OPPORTUNISITIC ROUTING AND MIDDLEWARE COMPOSITION FOR SENSOR AND ACTUATOR NETWORKS

By

Joel Wendamear Branch

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

Approved by the
Examining Committee:

Prof. Boleslaw Szymanski, Thesis Adviser

Prof. Alhussein Abouzeid, Member

Prof. Christopher Carothers, Member

Dr. John S. Davis II, Member

Prof. Carlos Varela, Member

Rensselaer Polytechnic Institute
Troy, New York

March 12, 2007

OPPORTUNISITIC ROUTING AND MIDDLEWARE COMPOSITION FOR SENSOR AND ACTUATOR NETWORKS

By

Joel Wendamear Branch

An Abstract of a Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

The original of the complete thesis is on file
in the Rensselaer Polytechnic Institute Library

Examining Committee:

Prof. Boleslaw Szymanski, Thesis Adviser
Prof. Alhussein Abouzeid, Member
Prof. Christopher Carothers, Member
Dr. John S. Davis II, Member
Prof. Carlos Varela, Member

Rensselaer Polytechnic Institute
Troy, New York

March 12, 2007

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

First, I must thank My Heavenly Father for gracefully supporting me with his love and blessings. I must also thank my parents, John and Billye Branch, and my brother, John Branch Jr., for continually encouraging me through the struggles of life and education. I owe all my success to them. I also thank my lovely wife and God's true blessing, Enobong, as well as her family, for always believing that I can and will achieve the impossible. I love you all.

Regarding the ideas presented in this thesis dissertation, I would first like to acknowledge my advisor, Prof. Boleslaw K. Szymanski, who has also encouraged and mentored me with the utmost sincerity and confidence. I could not have asked for a better guide throughout this process. I owe a large part of my success to him. I also thank Dr. Gilbert Chen, Mark Lisee, and Kamil Wasilewski for helping contribute vaulable ideas and hard work towards some items in this dissertation. Additonally, I thank all of my committee members: Prof. Christopher Carothers, Prof. Carlos Varela, Prof. Alhussein Abouzeid, and Dr. John S. Davis II. Last, I thank Prof. Volkan Isler for his insight and discussion.

Some ideas presented in this dissertation were also contributed by my colleagues at IBM Research. Therefore, I would like to thank Dr. Chatschik Bisdikian, Dr. Norman Cohen, Dr. Maria Ebling, Dr. Daby M. Sow, and again, Dr. John S. Davis II, for their thoughts and support.

The energy and inspiration required to complete my graduate work also came from many close friends. Therefore, I thank Dr. Alan and Kimberley Bivens, Dr. Kenneth Durgans, Dr. Juan Gilbert, Gemez and Dr. Brandeis Marshall, and Dr. Marcus Worsley. I love you all. Thanks so much.

ABSTRACT

Designing a receiver-decided cost-based routing protocol with transmission back-off delay and a local route repair algorithm establishes a novel and efficient wireless sensor network routing protocol that is fault-tolerant and seamlessly accommodates energy-efficient topology-control algorithms. Likewise, designing a new high-level middleware framework creates a foundation for developing extensible environmental control systems that may additionally require coordinating the behavior of coinciding actuators with interfering goals.

There exist many wireless sensor network routing protocols that achieve both fault-tolerant and energy-efficient routing. In many of these protocols, a node must maintain its neighbors' states (e.g., *active*, *sleeping*, *dead*) and adjoining links' qualities in order to support routing decisions. This approach is often inefficient for fault-prone environments because it requires additional communication overhead to route packets in a dynamic manner. Receiver-decided cost-based routing is better designed for communication in fault-prone environments since there is no reliance on maintaining neighbors' or links' states. This significantly lowers the overhead required to make dynamic routing decisions. Additionally, this technique is very extensible, easily allowing routing decisions to be guided by a combination of different criteria. The first contribution of this thesis is a receiver-decided cost-based routing protocol that uses transmission back-off delay and an optional local route repair algorithm to support fault-tolerant communication. Additionally, this protocol naturally accommodates out-of-band topology-control algorithms, which are often employed to reduce sensor network energy consumption. We show by simulation that this protocol performs better than typical static routing when dynamic network topologies are encountered.

There exist research activities that propose frameworks for supporting the composition of wireless sensor network middleware and applications. However, many produce device-level software that is often platform-specific. There exists a growing need for domain-specific frameworks to promote software extensibility and reusabil-

ity and integration with larger monitoring systems. Additionally, there is a growing need to coordinate actuators' behavior in response to sensor data and other actuators' behavior. The latter is critical because distributed control applications may manage actuators with individual tasks that are dependent on a shared pool of resources (e.g., treatment supplies or energy), significantly affecting an application's fidelity. Therefore, the second contribution of this thesis is the design of a high-level middleware framework to support the composition of sensor and actuator network systems that require the coordination of coinciding actuators in a distributed manner. An extensible set of manager components, supporting the essential tasks of sensor and actuator control systems, serves as the basis for system composition. A lightweight budgeting and auction-based mechanism is used to coordinate distributed actuators' access to shared and limited pools of resources. We describe the experience of using the framework to compose extensible sensor and actuator network middleware and show the benefits of its coordination-related features.

CHAPTER 1
Introduction

1.1 Background

1.1.1 The Pervasive Computing World

Recent advances in micro-electro-mechanical systems (MEMS) and wireless communications technology have fueled momentum in the field of pervasive computing. There exist various definitions for pervasive computing, with most of them following the lead of the late Mark Weiser's original vision presented in [74],

> The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.

Following this quotation, we state that the essence of pervasive computing involves the extension of computational resources into the human world so as to enable access to and control of data and processes from mostly anywhere in a largely transparent manner. Profound examples of pervasive computing systems are increasing. For instance, computing devices embedded throughout both the home and human body may continually monitor elderly and disabled people for impending medical emergencies. Also, modern cellular phones and networks have surpassed those supporting voice-only calls by also offering complete World Wide Web access almost anywhere on the planet. Furthermore, small embedded web servers will soon allow users to monitor and control any common household appliance via the World Wide Web, enabling the next generation of home automation applications [10]. By examining just these examples, it is evident that the pervasive computing paradigm has motivated, and continues to motivate, pivotal shifts in the design of communication infrastructures, client and server devices, and overall distributed network systems.

1.1.2 Sensor and Actuator Networks

A rapidly maturing component of the pervasive computing movement is the design and use of sensor and actuator networks, or *SANETs*, and related systems. Early SANET research primarily focused on distributed sensing systems for military purposes, with most early projects being led and supported by DARPA starting around 1980 [27]. The main purpose of these systems was to collect data describing enemy activities (e.g., troop and aircraft movement) for making strategic combat-related decisions. Supporting research projects regarding topics such as distributed resource management [71] and signal processing [53] for distributed sensor systems soon followed throughout the academic community. Most of the early sensor hardware used in the above examples consisted of large computational machines or radar stations with sizes on the cubic-foot scale. These sensors usually communicated with a central base station via direct wired or wireless connections. This configuration, particularly considering their hardware size, obviously restricted early distributed sensing systems to a limited class of applications that did not require unobtrusive monitoring or rapid set-up and removal of distributed sensor hardware.

Distributed sensor systems have undergone revolutionary changes in both design and usage, particularly within the last decade. Current-generation sensor hardware, or *nodes*, are outfitted with five major components: (1) a single (or multiple) sensors, (2) a microprocessor unit, (3) storage and program memory, (4) a wireless radio, and (5) a battery. Advances in MEMS technology have led to a significant shrinkage in the sizes of these devices. Now, sensor nodes are available in sizes roughly as small as a coin [24] and some current prototypes include device sizes reaching down to the cubic-millimeter scale [37]. Using their radios, these nodes may coordinate among themselves to form large scale wireless ad hoc networks by which sensed data may be shared among the nodes and transmitted to a base station in a multi-hop manner without the need for a static communication infrastructure. These present-day distributed sensor systems are commonly referred to as *wireless sensor networks (WSNs)* [81]. An illustration is shown in Figure 1.1.

WSNs have retained the same basic functionality as their predecessors in that they still perform environmental data collection. However, hardware modernization

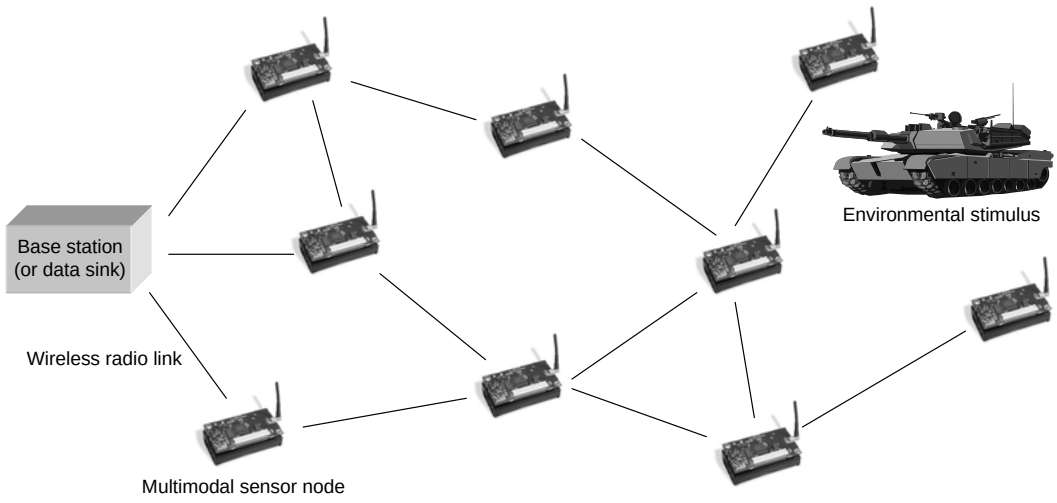


Figure 1.1: An example of a wireless sensor network.

has improved their capabilities and availability past those of their original counterparts. For instance, the decreased hardware sizes have enabled WSNs to become deeply embedded into various environments, enabling a growing class of applications that use large, dense populations of nodes to unobtrusively and remotely monitor sensitive environments. Furthermore, sensor hardware is becoming increasingly affordable, enabling an increasing availability of embedded data collection networks. The following is a list of some of the most popular instantiations and proposals for WSN applications realized by modern technology advancements:

- Monitoring and analyzing animal movement and behavior in remote habitats [45][47];
- Locating, tracking, and identifying targets for security and surveillance [5][30];
- Monitoring structural health for the safety of physical structures (e.g., bridges, buildings) [38][58];
- Monitoring product quality and safe operating conditions in manufacturing plants [39][55];
- Monitoring disabled and elderly people for supporting in-home health care [46].

Microprocessor unit	
Chip	ATMega128L, 7.37 MHz, 8 bit
Program flash memory	128 kbytes
Measurement serial flash	512 Kbytes
Analog to digital converter	10 bit ADC
Transceiver	
Protocol	IEEE 802.15.4
Frequency band	2400 MHz to 2483.5 MHz
Transmit data rate	250 kbps
RF power	-24 dBm to 0 dBm
Receive sensitivity	-90 dBm (min), -94 dBm (typical)
Outdoor range	75 m to 100 m
Indoor range	20 m to 30 m
Electromechanical	
Size (in)	2.25 x 1.25 x 0.25 (excluding 2 AA batteries)
Weight (oz)	0.7 (excluding 2 AA batteries)

Table 1.1: Hardware specifications for the Crossbow®MICAz ZigBee series mote.

To the best of our knowledge, the most popular hardware platform for WSNs is the Berkeley *mote* [24][62]. The hardware specifications for the Crossbow®MICAz mote [24], an example of a Berkeley mote that we use in our laboratory, is given in Table 1.1. Beyond its hardware specifications, the Berkeley mote platform is characterized by its use of the open-source TinyOS operating system, a component-based system designed for sensor nodes with limited memory and computational resources [34][43]. For most of the simulation-based experimentation performed for this thesis study, we emulate some of the radio specifications of the Berkeley mote platform. However, as well will explain in more detail later, our middleware-related research is platform-independent.

While WSNs have enabled significant advances in pervasive computing, they are only one part of a greater picture. WSNs are only responsible for the *collection* of environmental data, and in some cases, the analysis of it as well. Many applications must also assume responsibility for responding to sensed events and/or affecting a monitored environment via *actuators*. Actuators are commonly represented by either physical devices or more abstract services. Regardless of their representation,

actuators offer an application the option of responding to environmental phenomena in two ways. First, actuators can be used to simply *react* to sensed events. A good example of this would be a SANET-based perimeter intrusion detection system in which wireless motion sensors monitor the movement in restricted areas. Possible reactions to movement could be one of the following: (1) using a physical actuator to point a video camera at the location of the detected movement; or (2) using a radio-enabled actuator to notify those security personnel that are closest to the location of movement to investigate the event further. Second, actuators can be used to exert some form of *control* over an environment. A meaningful example of this would be a vehicle traffic management system in which roadside sensors are used to measure traffic congestion on various road segments. The goal of the system would involve relieving and preventing traffic jams by routing vehicles around areas of current and impending traffic congestion. Actuators may be used with traffic data to provide the following possible solutions: (1) roadside electronic signs (the actuators) could recommend alternative low-congestion routes to groups of drivers to take toward major destinations; or (2) in-vehicle navigation systems (the actuators) could recommend alternative low-congestion routes to individual drivers.

In essence, SANETs generally represent instantiations of control systems, which have been substantially studied in the field of automatic control systems and other related topics [1][40]. For instance, the first and second examples of SANET applications described above are clear examples of open and closed-loop systems, respectively. However, the increasing availability, pervasiveness, and standardization of embedded control technology (as evidenced by the formation of the ZigBee Alliance [83]) has enabled the creation of *Internet-scale* control systems that are abundant with networked and distributed computational power. This increased level of pervasiveness has enabled SANET applications to exercise environmental control beyond many traditional physical boundaries in ways not possible with the previous generation of control systems.

For an in-depth survey of well-known fundamental research problems regarding WSNs and SANETs, see [2] and [3].

1.2 Thesis Problem Statement

1.2.1 Wireless Sensor Network Routing

Many WSNs are expected to operate for extended periods of time in sensitive environments where human intervention may be infeasible or disruptive. Such expectations can be inferred from applications such as those described in [5], [30], [38], [45], [47], and [58]. In order to satisfy these expectations, WSNs must communicate, or route packets, in a largely autonomous manner to forward data to a base station. This thesis dissertation focuses on two major challenges arising from this requirement.

The first challenge is that WSN routing is naturally a fault-prone operation. This is fundamentally because wireless communication is not as resilient as wired communication due to various factors such as signal interference and multi-path signal propagation effects [64]. For various reasons, this lack of resiliency can be even further exasperated in WSNs. For instance, empirical data shows that the packet reception probability among sensor nodes is highly sensitive to atmospheric conditions (e.g., fog and rain) and the radio's ground elevation [4]; while the latter could be controlled, the former factors will likely change over time and are beyond human control. Other empirical data shows that *gray areas* frequently occur in nodes' communication ranges in which the packet reception probability wildly oscillates, creating a significant amount of asymmetry among wireless links [82]. Beyond wireless signal characteristics, routing faults may also arise simply from faulty node radios caused by imperfect manufacturing or environmental damage. All of these problems are compounded with the fact that most WSNs rely on ad hoc network configurations to enable the collection of data from sensors that are not within the communication range of a base station.

