s_0s_1) + P(d_0d_1|\bar{q}_0q_1)P(\bar{q}_0q_1|s_0s_1) \\
P(d_0\bar{d}_1|s_0s_1) &= P(d_0\bar{d}_1|q_0q_1)P(q_0q_1|s_0s_1) + P(d_0\bar{d}_1|q_0\bar{q}_1)P(q_0\bar{q}_1|s_0s_1) + P(d_0\bar{d}_1|\bar{q}_0q_1)P(\bar{q}_0q_1|s_0s_1) \\
P(\bar{d}_0d_1|s_0s_1) &= P(\bar{d}_0d_1|q_0q_1)P(q_0q_1|s_0s_1) + P(\bar{d}_0d_1|q_0\bar{q}_1)P(q_0\bar{q}_1|s_0s_1) + P(\bar{d}_0d_1|\bar{q}_0q_1)P(\bar{q}_0q_1|s_0s_1) \\
P(d_0d_1|s_0\bar{s}_1) &= P(d_0d_1|q_0q_1)P(q_0q_1|s_0\bar{s}_1) + P(d_0d_1|q_0\bar{q}_1)P(q_0\bar{q}_1|s_0\bar{s}_1) + P(d_0d_1|\bar{q}_0q_1)P(\bar{q}_0q_1|s_0\bar{s}_1) \\
P(d_0\bar{d}_1|s_0\bar{s}_1) &= P(d_0\bar{d}_1|q_0q_1)P(q_0q_1|s_0\bar{s}_1) + P(d_0\bar{d}_1|q_0\bar{q}_1)P(q_0\bar{q}_1|s_0\bar{s}_1) + P(d_0\bar{d}_1|\bar{q}_0q_1)P(\bar{q}_0q_1|s_0\bar{s}_1) \\
P(\bar{d}_0d_1|s_0\bar{s}_1) &= P(\bar{d}_0d_1|q_0q_1)P(q_0q_1|s_0\bar{s}_1) + P(\bar{d}_0d_1|q_0\bar{q}_1)P(q_0\bar{q}_1|s_0\bar{s}_1) + P(\bar{d}_0d_1|\bar{q}_0q_1)P(\bar{q}_0q_1|s_0\bar{s}_1) \\
P(d_0d_1|\bar{s}_0s_1) &= P(d_0d_1|q_0q_1)P(q_0q_1|\bar{s}_0s_1) + P(d_0d_1|q_0\bar{q}_1)P(q_0\bar{q}_1|\bar{s}_0s_1) + P(d_0d_1|\bar{q}_0q_1)P(\bar{q}_0q_1|\bar{s}_0s_1) \\
P(d_0\bar{d}_1|\bar{s}_0s_1) &= P(d_0\bar{d}_1|q_0q_1)P(q_0q_1|\bar{s}_0s_1) + P(d_0\bar{d}_1|q_0\bar{q}_1)P(q_0\bar{q}_1|\bar{s}_0s_1) + P(d_0\bar{d}_1|\bar{q}_0q_1)P(\bar{q}_0q_1|\bar{s}_0s_1) \\
P(\bar{d}_0d_1|\bar{s}_0s_1) &= P(\bar{d}_0d_1|q_0q_1)P(q_0q_1|\bar{s}_0s_1) + P(\bar{d}_0d_1|q_0\bar{q}_1)P(q_0\bar{q}_1|\bar{s}_0s_1) + P(\bar{d}_0d_1|\bar{q}_0q_1)P(\bar{q}_0q_1|\bar{s}_0s_1)
\end{aligned}
$$

Table 5.1. Reliability computation for Figure 5.4.

	Using Grey Nodes	Using Black Node
$P(d\|d_0d_1)$	p	$\geq p + p^3 - p^4$
$P(d\|d_0\bar{d}_1)$	p	$p + p^3 - p^4$
$P(d\|\bar{d}_0d_1)$	p^2	$2p^2 - p^4$
$P(q_0q_1\|s_0s_1) = P(q_0\|s_0)P(q_1\|s_1)$	$p(p + p^3 - p^4)$	$p \cdot p$
$P(q_0\bar{q}_1\|s_0s_1) = P(q_0\|s_0)P(\bar{q}_1\|s_1)$	$p(1 - p - p^3 + p^4)$	$p(1-p)$
$P(\bar{q}_0q_1\|s_0s_1) = P(\bar{q}_0\|s_0)P(q_1\|s_1)$	$(1-p)(p + p^3 - p^4)$	$(1-p)p$
$P(q_0q_1\|s_0\bar{s}_1) = P(q_0\|s_0)P(q_1\|\bar{s}_1)$	0	0
$P(q_0\bar{q}_1\|s_0\bar{s}_1) = P(q_0\|s_0)P(\bar{q}_1\|\bar{s}_1)$	p	p
$P(\bar{q}_0q_1\|s_0\bar{s}_1) = P(\bar{q}_0\|s_0)P(q_1\|\bar{s}_1)$	0	0
$P(q_0q_1\|\bar{s}_0s_1) = P(q_0\|\bar{s}_0)P(q_1\|s_1)$	0	0
$P(q_0\bar{q}_1\|\bar{s}_0s_1) = P(q_0\|\bar{s}_0)P(\bar{q}_1\|s_1)$	0	0
$P(\bar{q}_0q_1\|\bar{s}_0s_1) = P(\bar{q}_0\|\bar{s}_0)P(q_1\|s_1)$	$p + p^3 - p^4$	p

Table 5.2. Reliability computations for Figure 5.4. The product of each of the first 3 rows times each of the last 9 rows gives the 27 terms (ignoring scaling factors) of the full reliability computation. For each of the 27 products, the "using black node" product is $\geq$ the "using grey nodes" product.

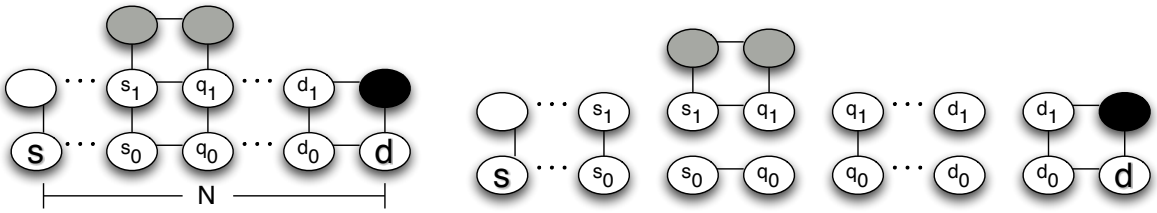
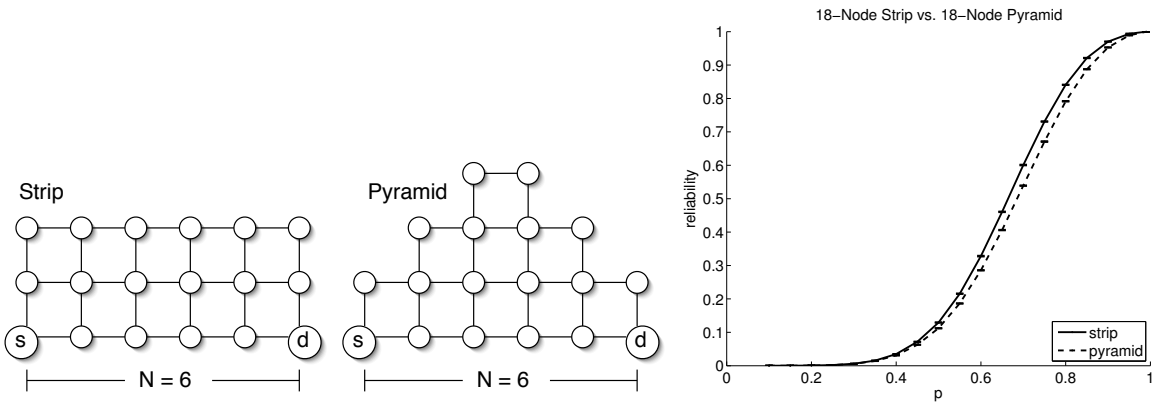


Figure 5.4. Graphs used in Lemma 1. We decompose the graph in (a) into the subgraphs in (b) so we need only compute the reliability for the subgraphs of interest.

Now consider adding two nodes at a time. Again consider the topologies in Figure 5.4. We partition the graph in Figure 5.4(a) into the sub-graphs shown in Figure 5.4(b); each edge appears only once, although nodes may be repeated (which will not affect the reliability). Using sub-graph decompositions we decompose the reliability by conditioning on the intermediate nodes as follows. We first condition on intermediate nodes s_0 and s_1 to obtain,

$$P(d|s) \quad = \quad P(d|s_0s_1)P(s_0s_1|s) + P(d|s_0\bar{s}_1)P(s_0\bar{s}_1|s) + P(d|\bar{s}_0s_1)P(\bar{s}_0s_1|s) \qquad (5.3)$$

where e.g., $P(d|s_0\bar{s}_1)$ is the probability that node d can be reached given that node s_0 but not s_1 can be reached, and $P(s_0\bar{s}_1|s)$ is the probability that node s_0 but not s_1 can be reached given that node s can be reached. We recursively condition on nodes $\{d_0, d_1\}$ and $\{q_0, q_1\}$ to obtain an equation for $P(d|s)$ as the sum of 27 terms, shown in Table 5.1. We use the resulting equation to compute both the reliability when adding both of the grey nodes in Figure 5.4 and when adding the black node. Ignoring those terms that correspond to subgraphs that are identical for both we need only compute the reliability for the $\{s_0, s_1\} \rightarrow \{q_0, q_1\}$ and $\{d_0, d_1\} \rightarrow d$ subgraphs. These calculations are shown in Table 5.2. Examining Table 5.2 shows that for each $P(\{q_0, q_1\}|\{s_0, s_1\})P(d|\{d_0, d_1\})$ product, adding the black node to the end of the $2 \times N$ node strip gives the same or higher reliability as compared with adding the two grey nodes anywhere on top of the strip. $\diamondsuit$



(a) (b)

Figure 5.5. (a) Two topologies, both using 18 nodes. (b) Reliability is averaged over 100 runs of 10,000 time-steps each. 95% bootstrap confidence intervals over the runs are shown.

While we used the half-plane network model shown in Figure 5.3 (in which a 1-hop braid corresponds to a $2 \times N$ node strip) to prove Lemma 1, we could also generalize Lemma 1 to a full 1-hop braid by using the full-plane and a similar proof method as that used to prove Lemma 1. Doing so would result in, however, an equation for $P(d|s)$ even more complex than the equation shown in Table 4.1.

Lemma 1 is more general than stated as it does not depend on the form of the $\{s_0, s_1\} \rightarrow \{q_0, q_1\}$ and $\{d_0, d_1\} \rightarrow d$ subgraphs. These results suggest that k-hop braids have several desirable reliability properties, at least in this well-structured environment and with uniform p, giving confidence in studying k-hop braids in scenarios where the optimum subgraph cannot be determined. In Figure 5.3, the $k \times N$ node strip from s to d is a $(k-1)$-hop braid. We close with two conjectures:

Conjecture 1: *Given N additional nodes (and their associated edges) plus the shortest path, the $2 \times N$ node strip is the most reliable subgraph.* It follows from Lemma 1 that for $N \leq 5$ additional nodes, reliability is maximized by the $2 \times N$ node strip (to see this, consider the ways 5 extra nodes could be arranged).

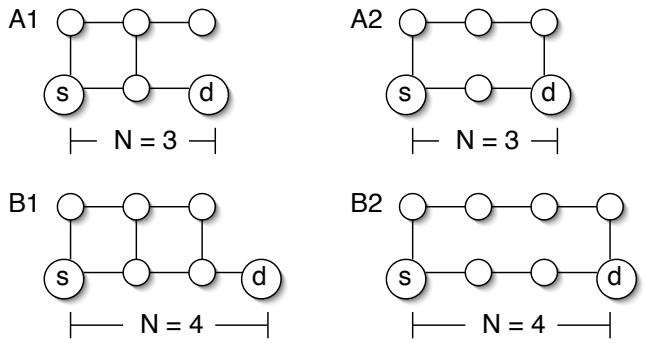


Figure 5.6. Counterexamples when adding links rather than nodes.