The second challenge is that basic communication in WSNs is an energy-intensive operation. More specifically, according to most hardware specifications and published experiments, radio operation typically requires more energy than operating the sensor and microprocessor components [4][24][52]. Furthermore, while a radio's transmission operation typically requires the most energy, a node's radio is envisioned to spend a large majority of its time in an idle state listening for packets.

Hence, while idling is comparatively a very low-power operation, it is very likely to consume the largest percentage of energy over the lifetime of the node. In addition to idle listening, over-listening is also a source of wasteful energy consumption and occurs when nodes receive packets that are addressed to other nearby nodes. Both idle listening and over-listening in WSN routing must be controlled since they affect the lifetime of the WSN and can unfairly drain energy from nodes that are not actively involved in any routing tasks.

Besides the fact that routing is a fundamental WSN requirement, all of the problems described above are important because they affect the overall quality and lifetime (or collectively, *reliability*) of WSNs and some SANETs as well. The traditional approach to addressing the problems above has involved the use of *unicast* routing algorithms. These algorithms are characterized by their use of routing tables that store the identity (or address) of the neighbor to which a packet must be directly forwarded in order to reach a particular destination. This approach is very similar to that of traditional wired networks, and requires nodes to constantly maintain knowledge of their neighbors' availability before making routing decisions. This approach is inefficient for WSNs because unexpected node failures (or subversions) and transient wireless links can cause unexpected topology changes that will invalidate knowledge of neighbors' availability. Hence, especially given a WSN's resource constraints, it is important to address the afore-mentioned problems with an efficient algorithm that does not require maintaining neighbors' states and adapts to network topology changes.

1.2.2 Sensor and Actuator Network Middleware Composition and Actuator Coordination

Dense deployments of WSNs and comparable sensor-enabled systems (i.e., not necessarily employing Berkeley motes) are expected to grow in usefulness and availability as they become more robust and their costs continue to decline; the same is expected for SANETs overall. Furthermore, SANETs are expected to evolve much in the same way as other networked resources such as databases and other mass storage devices and become easily and remotely accessible and controllable using

standardized protocols and interfaces. Following this trend, a growing number of new and existing large-scale control applications will use SANETs to achieve various goals. This thesis focuses on two challenges regarding these expectations.

The first challenge involves effectively managing and integrating SANET components to build SANET control systems. We state several observations regarding this particular challenge. One, the sensors and actuators to be integrated may exemplify various platforms (e.g., TinyOS or National Instruments). For some development tasks required in building a SANET control system (e.g., designing a data aggregation algorithm) using potentially heterogeneous components, it may be inefficient for the developer to learn and cope with the low-level details regarding how to access these components. Two, as suggested previously, a SANET control system could be associated with multiple underlying SANET components. It would be inefficient for a developer to access and control such a large amount of components on an individual basis especially since that process is not scalable. Three, the sensors and actuators made available to a system developer may be configured for different application purposes. For instance, a significant number of *off-the-shelf* sensor nodes contain multimodal sensor boards with the ability to sense light, sound, heat, etc. Therefore, the software used for building SANET control systems should remain extensible and reusable in order to reduce ongoing development costs.

The second challenge is that a growing number of available networked actuators presents the potential for actuator task interference. We define this problem as follows. For many applications, it is quite feasible to expect multiple actuators to use shared resources to fulfill their individual tasks. This may involve using a common resource for treatment supplies, as in the case of actuators treating soil plots with chemicals or fertilizers in an automated agricultural application. This may also involve using a shared energy resource, as in the case of a large-scale climate control system in which multiple heater and air conditioner components draw energy from the main energy grid or a shared power reserve. In the above scenarios, the common resource may be irreplaceable (within a given temporal context anyway), there may be an inflow capacity limit, or there may be an associated budget that limits the use of a resource at any one time. These scenarios present opportunities

for task interference, since the consumption of a resource can easily affect another actuator's ability to perform its task. Hence, actuator coordination algorithms will be important. It must also be noted that a centralized coordination algorithm could be infeasible taking into account the size and nature of the SANET system. First, a centralized algorithm easily presents a single point of failure, which can lead to catastrophic results in the environment. Second, a centralized coordination algorithm, and overall centralized system, may make scalability rather cumbersome.

The above problems are important because they affect the ability of SANETs to be integrated into larger, more meaningful systems. Furthermore, especially regarding actuator coordination, they affect a SANET control system's ability to act effectively and safely. If these problems are not addressed, SANETs will experience limited usefulness and risk of not being accepted by the larger software engineering community. A feasible solution to the above challenges is providing middleware software to help facilitate users' access to underlying SANET components while also abstracting away details for configuring, controlling, and querying those components. Within this solution set, middleware **instantiations** have been used to guide the behavior of WSNs in response to application requirements. However, middleware instantiations are often designed to optimize specific tasks. Such tasks may include managing energy or network topologies. This often limits a middleware's extensibility in trying to manage other tasks. In general, middleware instantiations can be very effective for particular applications, but we argue that their static nature limits their overall usefulness. Continuing, middleware solutions have often come in the way of being embedded directly on a sensor or actuator device. A limitation of embedded SANET middleware solutions is that they are usually designed for specific sensor platforms (e.g., TinyOS) and hence are limited in reusability. Also, embedded middleware are ill-equipped to facilitate the integration of heterogeneous SANET components into larger remote applications since they operate only at the device level.

1.3 Thesis Contributions

In response to the previously described research problems, this thesis makes several contributions related to WSN communication and SANET middleware composition. The following thesis research statement succinctly introduces these contributions:

> Designing a receiver-decided cost-based routing protocol with transmission back-off delay and a local route repair algorithm establishes a novel and efficient wireless sensor network routing protocol that is fault-tolerant and seamlessly accommodates energy-efficient topology-control algorithms.
>
> Likewise, designing a new high-level middleware framework creates a foundation for developing extensible environmental control systems that require coordinating the behavior of coinciding actuators with interfering goals.

The first contribution of this thesis, supporting energy-efficiency in WSNs, is an algorithm entitled *Energy-efficient Sensor network Communal Routing Topology*, or simply *ESCORT*. ESCORT is a topology-control algorithm that reduces energy usage for WSNs. ESCORT's main novelty is the use of wireless signal quality metrics to form communities of sensor nodes. Specifically, ESCORT uses wireless link quality and received signal strength between nodes to decide community membership. Using these metrics helps form communities in which nodes are of close spatial proximity to each other and share a resilient quality of wireless communication with each other. During normal network operation, ESCORT communities behave in a *redundant* manner such that all nodes in a given community share both the same identity and list of nodes outside the community with which they can all directly communicate. This allows redundant nodes to schedule the active states of their radios among each other so that they can significantly reduce network energy usage. Furthermore, this is done in a manner that is largely transparent to the network layer of the sensor node's protocol stack.

We use simulation to show how ESCORT increases WSN lifetime while minimizing negative affects on routing performance. ESCORT is beneficial in that its

use of wireless signal quality metrics controls communities' membership sizes (which affects network connectivity) and sustains a high quality of intra-community communication (which affects the quality of coordination between nodes). Also, the use of wireless signal quality metrics lessens the reliance upon location-tracking technology such as global positioning systems (GPS) to form sensor network communities, which can be infeasible for some WSN deployments. ESCORT's network layer-independent operation is also beneficial in that it increases ESCORT's usefullness across applications.

The second contribution of this thesis, *Self-Healing Routing*, or *SHR*, further addresses energy-efficient, as well as fault-tolerant, WSN communication. SHR is a cost-based receiver-decided WSN routing protocol. SHR's main novelty is its avoidance of maintaining static routes and instead routing packets opportunistically using the notion of routing costs and a transmission back-off delay algorithm. The crux of SHR is that each node maintains its routing cost to known base stations (or other nodes) and when a node has a packet to send to a particular base station, it freely broadcasts it to all available neighbors with its own routing cost associated with the base station included. Any node that receives the packet compares its routing cost with that of the sender and schedules its retransmission of the packet if its own routing cost is lower than that of the sender. This, along with other details, essentially controls how many nodes retransmit a packet at each hop. For our purposes, SHR defines a node's routing cost as its hop distance away from a given destination. Therefore, SHR dynamically transmits packets along the shortest available known routes. In the event that a node has no neighbor with a lower hop distance than itself, SHR also provides an optional route repair algorithm which locally adjusts surrounding nodes' hop distances such that a packet finds (and future packets continue to follow) the next available shortest path.

SHR's primary benefit is that it naturally accommodates WSNs with dynamic topologies since it does not require nodes to maintain any information about the state of its neighbors or adjoining links. We note that these dynamic topologies may be caused by wireless link and device failures or energy-efficient topology-control algorithms (e.g., ESCORT) that may operate independently of SHR. Furthermore,

SHR's avoidance of link and neighbor state maintenance enables it to accommodate dynamic topologies with a lower amount of communication overhead often required by unicast routing protocols, lending to its energy-efficiency. We use simulation to highlight this benefit in comparing SHR to a typical unicast, static routing protocol for wireless ad hoc sensor networks.

The third and final contribution of this thesis, *Sentire*, addresses middleware composition and actuator coordination for SANET-enabled systems. Sentire is a framework for composing high-level middleware for SANET-enabled systems. Specifically, Sentire offers a collection of inter-connected manager components for facilitating the flow and processing of queries, data, and control messages between applications and underlying sensors and actuators. We note that these components may also be heterogeneous in nature. Sentire's manager-based architecture serves to (1) provide extensible and reusable components for composing SANET middleware; (2) provide intuitive middleware primatives to shield solution developers away from intimate details of underlying SANET resources and to provide scalable control over sensors and actuators; and (3) logically separate SANET middleware development tasks based on system functionality (e.g., data processing, resource management, and query translation). As a particular novelty, beyond providing a manager-based development architecture, Sentire also supports decentralized actuator coordination between multiple Sentire middleware instantiations. Specifically, Sentire's actuator coordination functionality serves to manage the behavior of actuators whose tasks interfere with each other's. In support of this, Sentire uses a lightweight budgeting and auction-based mechanism to control the self-interested actions of distributed actuators. This method was chosen since (1) it well applies to distributed control systems composed of multiple self-interested agents, (2) it does not require a complex degree of computation, and (3) it is very scalable and easily allows Sentire instantiations to enter and leave the coordination process without causing a significant degree of reconfiguration. We describe our experiences using Sentire to compose and coordinate control systems using sensor and actuator devices as well as real-time simulation scenarios.

Regarding the first two contributions, we note that we do not address mobile wireless sensor networks. The topology-control algorithm and routing protocols proposed here are designed for battery-operated static sensor nodes. We do not consider node mobility for several reasons. One, mobility requires an extended amount of available energy. Since we assume the use of untethered battery-operated nodes that are not rechargeable, we also assume that there is not enough energy to drive motors for mobility. Second, there is a wealth of WSN applications that do not require mobility. Such examples include security, home monitoring, intelligent building management, and vehicle traffic management. Three, there are many proposals for applications that will use a mix of static and mobile nodes. In such a case, mobile nodes can communicate with their nearest one-hop static neighbor, and end-to-end routing can be conducted by static nodes. In that case, the contributions proposed here can still apply.

Regarding the third thesis contribution, we make clear that Sentire does not address the in-network processes of SANETs. Sentire addresses end-to-end middleware solutions. Therefore, SANETs may be configured as ad hoc or single hop networks. Mobility may be present as well, as long as there are protocols implemented on the SANET end that can accommodate such networks.

1.4 Dissertation Outline

The remainder of this document describes in detail this thesis's contributions to the previously discussed problem statement. The following list introduces the content of the chapters composing the remainder of this document.

- **Chapter 2:** This chapter provides a discussion of research efforts in the literature that are related to this thesis's contributions.
- **Chapter 3:** This chapter describes ESCORT, this thesis's contribution in energy-efficient topology-control for WSNs. Performance evaluation results are discussed.
- **Chapter 4:** This chapter describes SHR, this thesis's contribution in opportunistic, fault-tolerant, and energy-efficient WSN routing. Performance

evaluation results are discussed.

- **Chapter 5:** This chapter describes Sentire, this thesis's contribution in providing a SANET middleware framework used for decentralized actuator coordination. Experiences in using the middleware and performance results describing actuator coordination behavior are also discussed.

- **Chapter 6:** This chapter concludes this dissertation by summarizing the thesis's contributions, describing examples of some applications that would benefit from the contributions of this thesis, and describing potential future research activities that have been identified during the course of this thesis work.

CHAPTER 2
Related Literature

2.1 Wireless Sensor Network Routing Protocols

Our discussion of WSN routing protocols is separated into three sections: *unicast*, *cost-based*, and *topology-control* protocols.

2.1.1 Unicast Protocols

As previously mentioned, unicast protocols use traditional routing tables to propagate data from a source to a destination. Each node uses a table which stores the identifier (ID) of the neighboring node to which it should *directly* forward (hence, the label *unicast*) packets so that they eventually reach a given destination.

From the most general perspective, unicast protocols are classified as either *proactive* or *reactive*. Proactive protocols continuously establish and maintain routes throughout the network regardless if data needs to be sent. Prime examples of proactive routing protocols include Destination-Sequenced Distance Vector (DSDV) routing [60], Fisheye State Routing (FSR) [59], and Fuzzy Sighted Link State (FSLS) routing [65]. Proactive protocols are generally too aggressive for WSNs in that routes to all nodes in the network are constantly maintained, instead of just those routes which lead to destinations of interest. This maintenance of global topology information is wasteful both in terms of storage memory and communication energy. This problem can be further exacerbated if a WSN scales up to hundreds of nodes in size. Reactive protocols differ from proactive protocols in that they seek to establish routes only to specific destinations and only when they are needed. This approach better accommodates WSNs mainly because it is less wasteful of communication energy. Therefore, the remainder of the protocols we discuss in this chapter will be reactive.

Two seminal reactive routing protocols are Dynamic Source Routing (DSR) [36] and Ad-hoc On-demand Distance Vector Routing (AODV) [61]. The crux of DSR is the use of a route request-and-reply mechanism that results in a source

knowing the entire end-to-end route to a destination. This places most of the routing overhead on the source and naturally places a disproportionate memory burden on those sources that lie furthest from a common destination. In WSNs, where nodes' memory resources are scarce, it would be better to evenly spread the routing overhead among a larger group of nodes. In AODV, nodes involved in routing maintain a routing table that stores the IDs of neighbors through which packets should be routed to reach a particular destination. If a node detects an error-prone link in trying to forward data, it will it initiate an entirely new route request operation from its point in the route to find a new path. This is an inefficient process which requires a large amount of communication overhead to successfully route packets in a dynamic environment.

A significant number of unicast WSN routing protocols also fall under the classification of *data-centric*. In data-centric routing, users largely do not request data by querying specific nodes. Instead, queries describe properties that characterize the data that users want to receive and any node that can answer the query does so. Even if queries are addressed to specific nodes, the transmission of data is largely controlled by data-centric properties. A seminal data-centric protocol is Directed Diffusion [35]. Here, a user broadcasts *interest* packets into the network for sensor data with specific features (e.g., geographic location of data and requested data rate). This process populates nodes' routing tables much in the same way as AODV. However, neighbor IDs are associated with data interests, not a querier's address. Hence, when an interest reaches a qualifying node, it sends its data to the neighbor from the interest arrived and this process repeats at each hop until the data reaches the user. The protocol offers a reinforcement mechanism to both help filter the reception of the most desirable data and perform local route repairs when needed.