Conjecture 2: *Given $2N$ additional nodes (and their associated edges) plus the shortest path, the $3 \times N$ node strip is more reliable than the corresponding pyramid.* Comparing the $3 \times N$ node strip for $N = 6$ versus the pyramid that can be built using 18 nodes, see Figure 5.5 (a), we find experimentally[2] that the strip has higher reliability than the pyramid, as shown in Figure 5.5(b).

5.4.2 Comparison With Disjoint Path Routing

A degenerate case of a 1-hop braid, where all internal links are missing, is a pair of disjoint paths which use neighbouring nodes. Does the optimal braid with a constraint on the number of links contain holes of this type? The answer depends on the measure of overhead and the value of p. Consider the examples in Figure 5.6 of a partial braid and a pair of disjoint paths. Graphs A_1 and A_2 both use six links total. Suppose we compute the reliabilities $R(A_1)$ and $R(A_2)$ of graphs A_1 and A_2. We find (by considering the different ways that a path can exist),

[2]Given the difficulty of exactly computing reliability, except for relatively simple networks, we also use Monte Carlo simulation to estimate reliability. In a discrete-time simulation of a time-varying network, we can check whether there is a path from the source to the destination at each time-step. The ratio of the number of time-steps when there is a path and the total number of time-steps simulated is then an estimate of the probability that there exists an instantaneous path from source to destination. We refer to computing the reliability in this way as "computing the reliability experimentally."

$$
\begin{aligned}
R(A_1) &= p^2(1-p^3) + p^4(1-p) + p^5 \\
&= p^2 + p^4 - p^5 \qquad (5.4) \\
R(A_2) &= p^2(1-p^4) + p^4(1-p^2) + p^6 \\
&= p^2 + p^4 - p^6 \qquad (5.5)
\end{aligned}
$$

Since $R(A_1)$ is always less than $R(A_2)$, graph A_2 is more reliable than graph A_1 for all values of p. Similarly, suppose we compare the reliabilities of graphs B_1 and B_2, which both use eight links total. We compute,

$$
\begin{aligned}
R(B_1) &= \left[p(p(1-p^2) + p^2(1-p) + p^3)^2 + (1-p)(p^2(1-p^4) + p^4(1-p^2) + p^6)\right] p \\
&= \left[p(p + p^2 - p^3)^2 + (1-p)(p^2 + p^4 - p^6)\right] p \\
&= p^3 + 3p^5 - 2p^6 - 3p^7 + 2p^8 \qquad (5.6) \\
R(B_2) &= p^3(1-p^5) + p^5(1-p^3) + p^8 \\
&= p^3 + p^5 - p^8 \qquad (5.7)
\end{aligned}
$$

We can then compute when $R(B_1)$ is greater than $R(B_2)$,

$$
\begin{aligned}
R(B_1) &\geq R(B_2) \\
p^3 + 3p^5 - 2p^6 - 3p^7 + 2p^8 &\geq p^3 + p^5 - p^8 \\
2p^5 - 2p^6 &\geq 3p^7 - 3p^8 \\
2(1-p) &\geq 3p^2(1-p) \\
\sqrt{\frac{2}{3}} &\geq p \qquad (5.8)
\end{aligned}
$$

Thus, graph B_2 is only more reliable than graph B_1 when $p > \sqrt{2/3}$, i.e., the two-disjoint paths are only more reliable than the partial braid for large values of p (i.e., for more reliable links). Work in [58] extends our results here and show that the

regime in which the braid is more reliable becomes larger for larger networks. This assumes that "links used" is the right overhead metric; an alternative metric is "nodes used," for which the appropriate comparison is between the disjoint paths and the 1-hop braid, and the latter is always more reliable.

5.5 Reliability Simulations

In this section, we use simulation to compare the reliability properties of braids with those of other types of routing subgraphs. This lets us examine the reliability properties of braids in network structures more general than the bounded half-plane grid model considered in Section 5.4. We first describe our network model and then present results.

5.5.1 Network Model

Consider a graph $G = (V, E)$ with nodes V and edges E. We examine (i) a $\sqrt{|V|} \times \sqrt{|V|}$ torus where $|E|$ comprises the set of all edges in the torus and (ii) a random model, where $|V|$ nodes are placed uniformly randomly and independently in the plane, and edges exist between those nodes within a communication radius L of each other. We consider both link and node failures. To model the link (node) failures, we assume links (nodes) are IID; to model link (node) changes, we use a two-state Markov model where links (nodes) stay up with probability p and stay down with probability q at each time-step.[3] When a node fails, all of its links also fail.

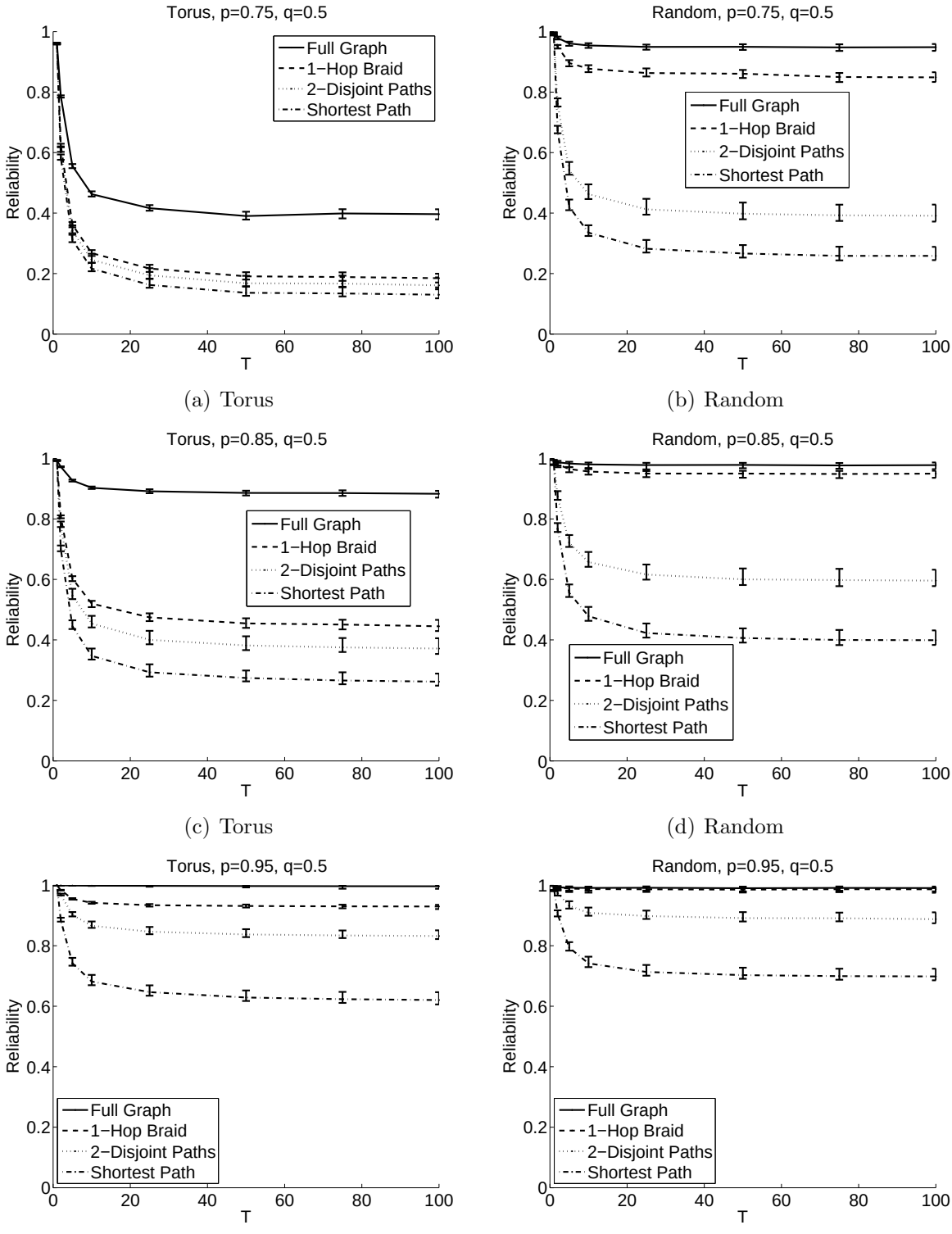
In our experiments, we use (i) a 10×10 torus and (ii) 100 nodes distributed randomly in an area of size 10×10 using a communication radius $L = 2$. We perform 500 simulation runs in the case of link failures, and 1000 runs in the case of node failures. Each run comprises 100 timesteps. At the start of each run, a random

[3]Note that this link (node) failure model differs from, and is more general than, the link failure model used in Section 5.4, where we assumed that links were IID and up with probability p.

source-destination pair is selected; if there is no path between the selected source and destination, another random source-destination pair is selected for the run. At each time-step, we check which links (nodes) are up. For the two-state Markov model we use $p = \{0.75, 0.85, 0.95\}$ and $q = 0.5$. We use the steady-state probability that a link (node) is up to select which links (nodes) are initially up. The routing sub-graph for each algorithm is recomputed every T timesteps, using only links (nodes) that are up in the graph at the time of re-computation. All algorithms were evaluated on identical network topologies, and we estimate the sub-graph reliability experimentally.

5.5.2 Link Failure Results

In this section, we consider link failures. Figure 5.7 shows that for all p, that as the size of the route update interval T increases, the reliability of the selected routing subgraph decreases and eventually reaches steady-state. For the torus, Figure 5.7(a) shows that for $p = 0.75$, the reliability of the 1-hop braid, 2-shortest-disjoint paths, and shortest path are all within a range of 0.1. Increasing p to 0.85 in Figure 5.7(c) shows a larger gap in reliability between the 1-hop braid and the 2-shortest disjoint paths, and also a larger gap between the braid and the full graph. Using $p = 0.95$ in Figure 5.7(e) shows an even larger gap in reliability between the 1-hop braid and the 2-shortest disjoint paths, but now a much smaller gap between the braid and the full graph. For the random model, Figures 5.7(b), (d), and (f) again show that for all p examined, the 1-hop braid has consistently higher reliability than the shortest path or 2-shortest disjoint paths, now as much as 0.4 greater than the 2-shortest disjoint paths when $p = 0.75$ or $p = 0.85$. This is in part a consequence of there not always being 2 disjoint paths in the graph (unlike in the torus). When $p = 0.95$, in Figure 5.7(f), the reliability achieved by the 1-hop braid is almost identical to that achieved by the full graph.

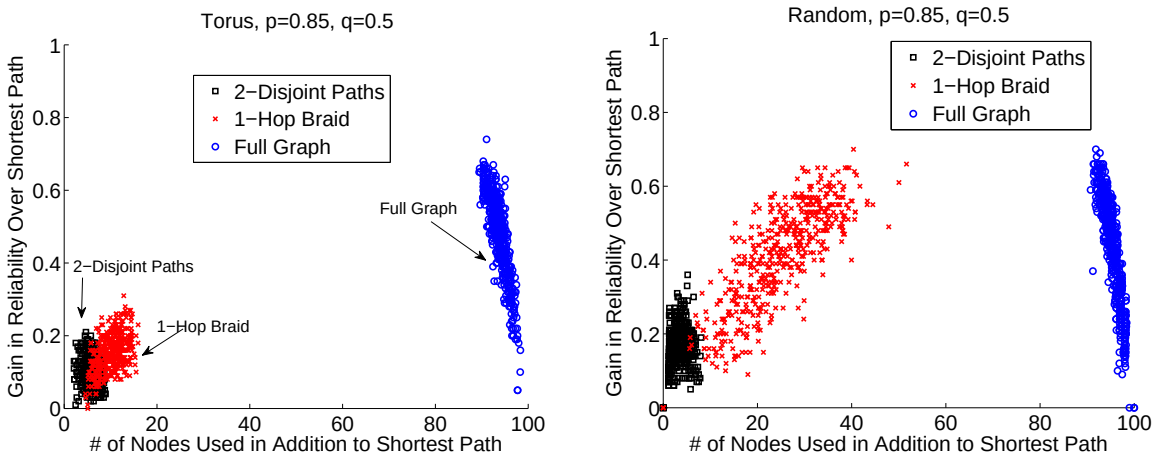


(a) Torus (b) Random

(c) Torus (d) Random

(e) Torus (f) Random

Figure 5.7. Link failure simulations. Reliability of different routing subgraphs. Reliability is averaged over 500 runs of 100 time-steps. 95% confidence intervals over the runs are shown. As not all sets of samples were normally distributed, bootstrap confidence intervals were computed using Matlab (hence the error bars are not symmetric).



(a) Torus, T=5, Link Failures

(b) Random, T=5, Link Failures

Figure 5.8. Overhead of 1-hop braid vs. that of the shortest path, the two shortest disjoint paths, and the entire graph. Reliability was estimated experimentally.

Figure 5.8 plots the reliability gain given the number of additional nodes used over the shortest path by the 2-shortest disjoint paths, 1-hop braid, and full graph. Each point represents a simulation run (i.e., a selected source destination pair); for clarity we show only results for when $T = 5$. For the torus, Figure 5.8(a) indicates that the 1-hop braid provides an increase in reliability while using fewer than 20 extra nodes. For the random model, Figure 5.8(b) indicates that while the braid provides consistent and significant (up to about 0.4) gains in reliability, it also uses around 40 more nodes than the shortest path, but fewer than half the nodes used by the full graph.

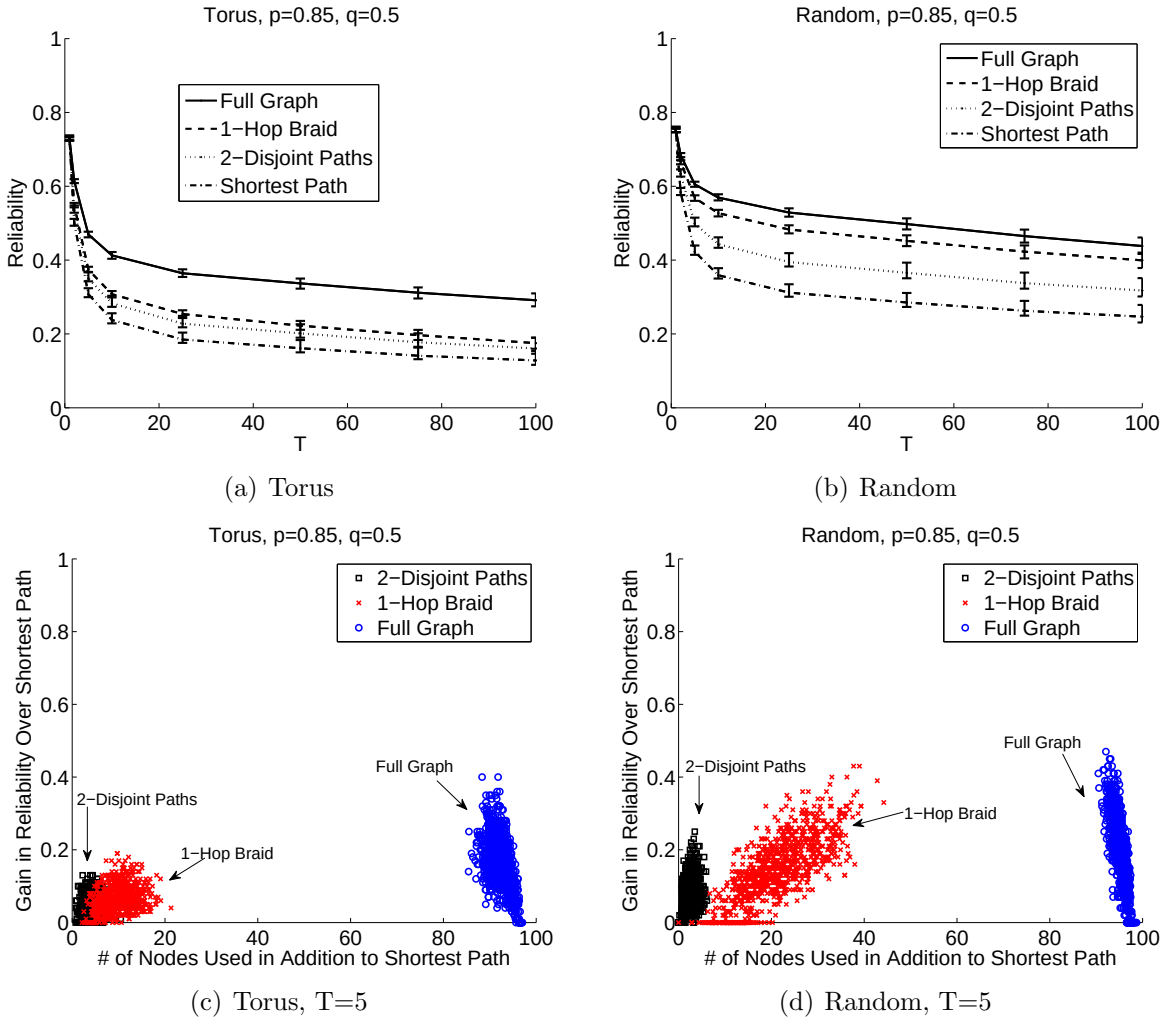
To summarize, the torus results indicate that the 1-hop braid can achieve reliability greater than that of the shortest path and the 2-shortest disjoint paths, and that the gains increase as p increases. We expect, however, that using a 2-hop braid would increase the reliability gain of the braid for small p. The results from the random model indicate that while using more nodes, the 1-hop braid can achieve reliability close to that of the full graph, and that the gain increases as p decreases.

5.5.3 Node Failure Results

In this section, we consider node failures. Figures 5.9(a) and (b) show that for $p = 0.85$, node failures (or perfectly correlated link failures) significantly impact the reliability gains that can can be achieved over the shortest path by using additional disjoint or non-disjoint paths. This is also reflected in the overhead plots in Figures 5.9(c) and (d). Compared with the link failure results in Figure 5.8, the node failure results in Figures 5.9(c) and (d) indicate that while the 2-disjoint paths or 1-hop braid do not use significantly fewer additional nodes than in the case for link failures, significantly lower reliability gains are achieved from the additional nodes used. For example, in Figure 5.8(b), the 1-hop braid can achieve reliability gains of almost 0.7 with link failures, but with node failures, in Figure 5.9(d), the 1-hop braid achieves reliability gains of at most about 0.4. In practice, we do not expect link failures to be either perfectly correlated or independent, and so would expect some reliability gain from using additional disjoint or non-disjoint paths.

5.6 Routing Simulations

In this section, we investigate the amount of control overhead incurred by braid routing in a mobile scenario via simulation. We also investigate the trade-off between control overhead and the percentage of packets delivered and packet delay. The analysis in Section 5.4 and the reliability simulations in Section 5.5 gave us insight into the trade-off between connectivity and overhead (in terms of nodes or links used). In comparison, the routing simulations in this section let us (i) examine more general network scenarios, specifically mobile networks where link failures are typically neither perfectly independent nor perfectly correlated as with node failures, (ii) investigate braid overhead in terms of routing control packets incurred, not just nodes or links used, and (iii) investigate routing performance in terms of the percentage of packets delivered and delay, not just braid connectivity. In the rest of this section, we first



(a) Torus

(b) Random

(c) Torus, T=5

(d) Random, T=5

Figure 5.9. Node failure simulations. (a), (b) Reliability of different routing subgraphs. Reliability is averaged over 1000 runs of 100 time-steps. 95% confidence intervals over the runs are shown. As not all sets of samples were normally distributed, bootstrap confidence intervals were computed using Matlab (hence the error bars are not symmetric). (c), (d) Overhead of 1-hop braid vs. that of the shortest path, the two shortest disjoint paths, and the entire graph. Reliability was estimated experimentally.

present routing simulation results for when the routing update interval T, described in Section 5.3.3, is constrained to be a constant value. We then relax this constraint in Section 5.6.2.

5.6.1 Constant T

When T is constant, we constrain braid routing as well as any other routing algorithm with which we compare to all re-compute their routes at the same fixed update interval T. We expect when T is constant that braid routing would incur about the same amount of control overhead as, for instance, shortest path routing, since both would re-compute routes at the same rate. We would also expect braid routing to have a higher delivery packet ratio, since braid routing would be more likely to have a path than shortest path routing during the interval T during which routes are not recomputed. In the rest of this section, we first describe our naive implementation of braid routing and then present simulation results in a MANET environment.

5.6.1.1 Naive Braid Routing Implementation

In our naive implementation of braid routing we use AODV [74] to construct the shortest path around which the braid is built. We construct a 1-hop braid around the AODV shortest path as follows. When a node receives data to forward along the AODV path, it broadcasts a braid request for the associated destination (if one has not yet been sent). When a node receives a braid request for a destination, it groups the request with other requests for that destination. If it can hear at least two nodes on the path, it sends a braid reply to all nodes it can hear (except to the node closest to the destination).

To tear the braid down, a braid node sends error messages to nodes it can hear on the AODV path when either one of its links to the AODV path breaks (i.e., drops a packet) and T has elapsed, or when it receives a more recent braid request for

the destination (indicating that the current AODV path has been replaced). A node deletes its next hop braid for a destination when either (i) its next hop or later link on its AODV best path has dropped a packet for that destination, or (ii) a node for that destination is updated in its AODV routing table. A node marks a link as "bad" whenever the node attempts to use a link and has a packet dropped. The AODV path and/or braid will be recomputed only when T has elapsed. Whenever routes are recomputed, links are marked as "good."

Nodes perform local forwarding within the braid as follows. Nodes on the AODV path select their AODV next hop with probability 1 if it is "good" or if there is no next hop braid node, and with probability 0.1 if it is "bad." If the AODV next hop was not selected, then the node iterates through its braid links. A braid link is selected with probability 1 if it is good or probability 0.1 if it is bad. If the node iterates through all of its braid links without selecting a next hop, then by default the AODV next hop is returned. If the node is a braid node, then it iterates through the nodes it can hear on the AODV path, selecting the AODV path node that is currently both closest to the destination and good. To ensure that bad links are also attempted, any AODV path node can be selected with probability 0.1. If the node iterates through all of its AODV path nodes without selecting a next hop, then by default the first AODV path node in its list is returned.

5.6.1.2 Simulation Set-up

We perform our simulations when T is constant using GloMoSim [95]. We use 60 nodes, moving according to the following mobility models. *(1) Random waypoint*: the pause time was 0 sec and node speeds were uniformly chosen between 4km/hr and 10km/hr. *(2) Gauss-Markov* [55]: average node speed was 7.2km/hr with standard deviation of 1.08km/hr and we use $\alpha = 0.2$ and $\Delta t = 100$. We use a 1.5km x 1.5km area for the random waypoint experiments and a 1km x 1km area for the Gauss-

Markov experiments. Traces of node mobility were generated using BonnMotion [34] and fed into GloMoSim, letting us evaluate braid routing and AODV on identical mobility scenarios. We use a constant bit rate flow between two nodes for which data was generated every 0.5 sec and a total of 5 million packets were generated. We performed 10 simulation runs, each for the duration of the flow (about 29 simulated days). To address the problem of a long transient phase, the length of the flow was selected by examining the packet drop rate for progressively longer flows; when the change in % of packets dropped was sufficiently small ($< 0.05\%$), we assumed that steady-state had been reached. A better method would be to e.g., implement the "perfect simulation" method of Le Boudec and Vojnovic [8]; we leave this for future work. The MAC protocol used was 802.11 and the transmission radius was about 250 meters (from setting the radio transmit power to 7.9dBM).