Many other data-centric protocols exist. In Declarative Routing Protocol (DRP) [23], data is routed similarly as in Directed Diffusion. However, the network uses a cluster-based topology and interest dissemination and data routing is primarily conducted by cluster heads. Factors such as reachability, residual energy, and link quality also determine how data is routed. Threshold sensitive Energy-Efficient

sensor Network (TEEN) protocol [48] and Adaptive Threshold TEEN (APTEEN) [49] use a similar cluster-based topology, however their data-centric approaches depart from that of Directed Diffusion and DRP. In TEEN, cluster heads filter the data sent from its clusters by broadcasting data thresholds to its cluster members. A node may send data to its cluster head only if the data's value exceeds the threshold. APTEEN is similar, only its cluster heads aggregate and store data in order to support a broader range of queries which require the analysis of historical data.

Address-based protocols are adequate for WSN applications in which dynamic topologies are rarely encountered. However, as previously discussed, they are not well-equipped for harsher environments in which network topologies are expected to change frequently and unexpectedly.

2.1.2 Cost-based Protocols

Whereas unicast protocols define routes based on neighboring nodes' IDs, cost-based protocols define routes based on some concept of routing *cost* between the node and a destination.

Cost-based protocols are classified as either *sender-appointed* or *receiver-decided*. In sender-appointed protocols, nodes must constantly maintain knowledge of their neighbors' routing costs and availability. One such protocol is Energy Aware Routing (EAR) [67]. In EAR, nodes compute the cost of forwarding a packet to a neighbor as a function of the neighbor's geographical distance from a destination and its residual energy. These costs are then converted to floating point values which enable nodes to probabilistically decide to which neighbor to forward data to. This results in the use of multiple paths between a source and destination. While sender-appointed protocols, such as EAR, enable more dynamic decisions about routing packets than unicast protocols, they still require the constant maintenance of neighbors' states.

This thesis's contribution in WSN routing is classified as receiver-decided, or *opportunistic*. In these protocols, nodes do not maintain any state regarding their neighbors. Instead, data is freely broadcast to all neighbors and they alone make the decision to forward the data based on cost values. This approach better

accommodates frequent and/or spontaneous topology changes since these changes do not require nodes to continuously evaluate the status of their neighbors.

Three well known receiver-decided protocols are Gradient Routing (GRAd) [63], GRAdient Broadcast (GRAB) [29], and ReInForM [25]. In GRAd, a route request-and-reply mechanism results in each node establishing a cost table. The critical information that composes the table entries are a destination ID (which indexes the table) and an estimated cost value which denotes the number of hops it takes the current node to reach the respective destination. After cost tables have been built, when a node wants to send data to a destination, it broadcasts a data packet, containing the destination ID and a remaining cost field indicating the expected number of hops it should take the packet to travel toward its destination, to all of its neighbors. Upon receiving the data packet, a neighbor compares the packet's remaining cost field with the estimated cost value that is stored in the cost table entry for the destination. If, and only if, the table entry's cost value is lower than that indicated in the packet, the node decrements the packet's remaining cost value and then rebroadcasts the data packet. However, if the neighbor overhears the same packet being rebroadcast with a lower remaining cost field before it itself has forwarded the packet, it will cancel its transmission to avoid flooding the network with redundant packets. This process repeats until the data reaches the destination. GRAd implements route repair largely through the use of end-to-end acknowledgments.

GRAB is very similar to GRAd. The network builds and maintains a cost field through which only those recipient nodes with a lower hop count to the destination than the sender are allowed to rebroadcast a received packet. However, GRAB uses an adjustable credit system which enables nodes to forward packets based on an additional cost budget in addition to forwarding based on hop count. This helps control the number of neighbors that can forward copies of the same packet which in turn creates non-disjoint paths of redundant data that strengthen fault-tolerance. ReInForm is similar to GRAB in that it also focuses on fault-tolerance. However, it makes a strong consideration of link error rates in determining the amount of path redundancy which it judges to be beneficial for fault-tolerance.

Another receiver-decided protocol is ExOR [8]. The crux of ExOR is that it transmits batches of packets simultaneously. With each packet, a source node will include a list of candidate forwarders that are prioritized by their geographic distance to the destination. Recipient nodes then buffer all packets in the batch and the node with the highest priority retransmits the buffered data with each packet containing a batch map. The batch map indicates the sending node's estimate of the highest-priority node believed to have received each packet. The remaining recipients, in order of priority, then broadcast their buffered packets that were not acknowledged in the batch maps of higher priority nodes. This process continues in order to get as many packets to the destination as possible. This scheme affords ExOR the opportunity to give each packet transmission multiple opportunities to reach the destination. However, a protocol must be used so that all nodes can agree on recipient nodes' identities and priorities. While the recipient nodes largely decide forwarding decisions at every hop, the assignment of priorities by the sending node also indicates that the sending node must maintain some information regarding its individual neighbors.

Other receiver-decided protocols have extended the principles of those described above. One major advancement regards the use of transmission back-off delay mechanisms that help facilitate the packet forwarding process and prioritize the recipient nodes in determining which one is chosen to forward a packet. One such protocol that uses this technique is Dynamic Delayed Broadcasting (DDB) [33]. DDB assumes that nodes know their locations (e.g., via GPS coordinates), the location of the destination, as well as their own transmission ranges. When a node broadcasts data, it includes its own location in the packet. Nodes that receive the data packet do not rebroadcast immediately, but determine a back-off delay before which they rebroadcast the packet. The back-off delay is determined as a function of the estimated additional area, given the sender's location, that it would cover with its own transmission. The length of the delay is inversely proportional to the amount of additional coverage, so that nodes that have a higher probability of moving the packet farther will retransmit earlier. If a node overhears another node rebroadcast the packet, it will adjust its back-off delay according to the new

estimate of its additional coverage area given the new and original transmissions. The new back-off time is also reduced by the time already spent in the previous back-off interval. This process repeats with any new overheard retransmission. A node will not rebroadcast a packet if it estimates its additional coverage area to be less than a prescribed threshold value. DDB is more of a *flooding* protocol, as the protocol's cost field does not disseminate packets toward any specific destination, even though the packet eventually reaches one. We prefer the traditional approach in which packets are routed toward a destination, as it is more efficient and beneficial for energy-constrained WSNs.

Geographic Random Forwarding (GeRaF) [84][85] also uses a transmission back-off delay mechanism. In GeRaF, all nodes are assumed to know their own locations as well as the destination's. In addition, GeRaF assumes the use of sensor nodes with two radios with different frequencies. One frequency is used for transmitting data while the other is used for transmitting a busy tone. When a node wants to send data, it checks both frequencies to see if either is busy. If either is busy, the node reschedules another transmission for later. However, if both are idle, the node broadcasts a request-to-send (RTS) packet with its own location and the location of the destination to all of its neighbors over the data frequency. Only those neighbors residing in an *eligibility region*, which is the portion of the sender's transmission area that is closer to the destination than the sender, are allowed to respond to the RTS message. GeRaF organizes the eligibility region into a finite number of *priority regions* that are associated with transmission time slots. The priority regions are ordered such that nodes in the furthest region, being the closest distance to the destination, will attempt to respond to the RTS packet in the earliest transmission slot. Upon receiving a RTS packet, a receiver will transmit a beacon on the busy tone frequency, to prevent collisions with other nodes, and determine its priority region and respective transmission slot using the destination location found in the RTS packet. The highest priority nodes will attempt to respond to the RTS message by broadcasting a clear-to-send (CTS) packet back to the sender. If the sender does not receive a CTS packet for the first time slot, indicating that there are no eligible nodes in that slot, it will signal nodes of the next highest priority

to continue. This process continues until either a node replies, or all priority zones have been exhausted. If the latter occurs, the sender will attempt to retransmit the RTS packet a finite number of times until it gives up. On the other hand, if the sender receivers a CTS packet, it will transmit its data packet to the receiver from which the CTS packet was received. Upon hearing the start of the transmission of the data packet to a designated receiver, all other receivers will deactivate their radios to save energy. The sender knows the receiver has successfully received the data packet when it receives an acknowledgment packet from it.

Other receiver-decided protocols are similar to GeRaF in that they use geographic location data to establish a group of eligible forwarders among all recipient nodes. Beacon-Less Routing (BLR) [32] establishes an eligibility region such that all nodes in the region are closer to the destination than the sending node and are also able to overhear each other's packet transmissions. Upon receiving a data packet, a node in the eligibility region will set its transmission back-off timer according to its distance from the destination. When a node's timer expires, it will rebroadcast the packet. Any other eligible node that overhears the rebroadcast before its own timer expires will cancel its timer and drop the packet. After a sender knows which neighbor rebroadcast the packet, it will send future packets *only* to this node using unicast delivery for a short time interval. During this interval, the sender will also attempt to save energy by adjusting its transmission power to only reach this particular node. A sender will retransmit a packet if no response if heard. However, in this case, no eligibility regions will be used and a recipient node's delay will be based on the angle between itself, the sender, and the destination; the larger the angle, the longer the delay.

In Priority-based Stateless Geo-Routing (PSGR) [78], location is also used to establish eligibility and priority regions, similar to GeRaF. However, PSGR attempts to define the priority regions in such a way that each region contains only one eligible recipient node. This is done to reduce the probability of receivers' transmissions colliding. The transmission back-off delay mechanism is similar to GeRaF's as well. PSGR offers two solutions in the case that no eligible nodes respond. One, the sender rebroadcasts the data packet a finite number of times in hopes that a node

will enter the eligibility region. Two, the sender will use a technique to try and route packets along one side of the original eligibility region. Because we do not use location coordinates in our related research, we use a different approach to route repair.

Several other protocols are very similar in nature. In Implicit Geographic Forwarding (IGF) [9], only those lower-cost neighbors, with cost being a function of residual energy and geographical distance from the destination, residing in an eligibility region defined by a 30° angle of a straight line connecting the source and destination are allowed to compete for packet forwarding rights. If no eligible nodes are found, the eligibility region may be shifted so that the sender can find an eligible node. In the Distributed Passive Routing Decision (DPRD) protocol [70], transmission back-off delays are used to select forwarding nodes, but no eligibility regions are used and no techniques are presented that describe how the protocol reacts to the absence of nodes which lie closer to the destination than the sender. In the State-free Implicit Forwarding (SIF) protocol [18], again, eligibility regions are used. It also uses a back-off delay packet forwarding mechanism with a RTS/CTS scheme similar to that of GeRaF. If no eligible recipient nodes are found, the sender will first increase its transmission power and rebroadcast the RTS packet. If this fails, the sender will deactivate itself for a short time to prevent further packets from being routed through it. The Simple Geographic Relay (SGR) protocol [19] is very similar to SIF. However, it avoids the use of RTS/CTS packet forwarding schemes.

Overall, we feel that the use, and energy requirement, of RTS/CTS handshaking mechanisms and GPS technology can be avoided in providing fault-tolerant routing. In Chapter 4, we describe this thesis's contributions to cost-based receiver-decided routing and describe how it markedly differs from and improves upon the related research background described here.

2.1.3 Topology-Control Algorithms

Regarding WSN communication, this thesis's primary contributions lie in the area of opportunistic routing. However, in some of our background research, we also contribute a cluster-based topology-control algorithm that operates indepen-

dent from routing behavior. We note that this promotes energy-efficient communication, which clearly addresses part of the problem statement guiding this thesis's research. Therefore, we also describe some background research in topology-control algorithms.

Topology-control algorithms aim to reduce network energy consumption by selectively activating and deactivating nodes' radios. In some cases, all of a node's components may be powered down as well. Topology-control algorithms largely operate independently of the network layer. Three notable examples of topology-control algorithms are Span [17], STEM [66], and ASCENT [16]. Span aims to establish coordinator nodes in the network which remain active to route packets while other node deactivate their radios (or *sleep*) and periodically wake up to judge if they should become coordinators. A node becomes a coordinator if it determines that two of its neighbors cannot communicate with each other either directly or via one or two other coordinators. Upon deciding to be a coordinator, a node broadcasts a hello packet advertising to its neighbors that it has assumed coordinator status. In the event that contention arises for becoming a coordinator, randomized back-off delays of hello packet transmissions are used; the first node to transmit a hello packet becomes the coordinator. Residual energy levels may also be used to break contentions. A node also periodically checks if it should relinquish coordinator status by checking if all of its neighbors can communicate either directly or via one or two coordinators. The details of Span ensure that an effective number of coordinators remain active in order to promote energy-efficiency while also maintaining network connectivity.

In STEM, nodes do not activate their radios until either they have data to transmit or want to receive data. STEM achieves this through the use of a dual-radio scheme. One radio operates on a frequency used only for data transmission while the other operates on a frequency used for wake up signaling (similar to GeRaF). These are referred to as the *data* and *wake up* planes, respectively. When a node wants to transmit data to a specific neighbor, it will first transmit a beacon over the wake up plane to the neighbor. If the neighbor's wake up radio, which sleeps periodically, is active and receives the beacon, it will transmit a response to the sender over the

wake up plane and activate its other radio to receive data. If, however, the wake up radio is not active, the sender node will try signaling the receiver again until it receives a response. Once the sender receives a wake up response, it will transmit its data to the receiver over the data plane. Once data transmission is accomplished, both nodes' data radios will again deactivate while the wake up radios periodically activate to listen for beacons.

ASCENT defines two types of nodes: *active* nodes that participate in routing by both receiving and transmitting packets and *passive* nodes that do not participate in routing, but allow the reception of packets. During data transmission, if a recipient node starts to detect packet loss from a neighbor (because of a weak wireless link), it will signal some passive nodes between it and the neighbor to become active and forward packets to help prevent the packet loss. The newly active node may also solicit other nearby passive nodes to help further reduce packet loss if it is unable to do it alone. Once the receiving nodes determine that the original weak link has returned to a reliable state, the newly activated nodes are signaled to return to a passive state. Other protocols which are very similar to Span and ASCENT include Basic Energy-Conserving Algorithm (BECA) and Adaptive Fidelity Energy-Conserving Algorithm (AFECA) [75]. BECA sacrifices latency in order to aggressively deactivate nodes' radios as much as possible. AFECA determines a node's sleep time largely as a function of the density of its local neighbors; the more neighbors are available, the longer a node is allowed to sleep.

Some topology-control algorithms aim to establish clusters throughout the network in which nodes can coordinate sleep schedules to balance energy usage. A seminal cluster-based topology-control algorithm is Geographical Adaptive Fidelity (GAF) [76]. The crux of GAF is its use of nodes' location coordinates and transmission ranges to establish a virtual grid upon the network. Nodes use their location coordinates to associate with a particular square in the grid. The grid is established such that all nodes residing in any one square, or cluster, are *redundant* in that they can all communicate with any node in an adjacent cluster. Hence, only one node in a given cluster must be awake at any one time. Duty cycles are established in each cluster to balance energy usage among all nodes. At the beginning of a

new interval, a discovery phase starts in which all nodes in a given cluster discover their redundant neighbors and determine which one has the most residual energy. The one with the most residual energy becomes the active node, estimates the following interval duration for which it should expend a prescribed percentage of its energy, and advertises this duration to its cluster. The active node then remains awake while all other nodes in the cluster deactivate their radios for the advertised duration, after which the entire process repeats.