5.6.1.3 Results

Figure 5.10 compares AODV and braid routing with respect to throughput, overhead, and links used. Figures 5.10(a) and (b) show for both mobility models that the braid achieves a maximum of about 5% higher throughput than AODV for $T = 50, 100, 200$. Figures 5.10(c) and (d) show for both mobility models that the braid uses about the same amount of AODV overhead when building its best path as AODV (as measured by the number of path requests and replies transmitted by AODV); under Gauss-Markov mobility, however, this overhead is about 2.7×10^6 fewer packets, likely due in part to the smaller, 1km x 1km, area used. Figures 5.10(c) and (d) also show that while the braid incurs overhead from braid requests and replies, this overhead is about 1/4 of the AODV overhead under random waypoint, and about 1/2 of the AODV overhead under Gauss-Markov; the total braid overhead, however, for both mobility models is similar. Figures 5.10(c) and (d) also show that the total number of error packets transmitted for braid routing (aggregating error packets for

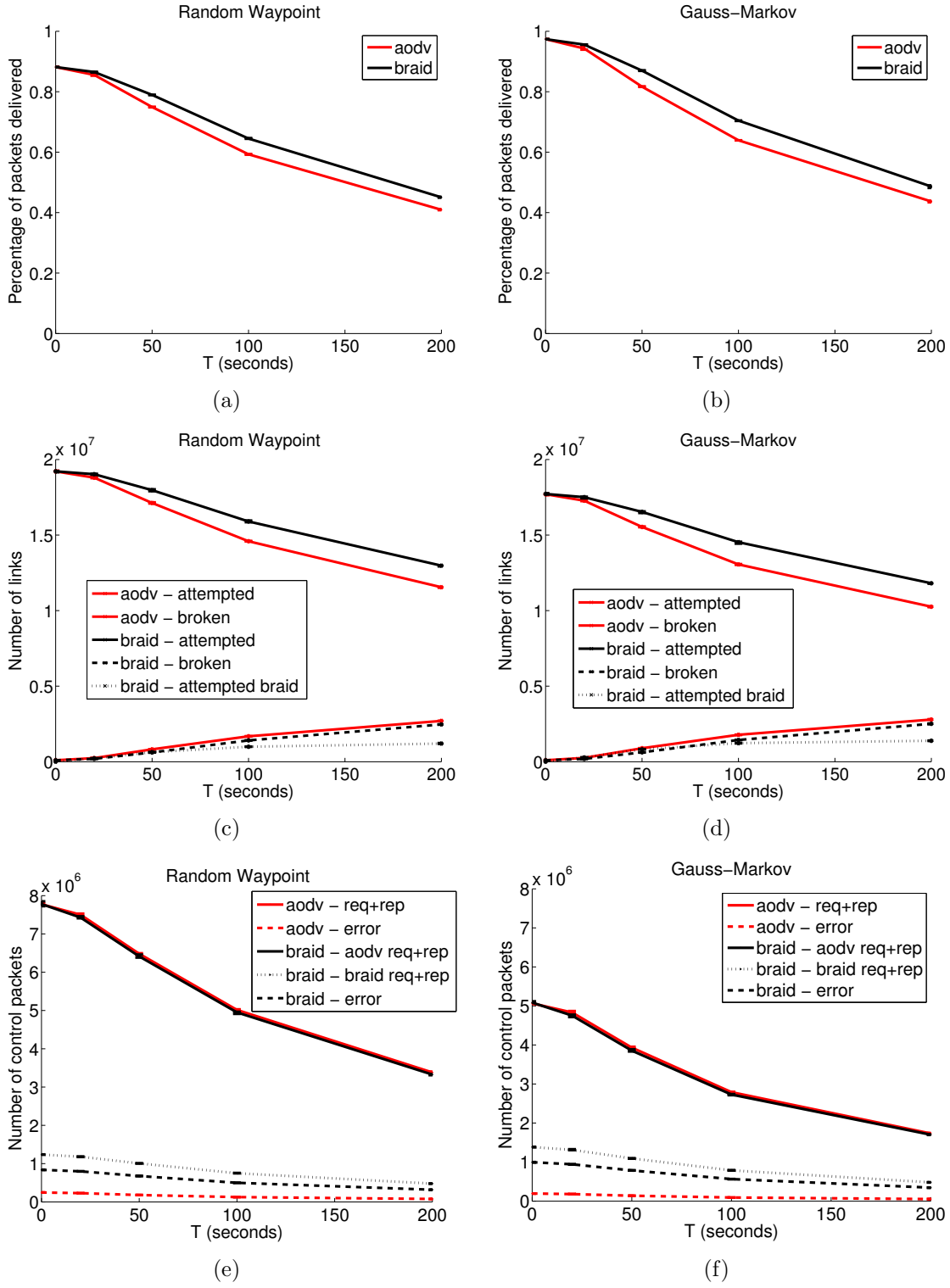


Figure 5.10. Comparison of 1-hop braid with AODV under (a), (c), (f) random waypoint and (b), (d), (f) Gauss-Markov mobility. 95% bootstrap confidence intervals over 10 simulation runs are shown.

both AODV and the braid) is perhaps five times greater than AODV error packets, in part because the braid involves more nodes in routing. As in Figure 5.10, other work, e.g., [16], has also observed that AODV can use as much (or more) control overhead as data transmitted, so we focus here on the additional overhead incurred by the braid. Since the braid construction is independent of the "best" path algorithm, another routing algorithm besides AODV could be used. Finally, Figures 5.10(e) and (f) show for both mobility models that the braid algorithm attempts to use more links than AODV (where "attempt" indicates that the routing algorithm attempted to transmit a packet over a link, but may not have been successful), in part because it may use a longer path. The braid, however, also has fewer links broken on average than does AODV.

In summary, Figure 5.10 indicates that the 1-hop braid gains about 5% more throughput while using significantly less overhead than, for instance, would be needed to construct a second disjoint AODV path. The gains in throughput, however, are not as significant as the gains in reliability shown in the Matlab experiments in Section 5.5. We conjecture that this discrepancy is in part a consequence of (1) building the braid around the shortest path rather than the most reliable path, and (2) the different network models, particularly in how they differ with respect to the rate at which links appear/disappear, and the temporal and spatial correlations among links changes. Note that since the braid construction is independent of the "best" path algorithm, a routing algorithm that identifies the most reliable path could be used rather than AODV.

Comparing the different network models, consider first the rate at which links appear/disappear. Results from [28] indicate that the inter-meeting times for two nodes using the random waypoint model are "well-approximated by an exponential distribution, at least for small to moderate transmission radii (with respect to the size of the area)." Using Lemma 1 in [28], we compute that for the transmission radius

and random waypoint model considered here, the expected inter-meeting time for two nodes is given by $1/\lambda$ with $\lambda = 2.65$/hr. Hence, on average, two nodes will meet once every 22.7 minutes. Thus, in our random waypoint GloMoSim experiments, when a link breaks (due to mobility) it likely stays down for an interval significantly longer than T. Conversely, the probability of transitioning from down to up during T was 0.5 in the models used in the Matlab experiments in Section 5.5. Long inter-meeting times limit the throughput gains achieved by the braid since when braid links fail it is unlikely that they will re-appear before the remaining time in the interval T has elapsed.

Next consider correlations among links. In the Matlab experiments we assumed links failed independently. Conversely, we would expect that outgoing links of a given node would tend to have correlated failures when links break due to mobility. We would also expect that since all link failures are varying functions of how much time t of the interval T has elapsed, that link failures among different nodes would also be dependent due to the shared dependence on t. Correlated link failures limit the throughput gains achieved by the braid since if a link on the AODV path fails, it is also more likely that one of the links routing around the failed link will also fail soon (if it has not already).

5.6.2 Variable T

In Section 5.6.1, we presented simulation results for when the routing update interval T, described in Section 5.3.3, is constrained to be a constant value. In this section, we present simulation results for when the value of T is allowed to vary in response to the failure of all paths within the braid. When T is variable, we allow braid routing as well as any other routing algorithms with which we compare to re-compute their routes at whatever frequency is necessary to ensure a route to the destination. We expect when T is variable that braid routing would incur less

control overhead than, for instance, shortest path routing, since braid routing would re-compute routes less frequently (assuming the network is sufficiently dense that a braid can be built). We would also expect braid routing to have about the same delivery packet ratio, since braid routing would be no more likely to have a path than shortest path routing, since both would be recomputing routes as frequently as necessary (for the respective algorithms) to ensure a path. In the rest of this section, we first describe how we implement braid routing efficiently and then present results for both a stationary wireless network and a MANET environment.

5.6.2.1 Efficient Braid Routing Implementation

Consider how reactive routing algorithms such as Ad-hoc On-Demand Distance Vector Routing (AODV) [74] or Dynamic Source Routing (DSR) [36] identify the shortest path between a source and destination. Such routing algorithms typically have the source send out route requests; route replies are then propagated back from the destination to the source specifying a path to the destination. Regardless of how the respective routing algorithms subsequently use this routing information, the algorithms must all incur the control overhead of these route requests and replies.

Suppose that overheard route requests and replies only contain information about nodes that are 1-hop away, such as in AODV. To construct a 1-hop braid then, like in backup routing [51], a node must incur some additional control overhead to determine which of its 1-hop neighbours to use to forward a packet when the node's next hop link on the best path breaks. Conversely, now suppose that overheard route requests and replies may contain information about nodes that are multiple hops away, such as with source routing and DSR. Then to construct a 1-hop braid, no additional control overhead is necessary. Thus, to efficiently implement braid routing, we use DSR [36] to construct the shortest path around which the braid is built. We show in Figure 5.11 a simple example of how 1-hop braid information is contained in overheard route

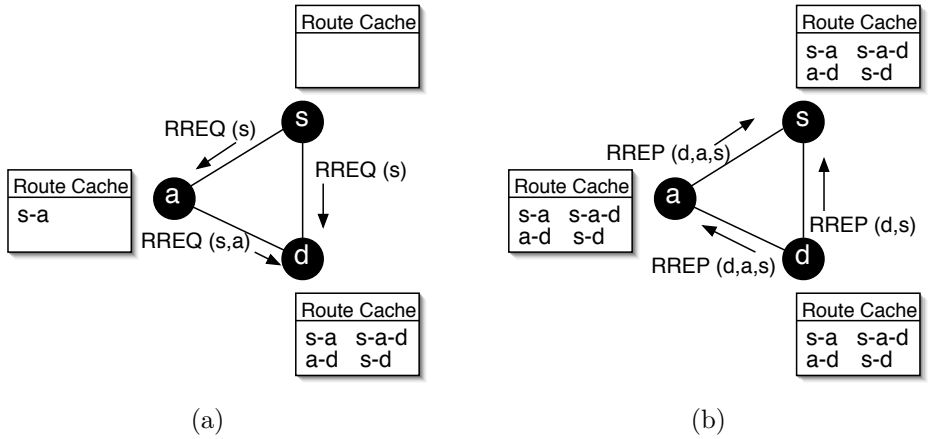


Figure 5.11. Transmission of (a) route requests (RREQ) from the source s to the destination d and (b) route replies (RREP) from the destination to the source. The source route associated with each RREQ or RREP is indicated in parentheses. The route caches contain the routes extracted from the RREQs and RREPs; the routes are assumed bi-directional. Note that because node a is within range of node d, node a will overhear d's reply, RREP(d,s), although the reply is not destined for a.

requests and replies when source routing is used. We note that to build a k-hop braid for $k > 1$, we expect some additional control overhead will be necessary, since there is no guarantee that all k-hop braid information will be contained in the route requests and replies transmitted to construct the path around which the braid is built.

To provide a bit of detail of how forwarding is actually done in a braid, suppose that a node i experiences a link break to a node j while transmitting a packet. In DSR, the following steps occur.

1. Any routes in node i's route cache that use the link from node i to j would be deleted.

2. A route error is broadcast indicating that the link from node i to j is broken.

3. Finally, node i will attempt to salvage the packet by checking for an alternate path to the destination.

In comparison, braid routing performs the following steps. ***(i)*** As in DSR, any routes in node i's route cache that use the link from node i to j would be deleted.

1. Unlike in DSR, node i would check its route cache for a braid path to any of the remaining nodes on the source route contained in the packet (not just for an alternate path to the destination).

2. If no braid path is available, braid routing defaults to the DSR procedure of broadcasting errors (and re-constructing the path, if there are further packets to send). The frequency with which this occurs determines T.