GAF's applicability is limited by the availability of location-tracking technology. In response, another topology-control algorithm, Cluster-based Energy Conservation (CEC) [77] forgoes the use of location coordinates and instead directly measures network connectivity to identify redundant nodes. CEC organizes the network into overlapping clusters that contain cluster heads and are connected via gateway nodes which belong to multiple clusters. Clusters are defined as being a group of nodes that are mutually reachable in two hops. A discovery phase facilitates the election of cluster head and gateway nodes. Similar to GAF, a node in CEC selects itself as a cluster head if it has the most residual energy out of all other nodes in the cluster. Also, a cluster head must be one hop away from all the nodes in its cluster. A node may select itself as a gateway if it falls either within the communication range of multiple cluster heads or within the range of a combination of cluster heads and other gateways. This process may result in multiple gateway nodes between the same clusters. Gateway nodes that have the most residual energy and connect with the most cluster heads are more likely to be used to route packets between clusters since this helps both balance energy usage and maintain better network connectivity. After cluster heads and gateway nodes are selected, all remaining nodes deactivate their radios. Controlling nodes' duty cycles then proceed much in the same way as in GAF.

To the best of our knowledge, and as evidenced by the seminal algorithms described above, most cluster-based topology-control algorithms depend on GPS, or some similar location-tracking technology, to form clusters. Use of such technology incurs a cost that might be infeasible for particular usage scenarios. Chapter 3 describes how we depart from this approach.

2.2 Sensor and Actuator Network Middleware

Our discussion of SANET middleware is separated into two sections: *embedded* and *high-level*. A brief discussion on actuator coordination is also presented.

2.2.1 Embedded Middleware

Embedded middleware is meant to be deployed directly on actual sensor and actuator devices. One such middleware solution is the Impala middleware architecture [44]. Impala's major design features include application adaptation and code updating for wireless sensor networks. Impala provides real-time application adaptation to improve the performance, reliability, and energy-efficiency of software systems running on the sensor hardware. For instance, Impala may switch communication protocols according to the percentage of device failures or residual energy of network nodes. Also, Impala allows software updates to be wirelessly distributed to nodes and be installed to the running system dynamically. This is done in a modular fashion so as to avoid both bandwidth saturation and high energy consumption due to frequent radio operation.

In [56], an embedded modular architecture is presented to support cooperating mobile embedded systems. The architecture itself is composed of several layers. The first layer, above the hardware and operating system layers, is responsible for adaptive resource scheduling. This layer manages the varying quality of service (QoS) and resource requirements from the application with the actual availability of hardware resources to help guarantee predictable timing behavior for higher layers of the architecture. The next higher layer provides reliable cluster-based communication services for data transport. Communication services address functions such as ad hoc routing and distributed clock synchronization. The architecture's highest layer provides support for individual applications to form a consistent view of the global state and environment of the overall system. This enables applications to operate in an orchestrated way while still performing local decision-making. Our proposed research is similar to [56] in that we also intend to address the coordination of multiple control entities. However, we intend to further address this topic by providing mechanisms to more explicitly guide actuator coordination.

The cluster-based middleware described in [80] provides a virtual machine abstraction for separating application logic from the underlying WSN infrastructure. Two primary middleware layers compose the virtual machine. The first, and lowest, layer is the *cluster* layer. This layer is responsible for organizing nodes to form clusters around environmental phenomena of interest and monitor their behaviors. Clusters may also adapt and dynamically determine node membership in the case that the phenomena move among locations in the network. The second layer, *resource management*, is responsible for controlling the allocation and adaptation of a cluster's resources in accordance with applications' priorities and required levels of QoS. This may involve tasks such as attempting to balance performance among all applications or trading the quality of service of a single low priority application for its level of resource usage. All resource management activities are performed at cluster head nodes, which are chosen at the time of cluster formation based on resource availability (e.g., available energy and computational ability).

The Milan middleware [31] also places a strong emphasis on meeting applications' expectations of QoS while managing WSN resources for increased network lifetime. Here QoS specifically refers to the accuracy of a sensor's data. In order to meet its goal, Milan requires the following information: (1) the variables (i.e., sensor data) of interest to the application, (2) the required level of QoS for each variable, and (3) the minimum QoS that each sensor can provide for each variable. Applications make Milan aware of its requirements via *state based variable requirements* and *QoS* graphs. The former graph describes the minimum required QoS for selected variables given an application's state. The latter graph describes the minimum QoS of a variable that each sensor can provide. This graph also indicates what sensors can be combined to provide higher levels of QoS for given variables. Given these graphs and knowledge of the state of the network, Milan can dynamically determine a subset of sensors to query to meet an application's demand while efficiently managing the resources of the network.

A benefit of embedded middleware is its potential to easily exploit details of the underlying physical architecture or operating system of the SANET component on which it resides. However, as previously discussed in Chapter 1, embedded

middleware is often limited in reusability and does not focus on interoperability between heterogeneous sensor and actuator devices or services.

2.2.2 High-level Middleware

The role of high-level middleware is also to control SANET components according to applications' demands. However, instead of residing on sensors and actuators, high-level middleware operates on devices that physically separate the end-user from actual sensors and actuators. These devices can include base stations, proxy servers, or even the machine that an end-user may be using to access the SANET's resources.

The first work we discuss is the Garnet middleware framework [72]. Garnet exhibits several major features. First, Garnet places an emphasis on providing sensor data streams as programming abstractions, instead of abstracting individual sensor devices. Hence, a programmer defines a data stream describing the sensor events to be collected while the middleware queries and controls the appropriate underlying sensors. Second, an organization of *consumer* processes may be defined to collect and filter data streams on behalf of the programmer to either return information to the application or other consumer processes. This helps provide an extensible middleware infrastructure. Third, a *super coordinator* component monitors the activity of all consumer processes in order to help enforce system-based policies regarding balanced network resource usage. Garnet is not restricted to controlling WSNs, as actuators may be controlled by the framework as well. However, Garnet makes no special provisions regarding actuator coordination.

The CORTEX middleware framework [7] provides a sentient object programming model and event-based publish-subscribe communication service to support middleware composition for SANET systems. A sentient object consists of interfaces to sensor and actuator components as well as internal control logic. Sensor and actuator components are programmable abstractions that represent the characteristics of the underlying physical sensors and actuators and provide access to them in a platform-independent manner. The internal control logic consists of three components: *sensory capture*, *context hierarchy*, and *inference engine*. The sensory capture component fuses incoming sensor data and uses probabilistic models

to manage uncertainty of sensor data. The context hierarchy component maintains a hierarchy of environmental or operational contexts in which a sentient object can exist as well as its current active context. The contexts themselves are derived from the fused data of the sensory capture component. The inference engine component contains an event-condition-execution model which uses current and predicted contexts from the context hierarchy component to make actuation decisions. Event-based publish-subscribe communication between sensor, actuator, and sentient object components is implemented using STEAM (Scalable Timed Events and Mobility), a location-aware communication service designed especially for ad hoc wireless networks. In addition to context filters, STEAM provides proximity filters which validate the geographical origin of sensor events. GEAR (Generic-Events Architecture) [15] describes an extension to the CORTEX communication service. In GEAR, more emphasis is placed on reliability and timeliness by providing support to enforce temporal constraints on event propagation.

MediaBroker [51] provides a similar approach to CORTEX. However, it directs a larger focus on managing component interactions based on the ability to manipulate relevant data formats. The MediaBroker architecture is composed of the following components: *clients*, *transport and transformation engine*, *type system facility*, and *name server*. Similar to CORTEX, clients are used to describe media sources and destinations. It is possible for a client to be both a source and destination where it will consume sensor data, modify it, and then redistribute it to some other client(s). The transport and transformation engine consists of the *MediaBroker engine* and *data brokers*. The crux of the MediaBroker engine is to enable clients to join and leave the MediaBroker system at runtime by managing the creation of data brokers. All clients attach to data brokers to distribute and consume data to and from each other. A data broker monitors the provided data type of the attached media source as well as the requested types from attached destinations. The job of the the data broker is then to calculate the least upper bound of the data types requested by all attached destinations and then notify the source to produce data appropriately. The data broker may also further transform the data itself in order to satisfy the least upper bound. The type system facility supports data broker

operation in providing an intuitive language for describing data types, their relation to each other, and how they can be transformed. Last, the name server maintains a mapping between source and destination names and their respective computation resources and data structures. Again, this largely supports the operation of the data brokers. MediaBroker is largely designed for data sources, or sensors, that are capable of producing rich data types such as audio and video. The framework makes no provisions for environmental control via attached actuators.

Hourglass [68] is very similar to CORTEX and MediaBroker in that it uses a series of distributed processes to distribute data from sensors to applications while also performing functions such as data compression, aggregation, and filtering. In Hourglass, these processes, along with the network communication links that connect them, are referred to as *circuits*. Hourglass places a strong emphasis on the QoS of sensor data that travel to applications. Hence, *circuit managers* are used to select which nodes will form a circuit and where specific data transformation processes will be placed in the circuit based on resource availability (e.g., computational power, storage, and available bandwidth). Also, in the case that a sensor unexpectedly disconnects from the Hourglass system, circuit managers will adapt the affected circuits to either find suitable alternative sensors or implement application-specific actions such as buffering current data until the sensor returns.

A dense survey of research activities in embedded and high-level middleware for WSNs is described in [79]. Similar to the works described above, these contributions propose an array of different architectural approaches to middleware design as well as support a variety of different application tasks. Our research regarding SANET middleware is classified as high-level. However, we include support for distributed actuator coordination. To our knowledge, no other SANET middleware frameworks address this issue.

2.2.3 Actuator Coordination

To our knowledge, very little SANET-related research addresses distributed actuator coordination. Melodia et al. [50] describe in-network distributed protocols for actuator coordination to find the best actuator(s) to perform actions in a given

event area. Using a mixed integer non-linear programming model and a distributed auction protocol, the protocol attempts to optimally select actuators such that the action(s) to be performed occur in a minimum amount of time while also balancing the energy usage of all actuators involved. Ngai et al. [57] describe research that is similar to that described above. The first contribution is a real-time, distributed event-reporting algorithm for sensors to transmit data to actuators in a timely manner. The second contribution is an algorithm to determine which actuators to perform a specific set of actions based on actuators' physical proximity to the event in question. We note that Ngai et al. assume the use of mobile actuators.

The research described above focuses on how actuators should be selected to execute certain application tasks or how they should work together to execute tasks collectively over a large geographic area. However, these works make no provisions in the case that nearby conflicting actuator tasks are encountered.

CHAPTER 3
Energy-efficient Sensor Network Communal Routing Topology Using Signal Quality Metrics

3.1 Introduction

This chapter describes *ESCORT (Energy-efficient COmmunal Routing Topology)*, which represents this thesis's contributions in WSN topology-control [11]. ESCORT is characterized by its novel use of signal quality assessment (SQA) to cluster nodes based on wireless link quality and spatial separation. The resultant cluster defines a group of *redundant* nodes that function as a single virtual routing entity and thus, operate transparently under most WSN routing protocols as a single node. Establishing sleep schedules within the clusters enables one node to forward packets at any one time, while the remainder of its cluster members sleep. This enables the network to extend its lifetime and evenly consume energy. Our contribution regards the use of SQA to form clusters, which helps ESCORT mitigate the effect of intra-cluster packet loss and limit the extent of community formation, in turn helping to preserve the connectivity of the overall network. Using simulation, we show that ESCORT enables nodes to sleep up to 60% of the time with a small effect on packet forwarding performance. For clarity, Figure 3.1 illustrates ESCORT's position in a typical WSN protocol stack.

Figure 3.2 shows ESCORT's effect on a WSN in which clusters of redundant nodes are formed (indicated by the dotted circles around the black nodes), and

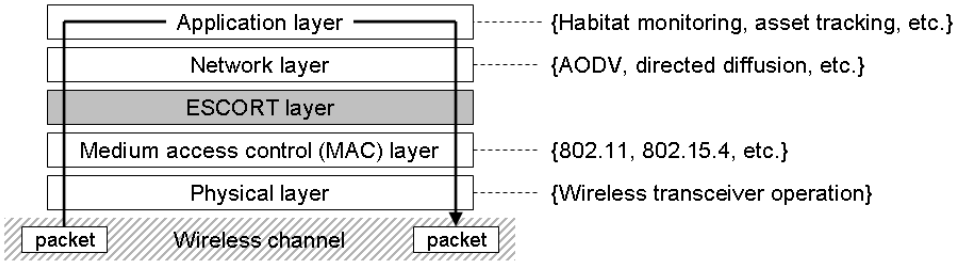


Figure 3.1: ESCORT's position in a WSN protocol stack.

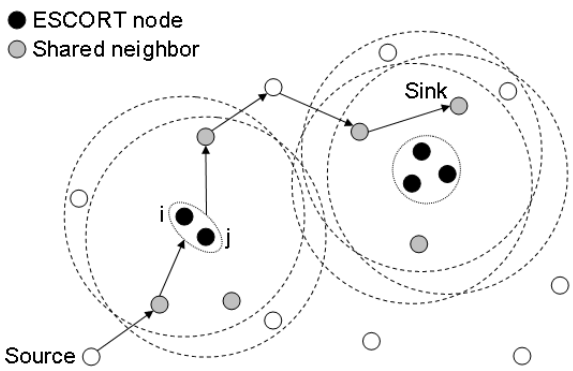


Figure 3.2: ESCORT topology applied to a small WSN.

shared neighbors, residing within the clusters' intersected transmission regions (indicated by the larger dashed circles), are established to ensure that clusters maintain only bidirectional links with neighboring nodes. In Figure 3.2, the example path is routed through the cluster on the left, in which either node i or j may forward the traffic depending on their radio's states.

Before continuing, we state some fundamental assumptions regarding ESCORT. First, sensor board components operate independently from radios and remain powered over all states. We do not address any challenges regarding coverage of the nodes' sensor components. Second, all wireless links are bidirectional, or *symmetric*. Third, nodes are stationary, not mobile. Additional assumptions are stated as needed.

3.2 The ESCORT Approach

3.2.1 Signal Quality Assessment

We define signal quality as a combination of two separate metrics: *link quality* and *signal strength*. We describe their uses below. Both factors help to determine the selection of redundant sensor nodes.

3.2.1.1 Link Quality

We use link quality assessment to form clusters of nodes with equivalent routing functionality for two main reasons. One, strong links promote energy-efficient packet delivery. Empirical data in [41] support this assertion by demonstrating that

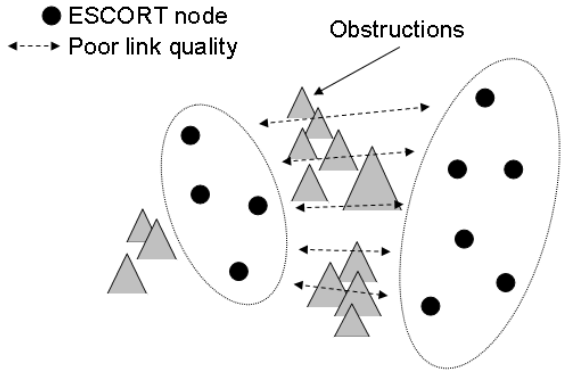


Figure 3.3: Effect of link quality assessment on ESCORT cluster formation.

extensive retransmissions over error-prone links waste energy. Two, strong intra-cluster links provide robust coordination that preserves the layer of transparency under the selected algorithm in the network layer. This helps ensure proper sleep scheduling and helps reliably share routing state information.

Figure 3.3 shows the result of considering intra-cluster link quality in constructing node clusters. The obstructions cause poor link quality between two groups of nodes. As a result, separate clusters have been formed on either side of the obstructions. Without link quality assessment, one cluster might have formed, exhibiting poor coordination due to the loss of control messages. Later, we describe where link quality assessment fits into the general ESCORT scheme.

Designing actual link quality assessment algorithms is beyond the scope of this thesis's research. Hence, we assume the use of a technique proposed in [41], in which link quality is measured using packet delivery rate and signal-to-noise ratio measurements. The authors observe that in energy-constrained networks, where nodes save energy via radio deactivation, the quality of the wireless links are not known *a priori* to packet transmission. Thus, a low-cost initialization phase is used during which nodes periodically wake up to measure link quality.

3.2.1.2 Signal Strength

Another important metric for cluster formation is the distance between nodes. Inter-node distance can be estimated in various ways, perhaps the most intuitive

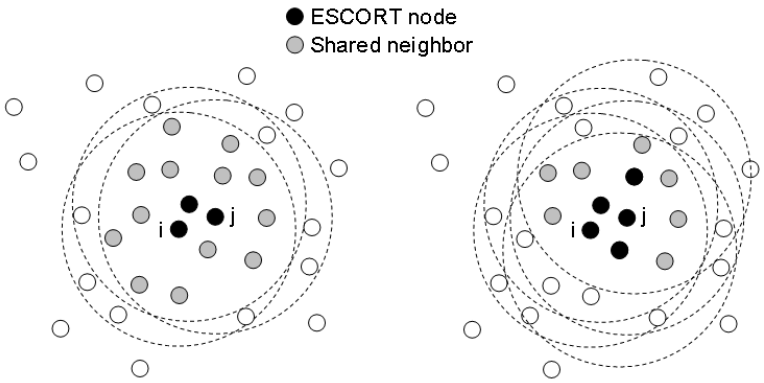


Figure 3.4: Comparative effect of using different signal strength thresholds on ESCORT cluster formation.

being the use of location coordinates. However, even though location coordinates could be used by the application layer, its use can be prohibitive in terms of energy and monetary costs. Therefore, we decide not to require its use. Instead, ESCORT uses the received signal strength (RSS) for distance estimation; hence, it is the second metric used for measuring signal quality. Other distance estimation methods are available (e.g., *time difference of arrival* and *angle of arrival*), but we adopt RSS for its relatively low computational over-head. For our purposes, we do not convert the actual signal strength reading into a distance measurement, since we only need a comparative measurement and not the absolute distance itself. We explain this later in Section 3.3.

Using RSS enables ESCORT to control the trade-off between energy savings and the network connectivity. Adding nodes to a cluster decreases its overall reachability. Figure 3.4 illustrates the difference between using smaller and larger RSS thresholds to form clusters. In the left cluster, nodes i and j cooperate in soliciting nearby nodes to join their cluster. We assume that all candidates exhibit acceptable link quality with both nodes i and j. With large RSS thresholds (such that nodes registering RSS measurements above a predefined value may join the cluster) only the closest neighbors gain membership. However, on the right, nodes i and j relax the RSS threshold, allowing the cluster to grow larger. As a result, the number of shared neighbors that are eligible for symmetric communication decreases. Since nodes' sleep times increase with cluster sizes, as does the number of hops required

for a packet to traverse a path in the network, the energy savings from sleeping are counter-weighted by the increased cost of traversing more hops in a path from a source to a destination. Even worse, an entire network's connectivity can be impacted if the only node connecting two parts becomes a member of a community.

One might assume that signal strength may also be used to predict packet loss behavior, thus eliminating the need for link quality assessment. However, in [82], Zhao and Govindan use empirical data to disprove the perceived strong correlation between signal strength and packet loss, finding that not all links with high RSS exhibit low packet loss. Thus, we do not rely solely on the use of RSS for measuring link quality.

3.3 The ESCORT Algorithm

3.3.1 Initialization

3.3.1.1 Signal Quality Assessment

Initialization begins with each node measuring the quality of its wireless links. As previously stated, we assume the use of a method such as that described in [41] to assess link quality. We propose combining RSS assessment with link quality assessment since it would be beneficial to measure links' signal strengths over multiple samples to obtain average RSS values for each link. We also assume that this sub-phase will account for variations in the environment. In ESCORT, neighbors exhibiting intolerable signal quality are pruned all together.

3.3.1.2 Topology Establishment

In this sub-phase, each node identifies an initial partner to begin cluster formation. Since multiple neighbors can potentially exhibit healthy link qualities, the neighbor exhibiting the highest link quality value is selected as the initial partner. We also do this in the interest of maintaining network connectivity in the unlikely case that the initial cluster can not further expand past the two initial nodes. A `JOIN_REQUEST` packet is sent to the potential partner and pairing is established when two nodes select each other as partners. The next step involves community expansion. For each initial node pair, i and j, the node with the highest ID (we assume

all nodes have unique numerical identifiers) is designated as the coordinator and thus takes responsibility for coordinating the cluster's expansion. For explanatory purposes, assume the coordinator to be node i. Included with each `JOIN_REQUEST` packet is the respective node's inner-neighbor set, A. Given i's full neighbor set, B_i, and a global link quality threshold, LQ_THRESH, A_i is defined as:

$$A_i = x : x \in B_i \; and \; LQ_{ix} > LQ\_THRESH \tag{3.1}$$

where LQ_{ix} is the link quality rating between i and x. For each node in A_i, i also maintains its average RSS value, $\overline{RSS_{ix}}$. Given a predefined global threshold, RSS_THRESH, i selects neighbors from both sets, A_i and A_j, to form a potential cluster set, PC, defined as:

$$PC = x : x \in A_i, A_j \; and \; \overline{RSS_{ix}}, \overline{RSS_{jx}} > RSS\_THRESH \tag{3.2}$$

where $\overline{RSS_{ix}}$ and $\overline{RSS_{jx}}$ are the $\overline{RSS}$ values of node x observed at i and j respectively. PC represents the set of candidates used for the cluster's expansion.

Continuing, the coordinator node solicits each node in the set PC to join its cluster by transmitting `JOIN_REQUEST2` packets with its ID included. In the event that multiple `JOIN_REQUEST2` packets are received, the $\overline{RSS}$ values of the requesting nodes are used as tie-breakers with the highest $\overline{RSS}$ value winning. Solicited nodes then send `JOIN_REPLY2` packets to their chosen coordinator and all nodes belonging to a particular cluster adopt a pseudo ID matching that of the coordinator node (the original ID is not discarded because of coordination requirements to be described later). This formation of one logical node allows transparent interaction between the network and ESCORT protocol layers since all nodes in a cluster are now receptive to a shared identity.

Routing protocol errors may occur if ESCORT produces ill-formed clusters. This is illustrated by the ill-formed cluster B in Figure 3.5, two of whose members i and j lie on either side of cluster A's communication boundary. Suppose during the employed routing protocol's path discovery phase, j's radio is active, while the rest of cluster B's radios are deactivated. Subsequently, A, perceived as one node

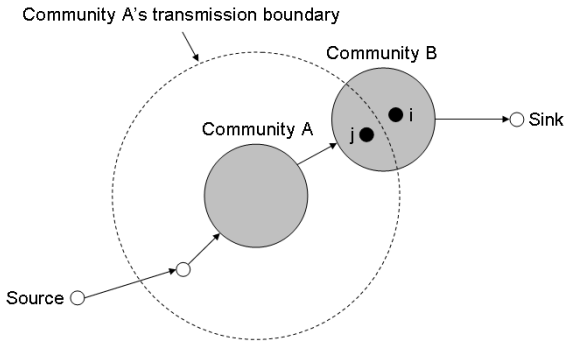


Figure 3.5: An ill-formed cluster causing potential packet loss.

to the network layer, arbitrarily selects B as its "next hop" on some constructed route, due to j's active state. However, at a later time, i, which lies outside of A's communication range, will become active and the route segment A-B will cause significant packet loss. Note also in the example illustrated in Figure 3.5 that community A may be replaced by a single node. Here, the previously described problem still exists. We prevent this scenario by utilizing the cluster members' individual neighbor sets constructed in the initialization phase.

Following a set delay after the previously described cluster formation phase, the intersection of all of the members' neighbor sets is calculated by each member. Afterward, each member knows the resultant reachability of the entire cluster, which is defined as the set of neighbors which all cluster members are able to directly communicate with. Additionally, a cluster coordinator will query each node in its intersected community's neighbor set to see if the node belongs to another cluster. If a node does belong to another cluster, the coordinator will obtain a list of the node's cluster members and and see if all these nodes are reachable by its own cluster. If the entire cluster is not reachable, the coordinator will remove this node from its neighbor list, as well as all other nodes which the entire community can not reach. The completed list is then broadcast to the rest of the community. As a final step, each cluster member will send an `IGNORE` packet to its excluded neighbor(s) indicating to the node that it should no longer consider it a valid neighbor. Also, if a coordinator finds that the intersection of its members' neighbors lists is empty, it will absolve its members of community membership, since no benefit can be gained.

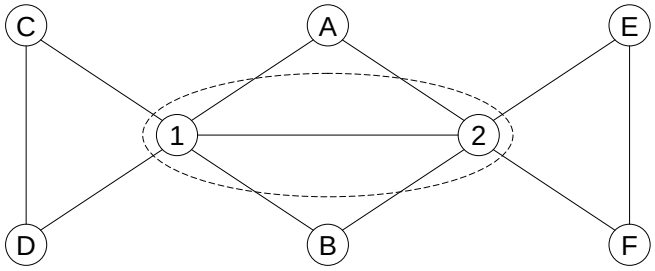


Figure 3.6: An example network illustrating how network segmentation may occur.

All of this helps establish well-formed communities that eliminate scenarios such as that shown in Figure 3.5 and significantly reduces the possibility of network segmentation.

We concede that the previously described neighbor selection routine does not entirely eliminate the creation of network segmentation. An example is shown in Figure 3.6 where it is assumed that nodes 1 and 2 have formed a community, indicated by the dashed circle. Following ESCORT's neighbor selection policy, nodes 1 and 2 are only allowed to communicate with nodes A and B. However, this will stop nodes C and D from communicating with nodes E and F, segmenting the network. This particular scenario is characterized by the inclusion of *articulation points*, which are nodes (or more generally, vertices in a graph) whose removal from a network cause the network to be disconnected. Such nodes are assumed to be a rarity for dense sensor networks, however their presence may still occur. A simply solution is to not allow such nodes to be members of ESCORT communities or be excluded as neighbors. Efficient algorithms for identifying articulation points have been well researched and proven to be effective (see [6]). Such algorithms can be used before ESCORT establishes communities in order to avoid possible network separation.

3.3.2 Runtime

ESCORT's runtime behavior is driven by *leader election* and *state sharing*. The leader node is the one that handles routing for its cluster during a given duty cycle. Routing protocol state-sharing is conducted between duty cycles to ensure

that new leaders route packets appropriately using updated information. To further prevent packet loss due to collisions, ESCORT's control messages are transmitted using a separate wireless channel from that used by the network layer.

3.3.2.1 Leader Election

Leader election is designed to be fault-tolerant and fair, promoting graceful network degradation. To balance the cluster's workload, the node with the most residual energy at the end of a duty cycle is chosen as the active node for the next cycle. The original coordinator node is the only node known to share high quality links with the rest of its cluster. Thus, it conducts leader election and state-sharing. This duty, coupled with normal routing duties, could potentially place an unfair burden on the coordinator nodes, so they are currently exempt from routing duties.

After the duty cycle time, T_{dc}, expires, all nodes enter the election state and send a `VOTE` packet, containing the node's residual energy, to the coordinator. The last active node also includes its energy dissipation rate over the last duty cycle. The coordinator then selects the node with the highest residual energy as the new leader and broadcasts its ID to the other nodes in the cluster.

Fairness is further achieved by the calculation of T_{dc}. So that all nodes dissipate energy at approximately the same rate, an exponential average is used to predict the dissipation rate in the next duty cycle based on past rates. Thus, if a node j wins the election, its predicted energy dissipation rate, DR_j, is calculated by the coordinator as:

$$DR_j = \alpha(DR_i) + \tau(1 - \alpha) \tag{3.3}$$

where DR_i is the energy dissipation rate of the last active node during the last cycle, τ stores the average over the history of operation, and α controls the responsiveness to recent history. T_{dc} is then calculated by:

$$T_{dc} = \frac{p \times IE_j}{DR_j} \tag{3.4}$$

where IE_j is the initial energy of node j and p is the predefined percentage of IE_j to be expended in the next duty cycle. T_{dc} is then broadcast to the entire community.

3.3.2.2 State Sharing

In the state sharing phase, the last active node sends the new leader its network layer state information (e.g., forwarding tables, counters, etc.) to maintain routing fidelity. We note that the last active node ignores any received packets on behalf of the cluster until the end of this phase. Any transient link failure imposed by this policy during the time state information is transmitted and received at the new leader should be negligible to performance. After the routing state is transferred, the new leader remains active while all other nodes sleep for the calculated duty cycle time.

3.4 Evaluation

In this section, we present an analysis of ESCORT's performance using the SENSE simulator [20], which has been implemented using the C++ programming language. We compare various performance metrics of a wireless ad hoc sensor network with and without ESCORT applied. In our study, we analyze the following performance metrics:

- *average packet delivery rate*: the average percent of total `DATA` packets successfully delivered from source to destination;
- *average packet delay*: the average end-to-end time incurred for `DATA` packet delivery;
- *average sleep rate*: the average percent of total simulation time a node spends sleeping;
- *average energy consumed per node*: the average percent of initial energy consumed by a node throughout the simulation;
- *network lifetime*: the average time needed for 70% of the path nodes[1] to drain their batteries.

[1]We only consider nodes belonging to current routes since ESCORT affects their performance the most

Network layer	AODV
MAC layer	IEEE 802.11
Signal propagation model	Free space
Transmission power consumption	75mW
Receiving power consumption	24mW
Idle power consumption	81μW
Terrain size	$800 \times 400\text{m}^2$
Network size	30 to 80 nodes (increments of 10), 5 sources, 1 destination
Transmission range	250m and 300m
Simulation time	50,000 seconds
α (see Equation 3.3)	0.9
p (see equation 3.4)	0.001

Table 3.1: Simulation parameters for ESCORT's evaluation.

Most of the simulation parameters that we used for our evaluation are shown in Table 3.1. All nodes were randomly placed in the terrain. For most experiments, nodes start with an initial energy of 1×10^4J. For the experiments in which we measure network lifetime, we change the initial energy to 1×10^3J in order to decrease potentially long simulation times. We used a bursty traffic model in which sources periodically generated data at 1 packet per second for random short time intervals. All tests were repeated ten times using different random number generated seed values.

We altered the SENSE channel component to model physical obstructions. Obstructions were placed randomly near the center of the terrain so as to maintain a variety of routes between the sources and destination, which are positioned on opposite sides of the terrain. For this evaluation, we assumed that any significant line-of-site obstructions rendered the wireless link unsuitable for communication. ESCORT uses these links to prune out undesireable neighbors.

3.4.1 Topology Characteristics

Figure 3.7 illustrates how ESCORT's effect on network topology increases along with network density. Figure 3.7 specifically shows that for a larger transmission range, ESCORT's effect on the network is more persistent as the network density increases, solidifying the potential for the network to save energy. For a

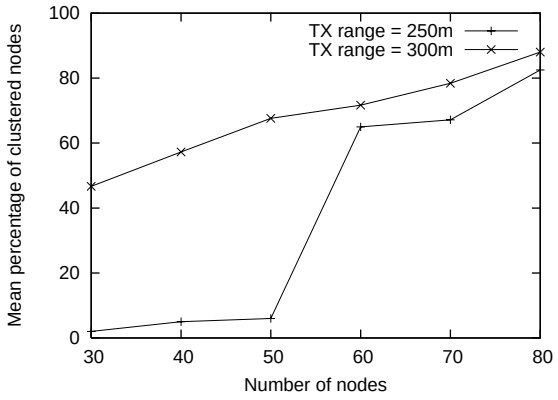


Figure 3.7: Percentage of nodes clustered by ESCORT.