3. If a braid path is available, the packet is sent out over the first hop of the braid path. The packet will still contain the broken source route, but the packet itself will be flagged as a braid packet (by setting the time to live to a large value) so that the source route contained in the packet is not added to the route caches of nodes overhearing or receiving the packet. A node k recognizes that it is a "braid node" for a packet whenever it receives a packet destined for itself and node k is not included on the source route contained in the packet. Whenever node k recognizes that it is being used as a braid node, node k will forward the packet to the appropriate next hop on the source route contained in the packet.

Thus, unlike in DSR, in braid routing a route error is sent only if there is no path to the destination within the 1-hop braid, rather than whenever the shortest path breaks. Consequently, we expect braid routing to incur both fewer route errors than DSR, and fewer route requests and replies since once a source route has been constructed, such messages will be sent only when the source receives a route error. We do not expect braid routing to deliver more packets than DSR, however, since for both algorithms, if a path breaks, a new path will be found: either a new DSR shortest path, or an existing path in the 1-hop braid built around the broken path. *Thus, the primary difference between braid routing and DSR is in the control overhead incurred to find this new path.*

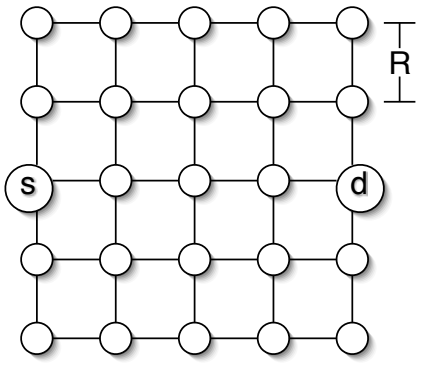


Figure 5.12. 5x5 node grid network used to obtain the simulation results shown in Figures 5.14 to 5.13. Edges represent wireless links, where R is the distance in meters between each pair of connected nodes.

5.6.2.2 Stationary Network Results

In this section we present simulation results examining the performance of braid routing in a stationary network when T is variable. Examining a stationary wireless network gives us insight into how the reliability results from Section 5.5, for grids and random graphs, translate into control overhead and percentage of packets delivered. We first describe our simulation set-up and then discuss our results.

To perform our stationary network simulations, we use the QualNet simulator, version 4.5. We consider the 5x5 node grid network shown in Figure 5.12; links are wireless and R is the distance in meters between nodes. The MAC protocol is 802.11b. The length of each simulation run is 2,500,000s. To model network traffic, we create a constant bit rate flow between the source (s) and destination (d) nodes, shown in Figure 5.12; one packet is generated every 0.25s (so 10,000,000 packets are transmitted in total). For this setup, we compare the performance of (i) DSR and (ii) the braid routing algorithm described in Section 5.6.2.1 using local forwarding over a 1-hop braid built around the shortest path found by DSR.

Figures 5.13(a) to (d) compare the control overhead for DSR and braid routing. Figure 5.13(a) shows the total control overhead incurred for each algorithm, while Figures 5.13(b) through (d) show the breakdown of the total control overhead incurred

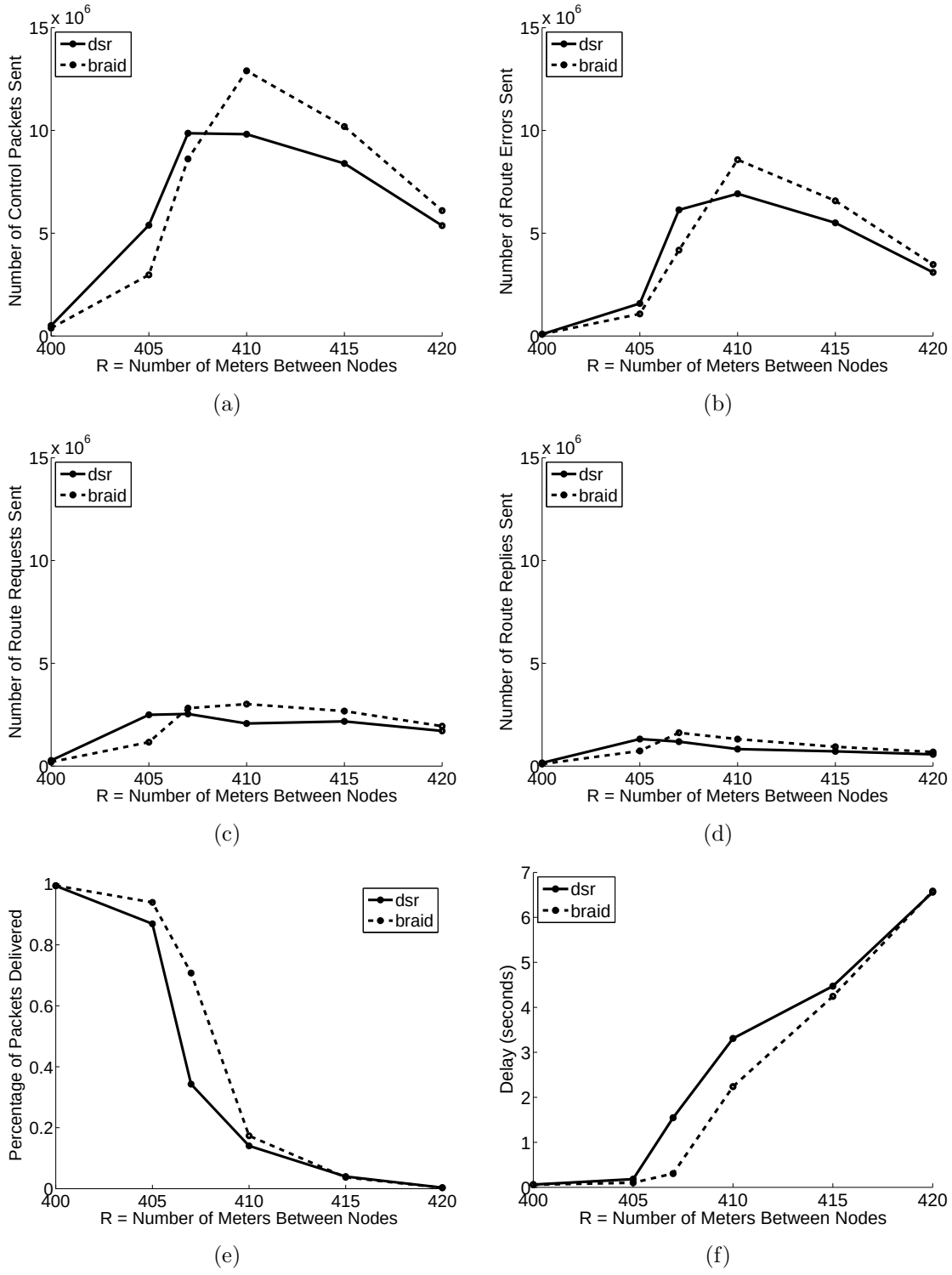


Figure 5.13. Performance of braid routing vs dynamic source routing in a stationary wireless network: (a) total control overhead, (b) route errors, (c) route requests, (d) route replies, (e) percentage of packets delivered, and (f) delay.

for each algorithm, according to route errors, route requests, and route replies. Figure 5.13(b) shows that when R is less than about 407m, that the braid routing protocol incurs fewer route errors than DSR. One reason for this is that since nodes are stationary, the possible paths from the source to the destination are not changing; instead it is *when* a possible path is available that is changing. Consequently, if a braid path is available and successfully used whenever the DSR shortest path fails, it is possible for the same DSR path to fail and then recover without incurring any route errors. Conversely, Figure 5.13(b) shows that when R is greater than about 407m, DSR incurs fewer route errors than braid routing. One reason for this is that as the network becomes more disconnected, braid routing is more likely to attempt braid paths which eventually fail: consequently route errors will need to be sent both for a link failure in the braid path that failed, and for a link failure in the DSR shortest path.

Figures 5.13(c) and (d) then show that the number of route requests and replies first increases for both DSR and braid routing as the network becomes more disconnected, and then eventually decreases (around $R = 410$m) as the network starts becoming too disconnected to be likely to contain a path from the source to the destination. Figures 5.13(c) and (d) also show for both DSR and robust routing that fewer route replies than route requests are sent (about 50% fewer), presumably due in part to DSR's mechanism for minimizing redundant route replies [37].

Next, Figure 5.13(e) compares the percentage of packets delivered for DSR and braid routing. Figure 5.13(a) shows that as nodes become more weakly connected, the percentage of packets delivered decreases for both DSR and braid routing. Braid routing is, however, better able to leverage weak connectivity: when nodes are separated by $R = 407$m, braid routing delivers about $2\times$ as many packets as DSR. When the network is too weakly connected for there to be many paths, let alone successful braid paths, the percentage of packets delivered also drops for braid routing: e.g.,

when nodes are separated by $R = 415$m, both DSR and braid routing deliver the same (small) percentage of packets.

Figure 5.13(f) then compares the average packet delay for DSR and braid routing. Figure 5.13(f) shows that as nodes become more weakly connected, the average packet delay increases. Figure 5.13(f) also shows that when there is sufficient connectivity for successful braid paths while simultaneously sufficient numbers of link breaks in the DSR shortest path so that braid paths are attempted (from about $R = 405$m to about $R = 415$m), that the braid routing algorithm has lower average delay than DSR (e.g., as much as 1s less than DSR for $R = 407$m). Since we would assume braid routing to use longer routes on average than DSR, why then does braid routing incur lower average delay? One reason for the lower average delay is that if the first hop link of the DSR shortest path breaks when transmitting a packet, with DSR, subsequent packets would be delayed waiting for a new route to be found. In comparison, with braid routing, a braid path may be available and immediately used.

Finally, Figure 5.14 compares the number of link failures for DSR and braid routing, and also shows the number of braid paths attempted by braid routing. As the distance R between pairs of nodes increases (i.e., as node density decreases), Figure 5.14 shows that the number of link failures increases for both DSR and braid routing, peaking around $R = 410$m. After $R = 410$m, the number of link failures begins to decrease, presumably due to pairs of nodes beginning to be too far apart for there to exist a link. We observe that around $R = 405$m, braid routing starts attempting noticeable numbers of braid paths ($\sim$595,000 attempts for $R = 405$m versus $\sim$42,000 attempts for $R = 400$m when examining the data). We also see that starting around $R = 407$m, braid routing starts to both have more link failures than DSR, and to attempt even more significant numbers of braid paths ($\sim$2,108,000 attempts). Since braid routing attempts both DSR shortest path links and braid links, braid routing will incur more link failures than DSR when the probability of link

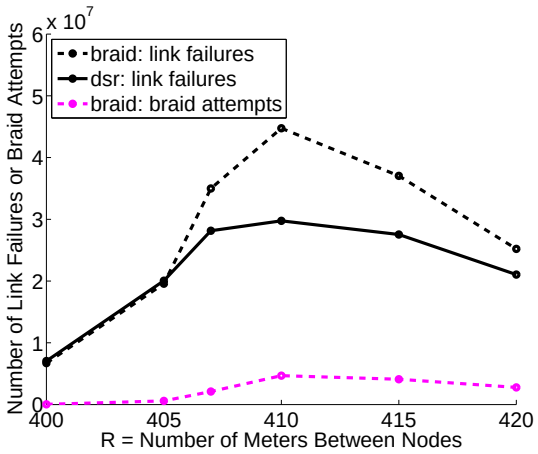
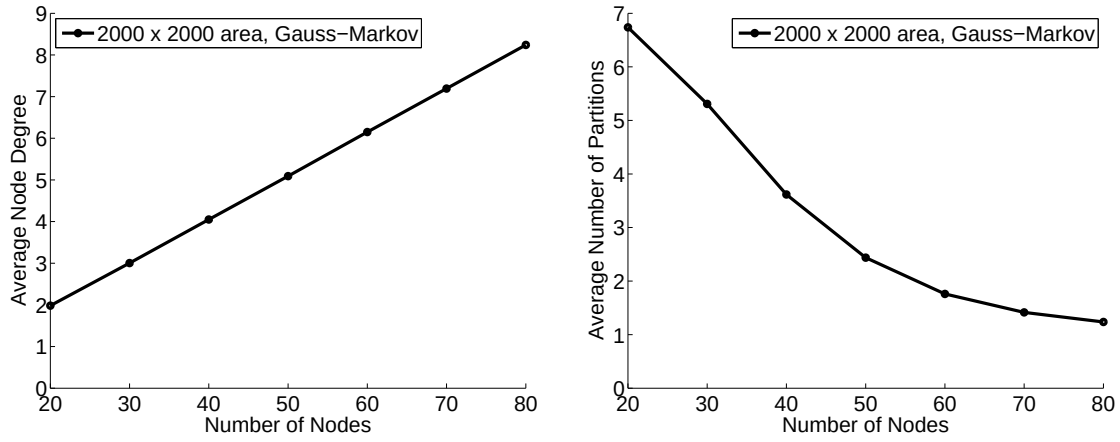


Figure 5.14. Stationary wireless network: link failures and braid attempts.

failure is sufficiently high (i.e., when R is sufficiently large). Observe that the shape of the total control overhead plots in Figure 5.13(a) and the plots for route errors in Figure 5.13(b) are similar in shape to the link breakage plots in Figure 5.14. Thus, link breakages significantly determine how much control overhead is incurred by an algorithm.