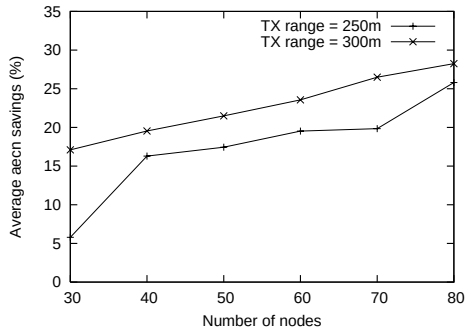
smaller transmission range, ESCORT displays a similar advantage only after a particular threshold, lying somewhere between a population of 50 and 60 nodes. These results support the idea that ESCORT is especially beneficial for those networks with larger node densities. We note that the RSS_THRESH values were adjusted so as to allow communities to grow larger along with the population size. This is because at smaller populations, larger communities risk having little or no neighbors to route to.

3.4.2 Energy Savings

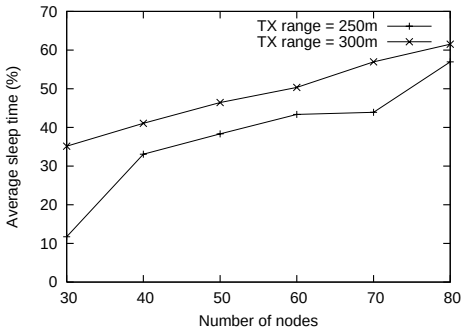
We assess energy savings using two measurements. First, we inspect the *average energy consumption per node (aecn). aecn* is adopted from [76] and is defined as follows:

$$aecn = \frac{E_0 - E_t}{n \times t} \tag{3.5}$$

where E_0 and E_t is the initial energy and the residual energy at time t (where t is the end of simulation), respectively, for n total nodes. Figure 3.8(a) illustrates the average aecn savings achievable with ESCORT, showing up to 25% reduction for the 250m case and 28% reduction for the 300m case. The second metric we use is the *sleep rate*. Figure 3.8(b) shows that ESCORT allows nodes to sleep up to more than 55% of the time for the 250m case and more than 60% of the time for the 300m case. According to these results, ESCORT allows a significant amount of sleep time. However, the savings in energy consumption may not appear as

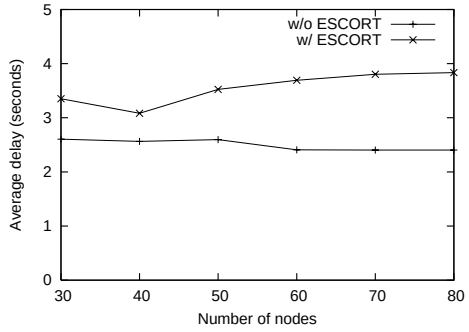


(a) Average aecn savings.

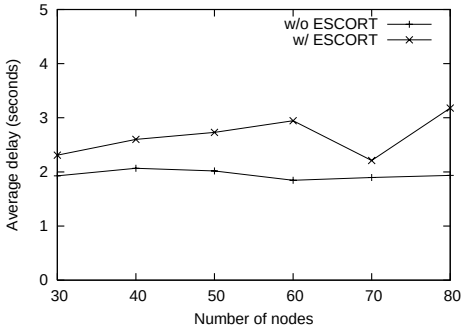


(b) Average percentage of sleep time.

Figure 3.8: ESCORT's energy performance.



(a) Average `DATA` packet delay for TX range = 250m.



(b) Average `DATA` packet delay for TX range = 300m.

Figure 3.9: ESCORT's packet delivery performance.

impressive in comparison. To explain this, we note that the amount of a node's energy consumption is not solely reliant on the time spent in sleeping and active states. Energy consumption is also affected by the power characteristics of the sensor node circuitry, particularly that regarding the node's radio. Hence, we focus on the induced sleep rates shown in Figure 3.8(b) to highlight the significant benefit of using ESCORT.

ESCORT also contributed an improvement regarding *network lifetime*. For the 250m case, ESCORT achieved up to an approximate 38% (at 80 nodes) increase in network lifetime over the case using just AODV. For the 300m case, a similar increase was achieved with 36% at 80 nodes. All of these results highlight the significance of ESCORT's energy savings that are essential for prolonged WSN operation.

3.4.3 Packet Delivery Performance

ESCORT's energy-efficiency is achieved without impacting the packet delivery rate, which remained at about 99% across all experiments. ESCORT's impact on end-to-end delay is shown in Figure 3.9(a) and Figure 3.9(b). The additional delay introduced by ESCORT is no more than approximately 1.3 seconds for the cases of 250m and 300m transmission ranges. This additional delay grows slowly with the increase of node population. In general, we expect ESCORT to impose additional delay upon end-to-end communication. This is primarily because the resultant transmission range of an ESCORT cluster is smaller than that of a non-clustered node. This is a direct result of the ESCORT cluster formation process, during which neighbors are pruned due to signal quality assessment and the intersection of individual nodes' neighbor sets. It is clear that since ESCORT reduces transmission range, it will take more hops for a source to communicate with a destination and hence, the delay will be increased. This phenomenon is a general feature among most topology-control algorithms.

3.5 Concluding Remarks

ESCORT shows that WSN energy savings can be achieved by simply using signal quality assessment as opposed to using location-tracking technology such as GPS. This is beneficial since augmenting a WSN with a system such as GPS may be cost-prohibitive. However, ESCORT is not without its own challenges. In this study, we have assumed an ideal operating environment with symmetric links. We acknowledge that this assumption may not always be true, especially in industrial settings where multiple sources of interference may unpredictably affect the quality of the wireless signal and in turn affect ESCORT's cluster formation phase. ESCORT can easily be made to repeat the signal quality assessment and cluster formation phases at any point to account for changes in the environment or the network (e.g., changes caused by sensor node faults). However, this could be a costly process depending on how often re-configuration needs to occur. Therefore, we propose that ESCORT be used in stable environments where the nature of wireless signal propagation would not largely affect ESCORT's cluster formation process.

Such environments are feasible and may be found in applications such as remote agricultural and habitat monitoring, where it is reasonable to expect a lower probability of signal interference.

CHAPTER 4
Self-Healing Routing for Wireless Sensor Networks

4.1 Introduction

This chapter describes *(SHR) Self-Healing Routing* [21][22][73], which represents this thesis's research on fault-tolerant and energy-efficient WSN routing. SHR is a cost-based receiver-decided protocol in which nodes do not maintain knowledge of neighbors' states to make routing decisions. Instead, packets are freely broadcast to all neighbors and they autonomously decide whether to forward packets further toward a destination based on some cost metric. In our case, SHR selects the node with the smallest hop distance from the destination to forward a packet. This concept is illustrated in Figure 4.1, where the gradient of the network's plane represents the decreasing cost of forwarding a packet toward the destination. Note that of all the nodes that received the packet (indicated by the arrows), only the one with the lowest cost forwards it further[2].

In general, SHR uses a prioritized transmission back-off delay scheme to facilitate the *self-selection* of the next forwarding node. This scheme (1) prioritizes the selection of nodes assigning the highest probability of selection to those nodes that offer the shortest routes, (2) enables the implicit (and efficient) acknowledgment of the selection of those nodes, and (3) enables local route repair for increased

[2]For simplicity, arrows to the previous sender are omitted.

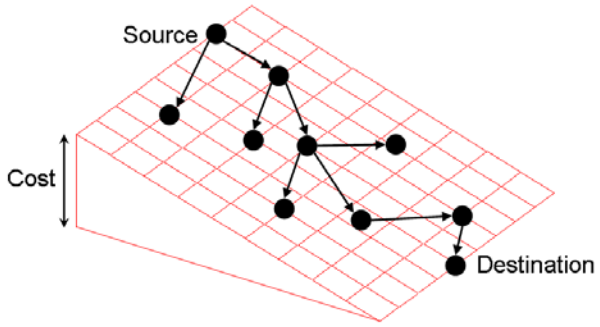


Figure 4.1: Simplified example of SHR's routing behavior.

fault-tolerant operation. The result is a WSN routing algorithm that (1) exhibits resilient packet delivery performance under less-than-ideal network conditions, (2) imposes low message-passing overhead, and (3) naturally accommodates energy-efficient topology-control algorithms such as that described in Chapter 3.

In this chapter, we describe two variations of the SHR routing protocol: the first, *SHR-M*, being more simple and the second, simply titled *SHR*, being more adaptive and employing a local route repair algorithm. Each protocol exhibits different benefits depending on given network conditions. First, we describe how both protocols establish hop distances for all nodes. Second, we describe the details of how SHR-M and SHR forward data to destinations. Third, we describe some beneficial properties of the protocols and fourth, we evaluate the performance of the protocols in comparison to AODV using simulation.

Before continuing, we state some fundamental assumptions regarding the SHR protocols. First, all wireless links are symmetric. Asymmetric links may negatively affect the efficiency, but not the correctness of the protocols. However, we note that some of the testing scenarios presented in this chapter do cause link asymmetry. Second, nodes are stationary, not mobile. Additional assumptions are stated as needed.

4.2 The SHR-M and SHR Protocols

The data structure used by the SHR protocols is fairly simple. Each node maintains a cost table with entries consisting of the following items:

- the *identity* of a target node (which may either be a source or destination),
- the *sequence number* of the last packet observed from the target node,
- the *hop distance* from the target to the current node.

Additionally, each node maintains a sequence number which helps identify the uniqueness of the packets it disseminates into the network.

4.2.1 Topology Establishment Phase

SHR-M and SHR employ an initial topology establishment phase in order for nodes to learn their hop distance from a given destination such as a base station. Similar to other WSN routing protocols, the SHR protocols' route discovery process starts with a route request phase. When a source wants to send data to a destination for which there is no cost table entry, it broadcasts a route request (`RREQ`) packet via a flooding protocol and then increases its own sequence number by 1. Each `RREQ` packet contains the following items:

- the *identity* of the source node,
- the *identity* of the destination node (which will often be a commonly known base station),
- a *sequence number* to distinguish the packet from the other `RREQ` packets originating from the same source,
- an *actual hop count field* that records the number of hops that the packet traveled from the source to the current receiving node.

When a source sends the `RREQ` packet, it will initialize the packet's actual hop count field to 1 and increment its own sequence number by 1.

If an intermediate node receives a `RREQ` packet from a source for which there is no entry in its cost table, this node creates a new entry with the source ID, sequence number, and actual hop count fields. Otherwise, if an entry for the source already exists, the entry is updated either if the `RREQ` packet's sequence number is higher than that in the table, or, in case of equality, when the actual hop count is lower than the stored hop distance for the source. In either case, the hop distance is updated with the value in the `RREQ` packet. If the first case occurs, the sequence number in the table is also updated. The intermediate node then increments the `RREQ` packet's actual hop count field by 1 and attempts to forward the packet. It will first set a random back-off timer upon the expiration of which the packet will be forwarded. This is to avoid collision with other nodes that received the same packet. If the node receives another packet with the same source id, sequence number, and actual

hop count before the back-off timer expires, it simply cancels the timer and does not relay the packet as this simply means that another node forwarded the packet.

The destination node, upon receiving a new `RREQ` packet, will reply with a route reply (`RREP`) packet, which contains the same fields as those of the `RREQ` packet. However, the packet will be addressed to the source which originally sent the `RREQ` packet. The dissemination of `RREP` packets progresses much in the same manner as that of `RREQ` packets. The only difference is that, for consistency, a node must have an entry in its cost table for the `RREP` packet's destination in order for it to rebroadcast the packet.

4.2.2 SHR-M Data Dissemination Phase

As previously mentioned, SHR-M is meant to be a more simplified variant of the routing protocol SHR. SHR-M uses only one type of packet for data dissemination, the `DATA` packet. The `DATA` packet contains the following items:

- the *identity* of the source node,
- the *identity* of the destination node,
- a *sequence number* to distinguish the packet from the other `DATA` packets originating from the same source,
- an *actual hop count field* that records the number of hops that the packet traveled from the source to the current receiving node,
- an *expected hop count field* that indicates the shortest number of additional hops a packet should travel to reach a destination,
- a data payload.

When the source has a `DATA` packet to send, it initializes the packet's actual hop count field to 1 and the expected hop count field to the stored distance in its own cost table for the respective destination. The packet's sequence number is also initialized using the source's current sequence number. Then, the source freely broadcasts the packet out over the wireless medium. Upon receiving a copy

of the `DATA` packet, a node will first determine if it is closer to the destination than the packet's sender by comparing its own hop distance in its cost table to the expected hop distance contained in the packet. If the node is not closer, it will drop the packet. If the node is closer to the destination than the sender, it will set a transmission timer for a period uniformly randomly selected between 0 and λ, which is a tuning parameter to be chosen by the application. The larger λ is, the greater the total communication delay between the source and destination. This delay generally shortens as λ becomes smaller; however, the probability of collisions increases as a trade-off. When the timer expires, the node will rebroadcast the packet, only it will increment the actual hop count field by 1 and change the expected hop count field to its own distance from the destination. On the other hand, if a node overhears a broadcast of the same copy of the `DATA` packet (determined by source id, destination id, and sequence number) before its timer expires, it will cancel its own transmission (and timer) to reduce the dissemination of redundant packets. Note that this is not guaranteed to completely eliminate redundant traffic, as nodes waiting to forward the same packet may be too far away from each other to overhear each other's transmissions. However, this feature may well be preferred in some scenarios. Continuing, after a node receives a packet with a given sequence number, it will ignore all further packets with the same sequence number for the same source-destination pair. This prevents uneccessary traffic and possible routing loops in the network. This entire process repeats at each hop until the `DATA` packet reaches its destination. A final rule is that when a node overhears any packet that is broadcast with an expected hop count of more than one lower than its own, the node will change its own hop distance (for the corresponding destination) to the packet's expected hop count plus 1. This follows the assumption that a node's neighbors are either one hop greater, one hop lower, or of equal hop distance from a destination. Hence, this operation is an adaptive feature in an attempt to continuously route packets along the shortest known path. The same behavior is also used in SHR, which is explained next.

4.2.3 SHR Data Dissemination Phase

SHR's data dissemination process is more adaptive and generally more cautious than SHR-M's. The `DATA` packet fields are identical to those used in SHR-M, however, there is the addition of a *maximum hop count* field. If a `DATA` packet's actual hop count exceeds the maximum hop count, it will be dropped. The use of both fields is primarily motivated by SHR's additional route repair operation which will be explained in full detail shortly. In SHR, nodes also maintain an additional field for each entry in their cost tables, *ignore count*. This field, whose purpose will be explained later, is initialized to 0.

When a node broadcasts a `DATA` packet (after updating its header fields just as in SHR-M), only nodes that are closer to the packet's destination AND have an ignore count (corresponding to the destination) of 0 will be allowed to respond by starting a transmission timer for a period uniformally randomly selected between 0 and λ. If the timer expires, the node will rebroadcast the packet in the same manner as SHR-M, but it will start a second timer with a delay uniformally randomly selected between 1.25λ and 1.75λ. This timer, referred to as r-timer, is used for retransmission purposes and its associated behavior will be described momentarily. Continuing the explanation of the first timer, if the node overhears another node rebroadcast the `DATA` packet before its timer expires, it will decide that another node has forwarded the packet and subsequently cancel the first timer to reduce transmission of redundant data. It will also ignore all further `DATA` packets that have the same sequence number and identical or lower expected hop counts for the same source-destination pair.