To summarize, Figures 5.13 and 5.14 show that as the stationary wireless network becomes less dense, braid routing starts to outperform DSR, incurring lower control overhead, higher throughput, and lower delays. More generally, we can characterize the regimes in which braid routing achieves its greatest performance gains. When the network is sufficiently dense (i.e., for $R \leq 400$m in Figures 5.14 and 5.13), braid routing essentially defaults to DSR. As the network becomes less dense and the probability of link breakage increases (i.e., from about $R = 400$ to about $R = 410$), braid routing starts to attempt alternative paths through the braid. As the network is still sufficiently dense for these alternative paths to succeed for some period of time, braid routing is able to outperform DSR. As the network becomes even less dense (i.e., from about $R = 410$ to about $R = 420$) and the alternative paths become likely to fail quickly, however, braid routing starts to incur more control overhead than DSR without gains in throughput. Finally, when the network is too sparse for alternative



(a) (b)

Figure 5.15. Statistics computed from the Gauss-Markov mobility traces obtained using BonnMotion [34]. (a) Average node degree. (b) Average number of partitions in the network.

paths to exist even if a DSR shortest path exists (i.e., for $R > 420$), braid routing again defaults to DSR.

5.6.3 MANET Results

In this section we present simulation results examining the performance of braid routing in a MANET when T is variable. These MANET routing simulations let us consider mobile networks where link failures are typically neither perfectly independent nor perfectly correlated as with node failures. We first describe our simulation set-up and then discuss our results.

To perform our MANET simulations, we use the QualNet simulator, version 4.5. We use N nodes moving in a 2000m×2000m area according to the Gauss-Markov mobility model [55]. Traces of node mobility were generated using BonnMotion [34] and fed into QualNet: we set the angle standard deviation to 1 radian, the maximum node speed to 2 m/s, the minimum node speed to 0.5 m/s, the speed standard deviation to 0.2 m/s, and the speed and angle update frequency to 100s. We also

have nodes bounce at boundaries. Figure 5.15 shows the average node degree and the average number of partitions computed from the Gauss-Markov mobility traces.

In our simulations, we vary the number of nodes N while holding the area fixed: we expect that increasing the node density (and correspondingly increasing the average node degree and decreasing the average number of partitions, see Figure 5.15) should increase the number of alternative braid paths available when a link fails. Our simulation runs are of 1,000,000s, where an additional initial 200,000s was removed for the transient phase. 95% bootstrap confidence intervals (over 10 runs) are shown. To model network traffic, we create a constant bit rate flow between two nodes; one packet is generated every 0.25s (thus, 4,000,000 packets are transmitted in total). The MAC protocol is again 802.11b and the connectivity range is as in Figures 5.14 to 5.13, where good connectivity is achieved when nodes are within 400m of each other.

In our simulations, we vary the number of nodes N while holding the area fixed: increasing the node density (and thus increasing the average node degree and decreasing the average number of partitions, see Figure 5.15) should increase the number of alternative braid paths available when a link fails.

Figure 5.16 compares the control overhead for DSR and braid routing. Figure 5.16(a) shows the total control overhead incurred for each algorithm, while Figures 5.16(b) through (d) show the breakdown of the total control overhead incurred for each algorithm according to route errors, route requests, and route replies. Figure 5.16(a) then shows for $N \leq 40$ that DSR and braid routing incur about the same amount of control overhead. As node density further increases, however, Figure 5.16(a) shows that DSR incurs increasingly more control overhead than braid routing, with DSR incurring about 25% more control overhead for $N = 80$. Examining Figure 5.16(b) to (d) indicates where the control overhead savings of braid routing occurs: Figures 5.16(b) to (d) indicate that braid routing reduces not just route errors, but also route requests and replies. Figures 5.16(c) and (d) then show for both

algorithms that as node density increases, route requests decrease while route replies increase.

Next, Figure 5.16(e) compares the delivery ratios for DSR and braid routing. Figure 5.16(e) shows that for both algorithms, as the node density increases, the percentage of packets delivered increases. For higher node densities, braid routing delivers slightly fewer packets than DSR: this is due in part to braid routing's use of longer braid paths which also eventually fail. We believe this gap could be removed by more efficiently implementing when and how braid routes are used, particularly more carefully deciding when source routes extracted from overheard packets are added to the route cache (since packets traversing a braid route still contain the original broken route). We note that the goal of braid routing was to maintain network connectivity while decreasing control overhead. Consequently, we did not incorporate any sophisticated packet recovery mechanisms, although doing so would potentially increase the packet delivery ratio of braid routing. Additionally, we consider only one flow in these simulations. We hypothesize that when there are multiple flows and the network is sufficiently loaded, decreasing the control overhead incurred by routing will allow more network capacity to be used to deliver packets, and ultimately let braid routing deliver more packets than DSR. We leave exploring the case when there are multiple flows in the network for future work.

Figure 5.16(f) then compares the average packet delay for DSR and braid routing. Figure 5.16(f) shows that for both algorithms, as node density increases, the average delay decreases. The average delay, however, decreases significantly more for DSR than it does for braid routing: again we believe this is due to braid routing's use of longer braid paths.

Finally, Figure 5.17 compares the number of link failures for DSR and braid routing, and also shows the number of braid paths attempted by braid routing. As the node density increases, Figure 5.17 shows that the number of link failures increases

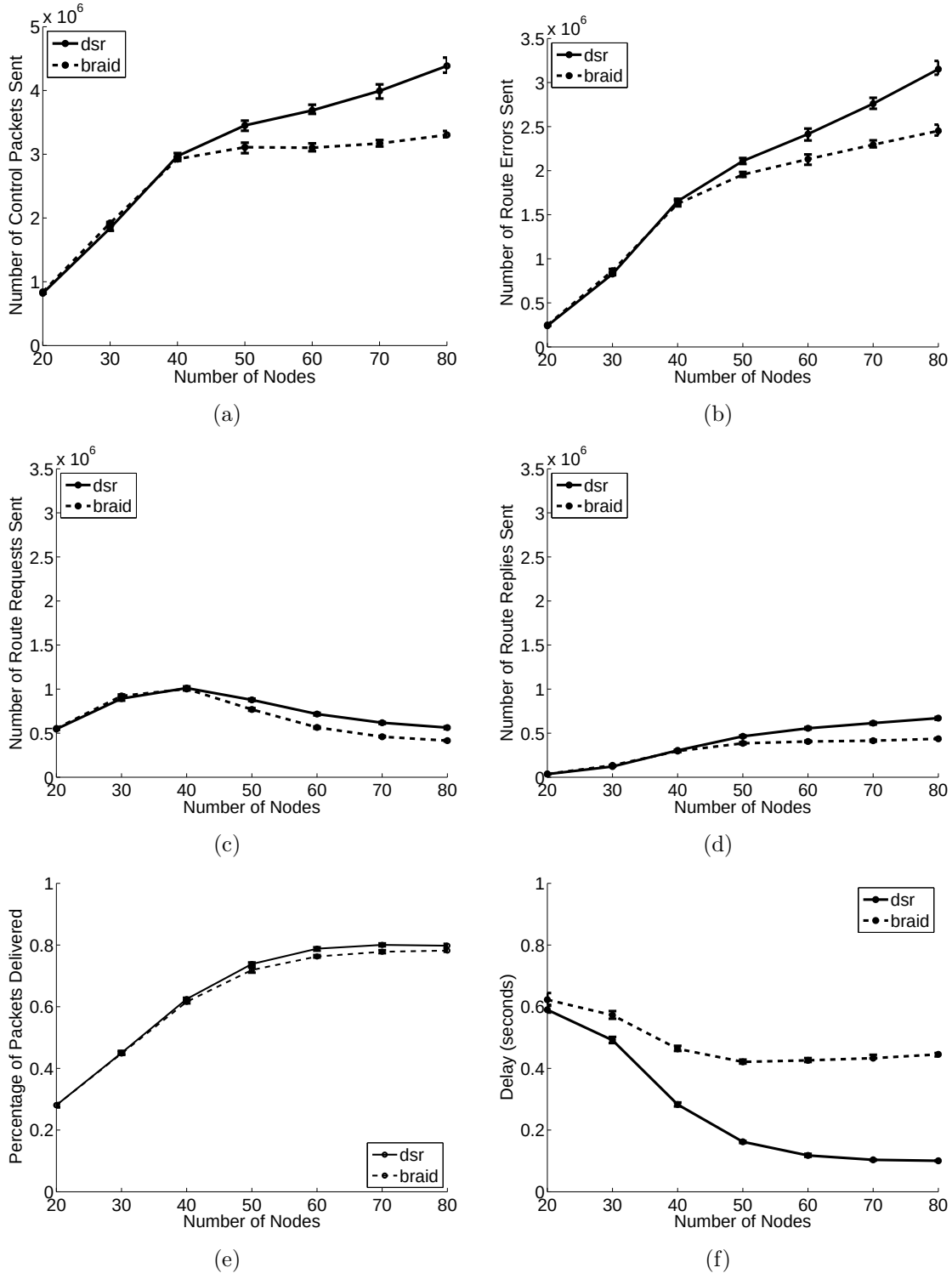


Figure 5.16. Performance of braid routing vs dynamic source routing under Gauss-Markov mobility: (a) total control overhead, (b) route errors, (c) route requests, (d) route replies, (e) percentage of packets delivered, and (f) delay. 95% confidence intervals are shown, computed over 10 simulation runs.

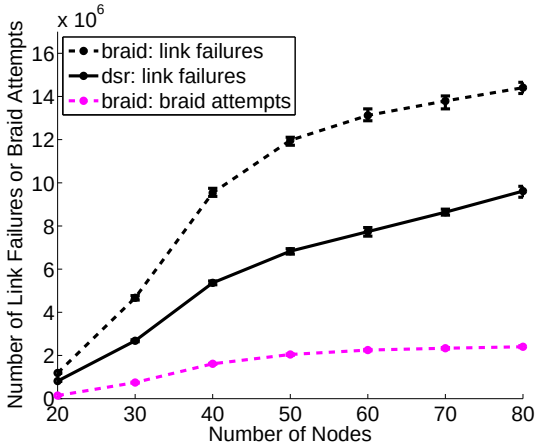


Figure 5.17. MANET: link failures and braid attempts.

for both algorithms and that braid routing also attempts more braid paths. Figure 5.17 also shows that braid routing has more link failures: we believe that this is due to braid routing attempting braid paths which eventually fail, in addition to attempting links on the DSR shortest path which also fail. Observe again that the shape of the total control overhead plots in Figure 5.16(a) and the plots for route errors in Figure 5.16(b) are similar in shape to the link breakage plots in Figure 5.17. This is because link breakages significantly determine the amount of control overhead incurred by an algorithm.

To summarize, Figures 5.16 and 5.17 show that as node density increases, braid routing starts to outperform DSR, incurring significantly lower control overhead while slightly degrading the number of packets delivered and increasing packet delays. We note that at least 40 nodes in the 2000×2000 area seems to be the minimum node density needed to obtain useful braid paths; from Figure 5.15(a), this corresponds to an average node degree of at least four nodes.