When the node that sent the packet (or equivalently, forwarded the packet due to the expiration of its first timer), and subsequently started r-timer, overhears its `DATA` packet forwarded by a downstream node (indicated by the packet's expected hop count being lower than its own distance from the destination) before r-timer expires, it will cancel r-timer since it knows that the packet was successfully forwarded. However, as described in SHR-M, multiple nodes waiting to transmit the `DATA` packet may not be within listening range of each other and hence, redundant transmissions may occur at any given hop. Therefore, if the sender overhears *an-*

other downstream node forward the same `DATA` packet, it will know that there are nodes outside of listening range of each other and that redundant traffic can occur. In this case, the sender will broadcast an *acknowledgment* (`ACK`) packet. Hence, if a node receives an `ACK` packet (which contains the related `DATA` packet's source id, destination id, and sequence number) before its first timer has expired, it will drop the packet it is waiting to transmit and cancel its timer. This explicitly signals all *competing* nodes that another node (outside their listening distance) has forwarded the packet. Additionally, upon the event of any of these nodes receiving an `ACK` and causing a timer cancellation, the node itself will set its ignore count to a maximum value which can be tuned by the application. For the next x number of sequential `DATA` packets (i.e., with x being the value of the ignore count and applied for those with increasing sequence numbers) associated with the same source and destination, the node will decrement ignore count by 1, drop these packets immediately, and not participate in the forwarding process. Additionally, the node will ignore all further `DATA` packets with the *current* sequence number and identical or lower hop counts for the current source-destination pair. As indicated previously, when ignore count reaches 0, the node will again compete to forward a packet from the sender. Overall, this limits the frequency at which a packet's sender needs to transmit `ACK` packets and hence, saves some energy. This entire logic markedly separates SHR from SHR-M. The primary benefit of the `ACK` packet is that it reduces both traffic congestion and energy consumption due to redundant traffic. Reducing congestion is especially beneficial for denser networks since it reduces the probability of collisions. Reducing energy consumption is a universal benefit for any sensor network. SHR is adaptive since it does not aggressively use `ACK` packets, but only does so when they are needed. Again, this behavior may be less suitable for certain scenarios and our simulation results will clarify when each should most likely be used.

Now we describe what happens if r-timer instead expires. In SHR, this event signifies that the packet was not successfully forwarded by the next downstream node. This may happen for multiple reasons such as collisions, transient links, or faulty/inactive downstream nodes. We note that the range of the r-timer's delay values were chosen such that the sender has adequate time to allow for (1) the

wireless transmission of the `DATA` packet to the next node, (2) a small amount of processing time at the next node, and (3) the maximum delay before the next node broadcasts a `DATA` packet. When r-timer expires, the sender will retransmit the `DATA` packet and reset r-timer. This time, if the sender receives notification that the packet was forwarded (i.e., by hearing a `DATA` packet with a lower expected hop count), it will cancel r-timer. However, if r-timer expires again, the node will assume that there is no neighbor available that is closer to the destination than itself and enter the *route repair* phase.

4.2.3.1 Route Repair

The route repair phase adjusts nodes hop distances from a destination such that further packets flow around the point in the network at which the route is severed and also find the next shortest route. First, the node that unsuccessfully retransmitted its `DATA` packet will increase its own hop distance by 2. If the new distance plus the packet's actual hop count value does not exceed the packet's maximum hop count value, then it will rebroadcast the `DATA` packet, this time with a new expected hop count value reflecting the node's new distance from the destination. The packet's increased expected hop count value will allow other nodes to participate in the election, specifically allowing packets to travel laterally (i.e., to neighbors with an equal hop distance to the destination) or backwards (i.e., back toward the packet's source).

The route repair process is illustrated in Figure 4.2. Suppose node C has left the topology and node B has a packet from a source node (SRC in the figure) to send to the destination (DST) as shown in Figure 4.2(a). This will cause node B to increase its hop distance to 4 (see Figure 4.2(b)) and rebroadcast the packet. Next, node A will receive the packet and since none of its neighbors will have lower hop distances (causing a route repair operation), A will increase its hop distance to 5 (see Figure 4.2(c)) and rebroadcast the packet. Now, either node B or D will try to rebroadcast the packet, since their hop distances are lower than node A's. We assume that they both rebroadcast the packet since, in this scenario, node A never had to broadcast an `ACK` packet to send any node into an ignore state. The

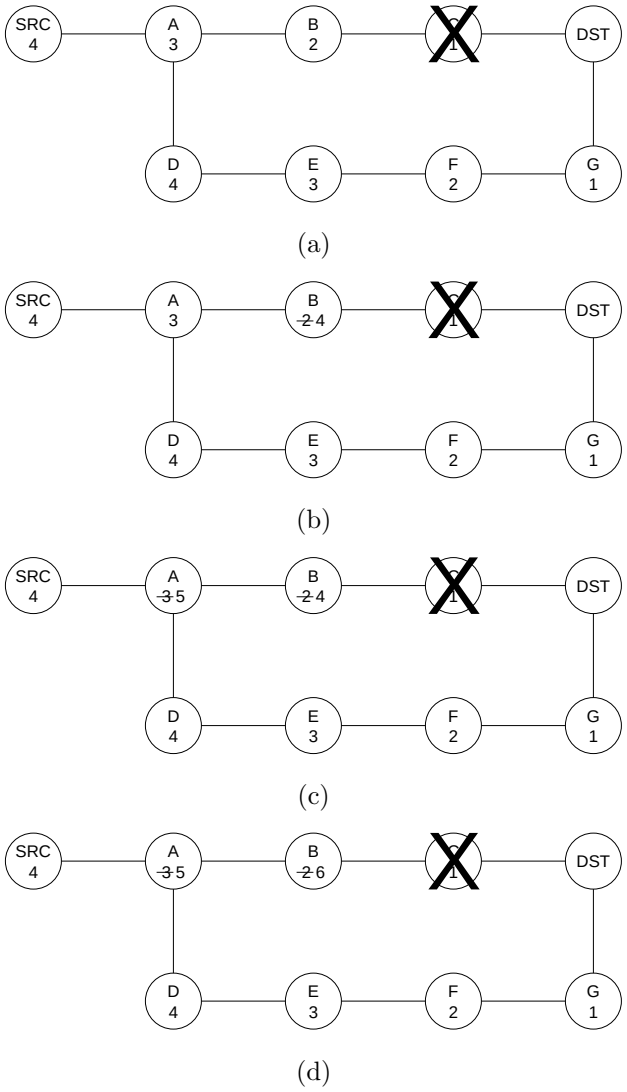


Figure 4.2: SHR route repair scenario.

resultant state of the network is shown in Figure 4.2(d). Node D will not change its hop distance and will successfully deliver the packet to the destination while node B will again increase its hop distance after another route repair operation. As SRC tries to deliver more packets to DST, its hop distance will eventually increase to 6; this is not shown in Figure 4.2.

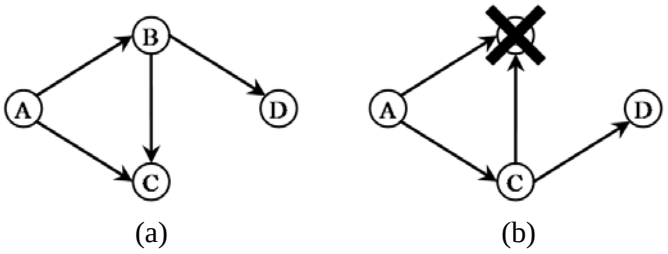


Figure 4.3: Scenario showing that packets immediately seek an alternative route when a node in the previous route fails.

4.2.4 The SHR Protocols' Properties

A strong feature of SHR-M and SHR is that they do not maintain explicit routes, so there is no need to constantly monitor the routes' connectivity. Hence, as opposed to most traditional routing protocols, SHR-M and SHR can handle most node or link failures without incurring any control packet overhead. Also, with traditional protocols, if a node on the route wants to go to sleep, it must explicitly inform its neighbors and pass them the task of relaying packets, as in ESCORT [11]. It is even more troublesome when a node or a link suddenly goes down because it is difficult to quickly distinguish a temporary fault from a permanent one. Furthermore, a substantial amount of time may elapse before the nature of the fault is discovered. In contrast, in SHR-M, when a node or link fails, other nodes will self-select themselves to quickly form a new route; the transition is seamless and no extra actions are needed. As a result, any node, even if it is on the route, can freely switch to the sleep or standby mode to save energy, making SHR-M well suited for energy limited WSNs.

Figure 4.3 illustrates a scenario where packets can immediately seek an alternative route after a node in the previous route fails. In Figure 4.3(a), node B is an intermediary on the route from node A to D. However, if node B is deactivated, either passively by a sudden failure or actively by itself in order to conserve energy, the next packet would naturally travel through node C, since node B would no longer be involved in the self-selection process. This is reflected in Figure 4.3(b). The transition from a path going through node B to a path going thought node C is seamless, and does not require extra control packets.

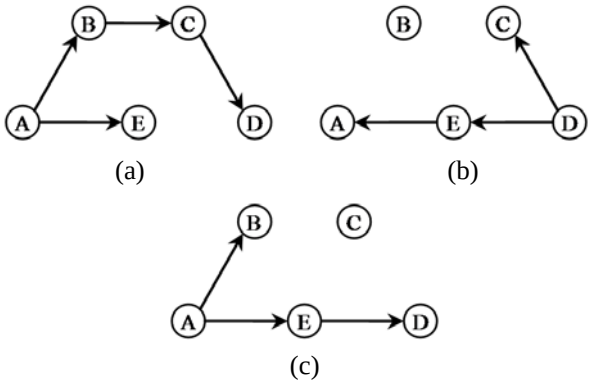


Figure 4.4: Scenario showing that SHR-M and SHR are capable of constantly looking for and switching to shorter paths.

In SHR-M and SHR, `DATA` packets and `DREP` packets always carry the most current information about the distance from the originating node. Hence, the protocols can often consistently choose the shortest routes to the destination. In other routing protocols, such information could also be made available to intermediate nodes. However, for packets to find the shortest routes, constant route changes would be required in those protocols, and the overhead of excessive route maintenance would likely offset the benefit. It is the SHR protocols' ability to effortlessly handle topology changes that makes it capable of always looking for the shortest paths as well. Figure 4.4 shows such an example. At first (Figure 4.4(a)), the route between nodes A and D is passing through nodes B and C. Although there is a shorter route via node E, these nodes are not aware of it because node E is excluded from the path discovery phase either by (1) randomness of delays, or (2) by the recent arrival of node E in this neighborhood. If the communication continues to flow one way from A to D, then node E would never get a chance to know that it is within 1 hop of node D. However, as soon as node D transmits a packet, node E immediately updates D's entry in its cost table. After that, both nodes E and C will compete for the next hop, and E will win because its distance to node A is just 1. The next time node A sends a packet to node D, node E will self-select itself for transmission, which effectively shortens the route between nodes A and D by 1. This example indicates that in order for SHR-M and SHR to choose the shortest routes, the communication must be bidirectional, otherwise nodes in shorter routes

would not able to update their cost table. This requirement is not too difficult to meet in practice, since many WSN applications operate based on a query-response model which requires bidirectional communication.

4.3 Evaluation

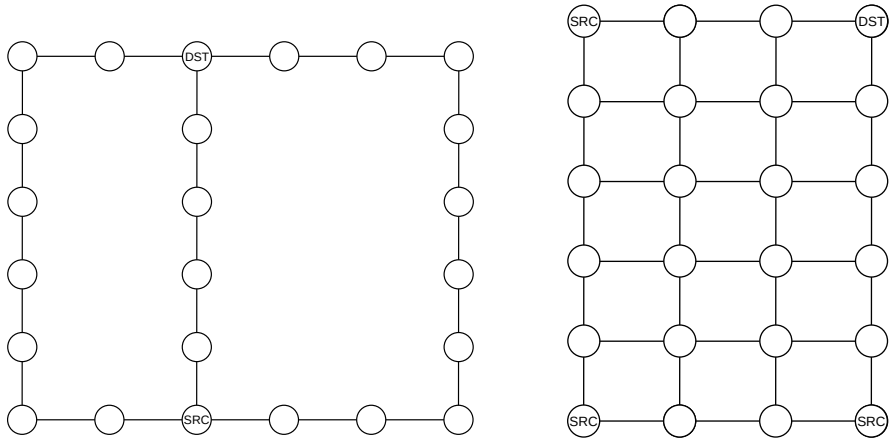
In this section, we present an analysis of the SHR protocols' performance using the SENSE WSN simulator [20]. We used two different sets of tests to evaluate protocol performance. The first set tests the protocols on three small-scale WSN topologies. The second set tests the protocols on a large-scale WSN with a randomly generated network topology.

4.3.1 Small-Scale Tests

Originally, some small-scale tests were used to enable easy debugging of the protocols' operation. However, we realize that certain WSN applications may make use of small-scale networks where the number of nodes will not number in the hundreds (as is often stated in the literature). Hence, we included some small-scale tests in this dissertation. While we do not include any formal hardware tests here, we do make use of some observations from our early experiences in operating actual sensor network hardware, which consisted of Crossbow®MICAz sensor motes (the hardware specifications for those motes are presented in Chapter 1). For the simulator, our aim was to use a realistic single hop packet reception rate to simulate node-to-node communication. In our preliminary tests using hardware, we observed that nodes with a transmission power of 0dBm on an artificial turf field were able to achieve an average packet reception rate of approximately 90% when separated by 5m (the nodes were placed directly on the ground). We accept this value an as adequate representation of a network's average quality of communication in a favorable environment (i.e., with a low amount of human activity) and for a small-scale deployment. Hence, we use this value in our small-scale simulations. We also observed that approximately 5% of the time, a node's transmitted packet reached beyond its closest neighbor and was successfully received at a node that was two hops, or 10m, away. This could be due to any number of factors such as hardware

`DATA` packet size	29bytes
`DATA` packet traffic rate	1 packet every 5s
λ	22ms
Simulation time	810 seconds

Table 4.1: Simulation parameters for small-scale tests.



(a) Route repair topology. (b) Grid topology.

Figure 4.5: Network topologies used for small-scale tests.

irregularity. Hence, we used this observation in our simulations as well.

Table 4.1 lists the simulation parameters that were used for the small-scale tests. All simulation results represent an average of ten runs using different random number seed values. Two network topologies were also used to enable a keener analysis of the protocols' performance. The network topologies are illustrated in Figure 4.5.

The *route repair* topology (Figure 4.5(a)) offers three unequal disjoint routes between a source and destination: one short, one medium, and one long. This is specifically used to evaluate the performance of the protocols when no redundancy (i.e., when multiple nodes able to forward a copy of the same packet) is present. During the simulation, two adjacent nodes in the mid-length route are deactivated to see how the protocols react. The *grid* topology (Figure 4.5(b)) shows the effect that multiple traffic sources may have on the protocols' performance. Multiple sources increase traffic density as well as overall traffic rate, both increasing the probability

	SHR-M (1 hop)	SHR-M (2 hop)	SHR (1 hop)	SHR (2 hop)
Delivery rate (%)	54.5	2.67	66.1	48.8
End-to-end delay (ms)	58.4	25.9	129.6	148.1
Avg. hop count	5.8	4.74	5.96	5.72

Table 4.2: Small-scale test performance results for route repair topology.