5.6.4 Discussion

Additional control overhead beyond that needed to initially construct a route arises for several reasons. First, as would be expected, whenever a route breaks,

control overhead is incurred to tear-down the old route and to construct a new route. Specifically, some packets find that their route is broken and must force route re-construction. Consequently, these packets are responsible for more control overhead than other packets. In the worst case, every packet finds its route broken. Rather than attempting to re-construct a route while connectivity is still poor, however, it may be better to delay route re-construction until network connectivity is expected to improve (and control overhead can be amortized over multiple packets).

Second, some control overhead is due to (soft) routing state whose expiration time is improperly set. Suppose it were possible to predict when a route would break. Then nodes could stop using the route before it breaks, thereby reducing the route errors incurred. The risk of such predictions, however, is that routes may be prematurely torn down, resulting in additional route requests and replies being sent.

5.7 Summary

In this chapter, we described a braid routing algorithm to improve routing robustness in wireless sensor networks and MANETs. We proposed a routing algorithm that selects a type of routing subgraph (a braid) that is robust to changes in the network topology. We analytically characterized the reliability of a class of braids and their optimality properties, and gave counter-examples to other conjectured optimality properties in a well-structured (grid) network. Comparing with dynamic source routing, we showed that braid routing can significantly decrease control overhead while only minimally degrading the number of packets delivered, with gains dependent on node density.

For future work, rather than using a fixed braid width, we are interested in techniques to locally widen the braid to meet a robustness target. While we focused on a single network flow in this work, we would also like to consider multiple flows, and to explore rate control mechanisms such as backpressure routing [92] for local forwarding

to achieve a solution that is robust in throughput as well as connectivity. It would also be interesting to incorporate link changes observed at the link layer both into the braid structure and into how rate control is performed within the braid. This could be done, for instance, by locally re-constructing the braid when link failures are observed instead of just performing local forwarding. Finally, we would generally like to investigate how network characteristics impact the control overhead needed for routing.

CHAPTER 6

CONCLUSIONS

6.1 Thesis Summary

This thesis examined sensor control and scheduling strategies to most effectively use the limited resources of an ad hoc network or closed-loop sensor network.

We first examined the question of where to focus sensing in a sensor network containing sensors, such as cameras and radars, that cannot simultaneously collect high fidelity data from all environmental locations. In Chapter 2, we specifically showed that the main benefits of considering expected future states of the environment in a sensing strategy for such sensors are when there are multiple meteorological phenomena in the environment, and when the maximum radius of any phenomenon is sufficiently smaller than the radius of the radar's footprint. We also showed that there is a trade-off between the average quality with which a phenomenon is scanned and the number of decision epochs before which a phenomenon is rescanned. Considering only scan quality, we found that a simple lookahead sensing strategy was sufficient.

Next, in Chapter 3, we considered the problem of call admission control (i.e., deciding which sensing requests to satisfy) in the context of a virtualized private sensor network. We showed that the call admission control problem in virtual private sensor networks can be solved in polynomial time when sensor requests are divisible or fixed in time. When sensor requests are indivisible but may be shifted in time, we showed that the call admission control problem in virtual private sensor networks is NP-complete.

In Chapter 4, we then considered the problem of transmitting both sensor control and data packets in the presence of network congestion. We investigated the value of separate handling of sensor control and data traffic, during times of congestion, in a closed-loop sensor network. Grounding our analysis in a meteorological radar network, we showed that prioritizing sensor control traffic decreases the round-trip control-loop delay, and thus increases the quantity and quality of the collected data and improves application performance.

Finally, in Chapter 5, we examined how to make routing robust to network changes. We considered the problem of routing in bandwidth-constrained networks such as wireless and mobile ad-hoc networks, which are additionally characterized by time-varying network topology. We specifically proposed a braid routing algorithm that is robust to changes in the network topology. We analytically characterized the reliability of a class of braids, their optimality properties, and counter-examples to conjectured optimality properties in a well-structured (grid) network. Comparing with dynamic source routing, we showed that braid routing can significantly decrease control overhead while only minimally degrading the number of packets delivered, with gains dependent on node density.

6.2 Future Work

Consider the problem of where to focus sensing. For future work, rather than identifying a policy that chooses the best action to execute in a state for a single decision epoch, as in Chapter 2, it may be useful to consider actions that cover multiple epochs, as in semi-Markov decision processes or to use controllers from robotics [32]. Another direction for future work is to compute an upper bound on the quality that can be achieved for a given storm track trace and re-scan interval. This could potentially be done by using a limited lookahead sensing strategy and assuming deterministic storm movements.

Next, while the work in Chapter 3 assumed that all sensor requests are known *a priori*, in future work, we are interested in online versions of the call admission control problem in virtualized private sensor networks, where new sensor requests appear over time. Here there is related literature on online interval scheduling [44, 56]. We are also interested in decentralized methods to solve the call admission control problem. For example, rather than requiring the call admission controller to have global knowledge (particularly knowledge of all user utility functions), the controller could instead iterate back and forth with users to converge upon an acceptable solution (e.g., as is possible for routing [24]). Finally, there are interesting trade-offs between maximizing the utility of the sensor requests executed and user fairness: one way to address the problem of fairness would be to require users to pay to have their requests satisfied.

Considering the problem of transmitting both sensor control and data packets in the presence of network congestion studied in Chapter 4, it would be interesting to see whether other sensor network applications, besides tracking, have performance metrics for which gains can accumulate across multiple decision epochs when sensor control traffic is prioritized over data traffic. Another direction for future work is to reduce the amount of sensor data that must be transmitted over the network. This could be done by summarizing or compressing the data, or by changing the sensing strategy so that less data is actually collected. For instance, it may not make sense to collect data if there is insufficient bandwidth to transmit the data to a control center. While this work assumed that each sensed value is equally valuable, in practice, sensed data from areas of interest, such as areas of high reflectivity in the meteorological application, are likely to be more important to a sensing application, e.g., see [54]. These data values could be handled at higher priority, while other data values could be transmitted at lower priority or discarded in times of congestion. We also assumed that when sensor control packets are dropped, that the default sensing strategy was to scan 360°. Thus, another direction for future work is to instead assume

that radars have some intelligence and are not solely reliant on the control center for sensor controls. The more general challenge is to define an overall architecture for pushing application-level performance considerations down into the lower layers of the system stack in an application-independent manner.

Finally, with respect to to the problem of how to make routing robust to network changes studied in Chapter 5, there are several directions for future work. Rather than using a fixed braid width, we are interested in techniques to locally widen the braid to meet a robustness target. While we focused on a single network flow in this work, we would also like to consider multiple flows, and to explore rate control mechanisms such as backpressure routing [92] for local forwarding to achieve a solution that is robust in throughput as well as connectivity. It would also be interesting to incorporate link changes observed at the link layer into both the braid structure and into how rate control is performed within the braid. This could be done, for instance, by locally re-constructing the braid when link failures are observed instead of just performing local forwarding. Finally, we would generally like to investigate how network characteristics impact the control overhead incurred when routing in dynamic environments, and in using the resulting insights to design robust routing algorithms. Given a model of how network characteristics are expected to change over time, how routing is performed can then be adapted to decrease the amount of control overhead incurred. For example, as network connectivity decreases, it may eventually become preferable to flood (rather than route) packets to their destinations. Route re-construction and the associated control overhead could then be postponed until network connectivity improves. By additionally choosing routing state that is robust to network changes, and by understanding how accurate routing state must be to achieve performance goals, unnecessary state updates can be avoided and control overhead can be further decreased.

BIBLIOGRAPHY

[1] The network simulator ns-2. In *http://www.isi.edu/nsnam*.

[2] Akyildiz, I., Melodia, T., and Chowdhury, K. A survey on wireless multimedia sensor networks. *Computer Networks 51* (2007).

[3] Arkin, E., and Silverberg, E. Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics 18* (1987), 1–8.

[4] Balakrishnan, H., Padmanaban, V., Fairhurst, G., and Sooritabandara, M. TCP performance implications of network path asymmetry. *Request for Comment, RFC 3449* (Dec. 2002).

[5] Berman, P., and Dasgupta, B. Mult-phase algorithms for throughput maximization for real-time scheduling. *Journal of Combinatorial Optimization 4* (2000), 307–323.

[6] Bhatnagar, S., Deb, B., and Nath, B. Service differentiation in sensor networks. In *Fourth International Symposium on Wireless Personal Multimedia Communications* (2001).

[7] Black, U. *ATM: Foundation for Broadband Networks*. Prentice Hall, 1995.

[8] Boudec, J.-Y. Le, and Vojnovic, M. The random trip model: Stability, stationary regime, and perfect simulation. *IEEE/ACM Transactions on Networking* (2006), 1153–1166.

[9] Boyce, J. Noise reduction of image sequences using adaptive motion compensated frame averaging. In *ICASSP* (1992).

[10] Casella, G., and Berger, R. *Statistical Inference*. Duxbury, 2002.

[11] Center for Collaborative Adaptive Sensing of the Atmosphere. http://www.casa.umass.edu.

[12] Colbourn, C. J. *The Combinatorics of Network Reliabiilty*. Oxford University Press, New York, 1987.

[13] Consortium, International Engineering. Signaling system 7 (ss7). *http://www.iec.org/online/tutorials/ss7/* (2006).

[14] Cover, T., and Thomas, J. *Elements of Information Theory*. Wiley Interscience, 2006.

[15] Cox, D., and Isham, V. A simple spatial-temporal model of rainfall. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences 415:1849* (1988), 317–328.

[16] Das, S., Perkins, C., and Royer, E. Performance comparison of two on-demand routing protocols for ad hoc networks. In *Infocom* (2000).

[17] D'Costa, A., and Sayeed, A. Data versus decision fusion for classification in sensor networks. In *International Conference of Information fusion* (2003).

[18] Deering, S., and Cheriton, D. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems 8* (1990), 85–110.

[19] Donovan, B., Hopf, A., Trabal, J. M., Roberts, B. J., McLaughlin, D. J., and Kurose, J. Off-the-grid radar networks for quantitative precipitation estimation. In *Proc. European Radar Conference* (2006).

[20] Donovan, B., and McLaughlin, D. J. Improved radar sensitivity through limited sector scanning: The DCAS approach. In *Proc. AMS Radar Meteorology* (2005).

[21] El-Rewini, H., Lewis, T., and Ali, H. Task scheduling in multiprocessing systems. *IEEE Computer Society Press* (1995).

[22] Feitelson, D., Rudolph, L., and Schwiegelshohn, U. Parallel job scheduling a status report. Springer Verlag, pp. 1–16.

[23] Fredj, S. Ben, Bonald, Thomas, Proutiere, A., Regnie, G., and Roberts, J. Statistical bandwidth sharing: A study of congestion at flow level. In *Proc. ACM Sigcomm* (August 2001).

[24] Gallager, R. A minimum delay routing algorithm using distributed computation. *IEEE Transactions on Communications 25*, 1 (1977), 73–85.

[25] Ganesan, D., Govindan, R., Shenker, S., and Estrin, D. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *Mobile Computing and Communications Review 4*, 5 (2001).

[26] Ghosh, J., Ngo, H., Yoon, S., and Qiao, C. On a routing problem within probabilistic graphs and its application to intermittently connected networks. In *Infocom* (2007).

[27] Global Environment for Network Innovations (GENI) Project, http://www.geni.net/.

[28] Groenevelt, R., Nain, P., and Koole, G. The message delay in mobile ad hoc networks. Tech. Rep. RR-5372, INRIA Sophia Antipolis, Nov. 2004.

[29] Heffes, H., and Lucantoni, D. A markov modulated characterization of packetized voice and data traffic and related statistical multiplexer performance. *IEEE Journal on Selected Areas in Communication 4:6* (1986).

[30] Hokayem, P. F., and Abdallah, C. T. Inherent issues in networked control systems: A survey. In *Proc. American Control Conference* (2004).

[31] Hondl, K. Capabilities and components of the warning decision and support system - integrated information (WDSS-II). In *Proc. American Meteorological Society Annual Meeting* (2003).

[32] Huber, M., and Grupen, R. A feedback control structure for on-line learning tasks. *Robotics and Autonomous Systems 22(3-4)* (1997), 303–315.

[33] III, W. T. Miller, Glanz, F., and III, L. G. Kraft. CMAC: An associative neural network alternative to backpropagation. *Proceedings of the IEEE 78:10* (1990), 1561–1567.

[34] Institute of Computer Science IV, University of Bonn. BonnMotion: A mobility scenario generation and analysis tool. *http://web.informatik.uni-bonn.de/IV/Mitarbeiter/dewaal/BonnMotion/* (2005).

[35] Jayasumana, A. P., Han, Q., and Illangasekare, T. Virtual sensor networks - a resource efficient approach for concurrent applications. In *International Conference on Information Technology: New Generations* (2007).

[36] Johnson, D. B., and Maltz, D. A. Dynamic source routing in ad hoc wireless networks. *Mobile Computing* (1996).

[37] Johnson, D. B., Maltz, D. A., and Broch, J. DSR: The Dynamic Source Routing in Ad Hoc Wireless Networks (Chapter 5). In *Ad-Hoc Networking* (2001), pp. 139–172.

[38] Kalampoukas, L., Varma, A., and Ramakrishnan, K. Improving TCP throughput over two-way asymmetric links: Analysis and solutions. In *Proc. ACM Sigmetrics* (1998), pp. 78–89.

[39] Karenos, K., Kalogeraki, V., and Krishnamurthy, S. V. A rate control framework for supporting multiple classes of traffic in sensor networks. In *26th IEEE International Real-Time Systems Symposium* (2005).

[40] Karger, D. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM Review 43:3* (2001).

[41] Kellerer, H., Pfershy, U., and Pisinger, D. *Knapsack Problems.* Springer, New York, 2004.

[42] Kleinrock, L. *Queueing Systems Volume II.* Wiley Interscience, 1976.

[43] Kodialam, M., and Lakshman, T. V. Dynamic routing of restorable bandwidth-guaranteeed tunnels using aggregated network resource usage information. *Transactions on Networking 11:3* (2003).

[44] Kolen, A., Lenstra, J., Papadimitriou, C., and Spieksma, F. Interval scheduling: a survey. *Naval Research Logistics 54*, 5 (2007).

[45] Kompella, Vachaspathi P., Pasquale, Joseph C., and Polyzos, George C. Multicast routing for multimedia communication. *IEEE/ACM Transactions on Networking 1* (1993), 286–292.

[46] Kreucher, C., and III, A. O. Hero. Non-myopic approaches to scheduling agile sensors for multistage detection, tracking and identification. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing* (2005), pp. 885–888.

[47] Kumar, R., Crepaldi, R., Rowaihy, H., Harris III, A. F., Cao, G., Zorzi, M., and La Porta, T. F. Mitigating performance degradation in congested sensor networks. *IEEE Transactions on Mobile Computing 7:6* (2008).

[48] Kurose, J., Lyons, E., McLaughlin, D., Pepyne, D., Phillips, B., Westbrook, D., and Zink, M. An end-user-responsive sensor network architecture for hazardous weather detection, prediction and response. *AINTEC* (2006).

[49] Kwok, C., and Fox, D. Reinforcement learning for sensing strategies. In *IROS* (2004).

[50] Kyasanur, P., Padhye, J., and Bahl, V. On the efficacy of separating control and data into different frequency bands. In *Proc. Broadnets* (2005).

[51] Lee, S-J., and Gerla, M. AODV-BR: Backup routing in ad hoc networks. In *IEEE Wireless Communications and Networking Conference* (2000).

[52] Lee, S-J., and Gerla, M. Split multipath routing with maximally disjoint paths in ad hoc networks. In *IEEE ICC* (2002).

[53] Lemmon, M., Ling, Q., and Sun, Y. Overload management in sensor-actuator networks used for spatially-distributed control systems. In *SenSys* (2003).

[54] Li, M., Yan, T., Ganesan, D., Lyons, E., Shenoy, P., Venkataramani, A., and Zink, M. Multi-user data sharing in radar sensor networks. In *SenSys* (2007).

[55] Liang, B., and Haas, Z. Predictive distance-based mobility management for multidimensional pcs networks. *Transactions on Networking 11*, 5 (2003).

[56] Lipton, R., and Tomkins, A. Online interval scheduling. In *Fifth Annual ACM-SIAM Symposium on Discrete Algorithms* (1994).

[57] Mainland, G., Parkes, D., and Welsh, M. Decentralized, adaptive resource allocation for sensor networks. In *NSDI* (2005).

[58] Manfredi, V., Hancock, R., and Kurose, J. Robust routing in dynamic MANETs. Tech. Rep. 08-25, U of Massachusetts Amherst, Department of Computer Science, 2008.

[59] Manfredi, V., and Kurose, J. Scan strategies for adaptive meteorological radars. In *Advances in Neural Information Processing Systems 21* (2007).

[60] Manfredi, V., Mahadevan, S., and Kurose, J. Switching kalman filters for prediction and tracking in an adaptive meteorological sensing network. In *Proc. IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON)* (2005).

[61] Marina, M., and Das, S. On-demand multipath distance vector routing in ad hoc networks. In *IEEE ICNP* (2001), pp. 14–23.

[62] McLaughlin, D., Chandrasekar, V., Droegemeier, K., Frasier, S., Kurose, J., Junyent, F., Philips, B., Cruz-Pol, S., and Colom, J. Distributed collaborative adaptive sensing (DCAS) for improved detection, understanding, and predicting of atmospheric hazards. In *Proc. American Meteorological Society Annual Meeting* (2005).

[63] Mohring, R. H., Schulz, A. S., Stork, F., and Uetz, M. On project scheduling with irregular starting time costs. *Operations Research Letters 28* (2001).

[64] Mohring, R. H., Schulz, A. S., Stork, F., and Uetz, M. Solving project scheduling problems by minimum cut computations. *Management Science 49*, 3 (2003).

[65] Mosko, M., and Garcia-Luna-Aceves, J. J. Multipath routing in wireless mesh networks. In *IEEE Workshop on Wireless Mesh Networks* (2005).

[66] Murphy, K. A survey of POMDP solution techniques. Tech. Rep. Technical Report, U.C. Berkeley, 2000.

[67] Ng, A., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., and Liang, E. Inverted autonomous helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics* (2004).

[68] Nicolaou, N., See, A., Xie, P., Cui, J-H., and Maggiorini, D. Improving the robustness of location-based routing for underwater sensor networks. In *MTS/IEEE OCEANS conference* (2007).

[69] Pallot, X., and Miller, L. Implementing message priority policies over and 802.11 based mobile ad hoc network. In *Milcom* (2001).

[70] Papadimitratos, P., Haas, Z., and Sirer, E. Gun. Path set selection in mobile ad hoc networks. In *MOBIHOC* (2002).

[71] Pepyne, D., Westbrook, D., Philips, B., Lyons, E., Zink, M., and Kurose, J. Distributed collaborative adaptive sensor networks for sensing applications. In *American Control Conference* (2008).

[72] Pepyne, D. et al. Defining and Optimizing Utility in NetRad: a Collaborative Adaptive Sensor Network for Hazardous Weather Detection. *CASA Technical Report* (2006).

[73] Pepyne et al., D. Defining and optimizing utility in NetRad, a collaborative adaptive sensor network for hazardous weather detection, CASA technical report.

[74] Perkins, C. E., and Royer, E. M. Ad-hoc on-demand distance vector routing. In *IEEE Workshop on Mobile Computing Systems and Applications* (1999).

[75] Perkins, Theodore, and Precup, Doina. A convergent form of approximate policy iteration. In *NIPS* (2002).

[76] Provan, J.S., and Ball, M. O. Computing network reliability in time polynomial in the number of cuts. *Operations Research 32:3* (1984).

[77] Ramamritham, K., and Stankovic, J. Scheduling algorithms and operating systems support for real-time systems. In *Proceedings of the IEEE* (1994), pp. 55–67.

[78] Reeve, A. Resilient real-time communications across meshed networks under adverse conditions. In *1st SEAS DTC Technical Conference* (2005).

[79] Rodrigues-Iturbe, I., and Eagleson, P. Mathematical models of rainstorm events in space and time. *Water Resources Research 23:1* (1987), 181–190.

[80] Sanchez-Barbetty, M., and Jackson, R. Architecture for low cost electronically steered phased arrays. In *IEEE MTT International Microwave Symposium* (2008).

[81] Seto, D., Lehoczky, J.P., Sha, L., and Shin, K.G. On task schedulability in real-time control systems. In *Real-Time Systems Symposium, 1996., 17th IEEE* (1996).

[82] Shacham, N., Craighill, E., and Poggio, A. Speech transport in packet-radio networks with mobile nodes. *IEEE Journal on Selected Areas in Communications SAC-1:6* (1983).

[83] Spieksma, F. On the approximability of an interval scheduling problem. *Journal of Scheduling 2* (1999), 215–227.

[84] Stankovic, J., Spuri, M., Di Natale, M., and Buttazzo, G. Implications of classical scheduling results for real-time systems. *IEEE Computer 28* (1995), 16–25.

[85] Stone, P., Sutton, R., and Kuhlmann, G. Reinforcement learning for robocup-soccer keepaway. *Adaptive Behavior 3* (2005).

[86] Su, X., Chan, S., and Chan, K.-S. RLAR: Robust link availability routing protocol for mobile ad hoc networks. In *ICC* (2007).

[87] Sutton, R. Tile coding software. *http://rlai.cs.ualberta.ca/RLAI/RLtoolkit/tiles.html*.

[88] Sutton, R. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *NIPS* (1996).

[89] Sutton, R., and Barto, A. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts, 1998.

[90] Suvorova, S., Musicki, D., Moran, B., Howard, S., and Scala, B. La. Multi step ahead beam and waveform scheduling for tracking of manoeuvering targets in clutter. In *Proceedings of ICASSP* (2005), pp. 889–892.

[91] Tan, W. L., Yue, O., and Lau, W. C. Performance evaluation of differentiated services mechanisms over wireless sensor networks. In *Vehicular Technology Conference* (2006).

[92] Tassiulas, L., and Ephremides, A. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control 37:12* (1992).

[93] Trabal, J., Donovan, B., Vega, M., Marrero, V., McLaughlin, D., and Colom, J. Puerto Rico student test bed applications and system requirements document development. In *Proceedings of the 9th International Conference on Engineering Education* (2006).

[94] Tschopp, D., Diggavi, S., and Grossglauser, M. Hierarchical routing over dynamic wireless networks. In *Sigmetrics* (2008).

[95] UCLA Computer Science Department Parallel Computing Laboratory and Wireless Adaptive Mobility Laboratory. GloMoSim: A scalable simulation environment for wireless and wired network systems. *http://pcl.cs.ucla.edu/projects/glomosim/*.

[96] Walsh, G. C., and Ye, H. Scheduling of networked control systems. *IEEE Control Systems Magazine* (2001).

[97] Wang, B., and Hou, J. Multicast routing and its qos extension: problems, algorithms, and protocols. *IEEE Network 14* (2000), 22–36.

[98] Welch, G., and Bishop, G. An introduction to the Kalman filter. Tech. Rep. TR95-041, U of North Carolina at Chapel Hill, Dept. of Computer Science, 1995.

[99] Zilberstein, S., Washington, R., Bernstein, D., and Mouaddib, A. Decision-theoretic control of planetary rovers. In *Plan-Based Control of Robotic Agents, LNAI* (2002).

[100] Zink, M., Lyons, E., Westbrook, D., Kurose, J., and Pepyne, D. Closed-loop architecture for distributed collaborative adaptive sensing of the atmosphere: Meteorological command and control. *International Journal of Sensor Networks* (to appear).

[101] Zink, M., Westbrook, D., Abdallah, S., Horling, B., Lakamraju, V., Lyons, E., Manfredi, V., Kurose, J., and Hondl, K. Meteorological command and control: An end-to-end architecture for a hazardous weather detection sensor network. In *ACM Mobisys Workshop on End-end Sense-and-Response Systems* (2005).