	SHR-M (1 hop)	SHR-M (2 hop)	SHR (1 hop)	SHR (2 hop)
Delivery rate (%)	85.2	6.58	84.4	75.3
End-to-end delay (ms)	31.6	16.7	66.2	82.2
Avg. hop count	4.32	2.65	4.69	4.48

Table 4.3: Small-scale test performance results for grid topology.

of packet collisions.

The results of the small-scale tests are shown in Tables 4.2 (route repair topology) and 4.3 (grid topology). For the columns labeled *<protocol name> (1 hop)*, the results only for a 90% one hop packet reception rate are listed. The columns labeled *<protocol name> (2 hop)* list the results for when the additional 5% two hop packet reception rate is used. The values for the average end-to-end packet delivery rate, end-to-end packet delivery delay, and length of route traveled from source to destination are shown.

Comparing the tables shows that the route repair network topology yields the weakest results. This is generally because no amount of redundancy exists in the network, causing a large amount of retransmissions and route repair operations. However, the retransmissions and route repair operation do help SHR outperform SHR-M for this topology. While these adaptive features grant SHR better packet delivery rates, they do increase end-to-end delays, which is correlated with a greater amount of hops needed for packets to reach the destination. Results for the *2-hop* tests are noticeably weaker and it was expected. If a node overhears a response from a node two hops further downstream, it will decrease its own hop distance by one. At this point, 95% of the time, the node will not be able to forward any `DATA` packets further downstream because its one-hop neighbor that was previously defined as being closer to the destination (i.e., having a lower hop distance) will now behave as if it is the *same* distance to the destination. This is very detrimental for SHR-M, but SHR's adaptive features significantly alleviate this problem.

Performance results for the grid topology are significantly better, clearly indicating that the SHR protocols are better suited for networks with a sufficient degree of density and redundancy. When a packet has no chance of directly traveling two hops ahead, SHR-M is clearly the better protocol. Its minimalist approach to forwarding packets slightly increases the packet delivery rate, but significantly reduces the end-to-end delay by more than half that of SHR. However, for the second case in which a packet can occasionally jump two hops ahead, SHR's adaptive qualities once again prove beneficial, yielding a much larger packet delivery rate than that of SHR-M.

These small-scale tests confirm that both SHR-M and SHR perform better on denser networks. However, either protocol may provide a better routing solution depending on how packets tend to disseminate throughout the network. If a network designer can guarantee that packets will never occasionally travel two hops beyond their senders (perhaps by using precise tuning of transmission powers and placement of nodes), then SHR-M would a better protocol to use. This is especially true if the data to be collected is time-sensitive. On the other hand, if the application can tolerate slower data delivery, then SHR would be the best protocol since it is more robust. We also note that the performance of SHR may be further improved if it is allowed attempt to retransmit packets a greater number of times before executing its route repair routine.

4.3.2 Large-Scale Tests

Large-scale tests were used to evaluate the performance of the protocols in a dense setting with randomly generated network topologies. Table 4.4 lists the simulation parameters that were used to generate results for this series of tests. Both SHR-M and SHR were compared against the AODV wireless routing protocol. AODV was chosen since it is highly representative of traditional unicast routing protocols (see Chapter 2) and its implementation is well documented [61]. Furthermore, no significant tunable parameters can affect its behavior. Hence, it serves as a good standard benchmark for testing.

We compared the protocols' performance under multiple network conditions.

`DATA` packet size	1000bytes
`DATA` packet traffic rate	1 packet every 40s
λ	100ms
Max. ignore count value	9
`DATA` packet's max. hop count	(Source's expected hop distance) + log_2(source's expected hop distance)
Terrain size	$(2000 * 2000)\text{m}^2$
Network size	500 nodes (varied)
Transmission range	250m
Signal propagation model	Freespace

Table 4.4: Simulation parameters for large-scale tests.

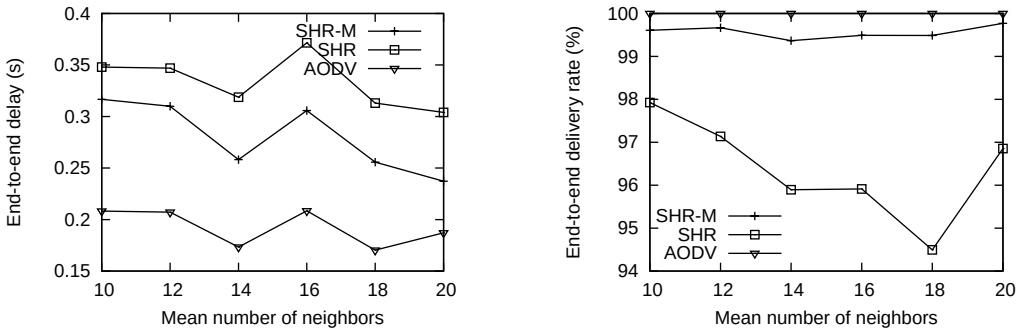
First, the density of the network was varied such that each node had an average number of neighbors ranging from 10 to 20 nodes. For these tests, 10 pairs of sources and destinations were used to generate network traffic. Second, the number of source-destination pairs was varied between 2 and 20. This helps evaluate how the protocols perform with an increasing amount of network traffic. We also note that communication between sources and destination was bidirectional. Second, only one destination was used while the number of traffic sources was varied between 2 and 20. We used this test since most sensor network configurations are known to employ only one destination, or base station. Again, communication was bidirectional. The prior three sets of tests were run for 3000s. Fourth, the transient failure rate for the network was adjusted between 0 and 30%. A transient failure can be used to model error-prone wireless links, power management induced duty cycles (similar to that used in ESCORT), excessive packet collisions, or simply hardware malfunctions. Hence, performance results corresponding to this test set are an indicator of how SHR-M and SHR will perform when a topology-control algorithm is employed to save energy. A transient failure is implemented by having a node assigned a mean active time and a mean sleep time. The sum of these two times was fixed at 200s. The time spent in each mode was distributed exponentially about the mean value. The failure rate indicates what percentage of the network's nodes will assume this behavior. Fifth, the permanent failure rate for the network was adjusted between 0 and 30%. This simulates what happens when a node dies in the network and does not return to the topology. For this test set, each node had an x% chance

of failing, where x is the permanent failure rate. Nodes that were selected to fail started their failure time uniformly distributed among the remaining simulation time. Simulations corresponding to node failures lasted for 6000s to gain a better understanding of how the protocols behaved.

As in the small-scale tests, we collected data describing the average packet delivery rate, end-to-end delay of successfully delivered packets, and number of hops traveled from a source to a destination. Additionally, we collected the total number of transmitted MAC packets and average number of packets transmitted per hop. The first metric indicates the communication overhead required for a given protocol to achieve its packet delivery performance. This metric includes all the packets sent by the MAC protocol layer, e.g., retransmissions and `RTS`, `CTS`, and `ACK` packets sent independently of the network layer in addition to packets under control of the network layer. We note that AODV's unicast MAC layer (based on IEEE 802.11) utilizes additional `RTS`, `CTS`, and `ACK` packets while the more simple broadcast MAC used by SHR-M and SHR does not use such MAC-level packets. Continuing, the second metric, average number of packets transmitted per hop, supports the observation of communication overhead as well, but isolates the observation to a *per-flow* basis. The number of transmissions per hop was calculated by dividing the number of total transmitted MAC packets by the number of *useful* hops. We define a useful hop as one that causes the first arrival of a `DATA` packet at its destination. This calculation helps account for the delivery of redundant `DATA` packets. Note that packets that never arrive at a destination incur no useful hops. Again, all simulation results represent an average of ten runs using different random number seed values.

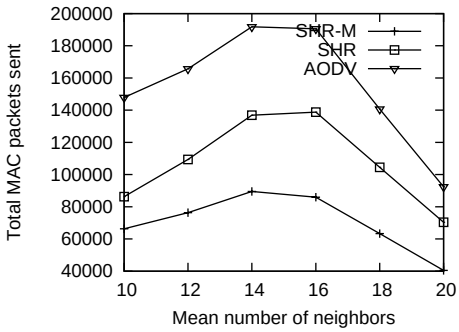
4.3.2.1 Effect of Network Density

Increasing the number of neighbors adds redundancy to the network, a feature that, as observed earlier, SHR-M and SHR exploit very well. However, the probability of collisions may increase as more neighbors compete to forward packets from a sender. Increasing density also increases the probability that a node will travel physically farther at each hop, this is reflected in Figure 4.6(a). As density increases, packet delivery delay for all protocols generally decreases. AODV has the lowest

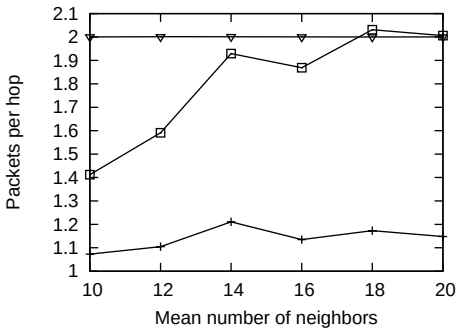


(a) Average end-to-end delay.

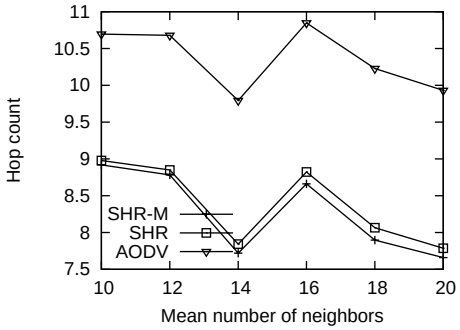
(b) Average delivery rate.



(c) Average total MAC packets sent.



(d) Average packets transmitted per hop.



(e) Average route length.

Figure 4.6: Large-scale test performance results versus network density.

delay, which is expected since it does not impose a transmission back-off delay at each hop in the route. For an average density of 16 neighbors per node, the delay increases for all protocols, most likely indicating the generation of a challenging and anomalous network topology for all protocols. This topology somehow forces longer routes, as indicated for the same density value in Figure 4.6(e). After this data point, the delay generally decreases. SHR-M exhibits the largest delay since it makes no provisions to reduce redundant traffic, which consequently increases

network congestion and slows down traffic. In observing Figure 4.6(b), SHR-M's behavior is more beneficial than SHR's regarding the packet delivery rate, which is nearly equivalent to the rate of AODV. This is consistent with the results obtained for the small-scale tests. The results shown in Figure 4.6(c) are interesting. As network density increases, the total number of MAC packets increases and then decreases. Again, this loosely correlates to the activity of the plots mentioned earlier, further indicating that the topologies generated do not induce monotonically changing routing behavior. AODV requires the highest number of packets, reinforcing the thought that its unicast nature makes it less efficient than SHR-M and SHR. SHR requires more communication overhead than SHR-M due its occasional use of `ACK` packets. So while SHR-M has a higher probability of causing redundant traffic streams, SHR's more conservative behavior actually causes more communication overhead. Considering all the previous observations, Figure 4.6(d) shows expected behavior. AODV needs two transmissions per hop: the application's `DATA` packet and the MAC layer's `ACK` packet. `RTS` and `CTS` packets are not used since these packets are used only if the `DATA` packet's size exceeds a certain threshold, which it does not in these tests. SHR-M requires slightly above one transmission per hop. One might think that SHR-M should consistently exhibit one transmission per hop. However, using the previously described method to calculate this value yields a value above one. This is why we used this method to account for traffic redundancy. SHR produces an increasing number of packet transmissions per hop as network density increases. This is due to more packet retransmissions and possible route repair operations. Last, according to Figure 4.6(e), AODV produces the longest routes. This is because AODV does not adapt to continuously changing topologies by searching for the shortest available routes to a destination.

4.3.2.2 Effect of Network Traffic Rate

The performance results for increasing the number of source-destination pairs are shown in Figure 4.7. Again, AODV exhibits the lowest delay since it does not employ transmission back-off delay. SHR's delay is the highest due to its corrective overhead. As before, SHR-M's packet delivery rate is fairly close to that of AODV,

but decreases by a negligible amount as traffic increases. This is because the effect of SHR-M's allowance of redundant traffic becomes exasperated as the ambient network traffic level increases. However, this behavior still bests that of SHR, which exhibits the lowest packet delivery rate. According to Figure 4.7(c), the comparison of the total MAC packets sent by the protocols per source-destination pair is similar as in the previous test set. The only difference is that the number of packets increases as traffic increases, which is expected since more sources are generating packets. There are points at which the number of MAC packets actually decreases, and these points are the same for all protocols. We attribute this to anomalous data points in the simulation runs that disturb the overall data trends as the number of source-destination pairs increase. The data shown in Figure 4.7(d) is similar to that previously shown in Figure 4.6(d), and the same explanation applies. As traffic increases, causing more packet collisions, SHR responds by using more retransmissions and repair operations. We note that even though SHR eventually requires more packets per hop than AODV, AODV requires a larger number of MAC packets. This is because to repair a route, AODV starts an entirely new topology establishment phase with `RREQ` and `RREP` packets, where SHR's route repair phase uses less packets and only uses `DATA` packets. Since Figure 4.7(d) only reflects the transmission of packets directly involved with data forwarding (i.e., `DATA` and `ACK` packets), the full magnitude of AODV's repair operations is not represented. The activity shown in Figure 4.7(e) is fairly similar to that in Figure 4.6(e) as well and the same explanation applies.

Figure 4.8 shows protocol performance when the number of sources is increased while only one destination is employed. Generally, as shown in Figure 4.8(a) the delay for all protocols exceeds their delays associated with multiple source-destination pairs. This is because this particular traffic model creates an area of congestion near the base station, which noticeably increases resultant delay due to more channel contention and collisions. Again, SHR has the largest delay, which markedly increases along with the number of traffic sources. On the other hand, SHR-M exhibits more static behavior. Interestingly enough, AODV's delay decreases along with increasing traffic. There is no clear explanation for this behavior. We attribute it to the

existence of more source nodes physically closer to the destination, effectively lowering the average delay. This benefit has no effect on SHR-M and SHR. Inspecting Figure 4.8(b), AODV and SHR-M still exhibit the best packet delivery rates. SHR's rate decreases as in the previous test set, but remains above 90%, which is tolerable for most WSN applications. Figure 4.8(c) shows that AODV and SHR approach a nearly equivalent number of transmitted MAC packets per source as the number of traffic sources increases. The initial number of MAC packets transmitted per source node decrease sharply since, especially for AODV, a single route establishment or repair operation benefits other sources' packet delivery behavior since the entire network easily becomes configured to route packets to the single destination; multiple route establishment and repair operations are not needed. Hence the repair operations become more efficient and lower the value for this statistic. In general, this behavior is unique for this test set because traffic streams are closer together and less disjoint than for a network with separate source-destination pairs. However, after a certain point, this benefit fades and the number of MAC packets again increase as the traffic increases. Figure 4.8(d) shows that SHR's packets per hop increases along with the traffic, which is again expected as more corrective actions are needed. Figure 4.8(e) shows that while AODV exhibits the largest number of traveled hops, this value decreases along with an increasing number of sources. Again, we attribute this to more sources being closer to destination and hence lowering the average value; similar behavior is reflected in SHR-M's data.

4.3.2.3 Effect of Network Failure Rate

The performance results for increasing network failure rates are shown in Figures 4.9 (transient failures) and 4.10 (permanent failures). Figure 4.9 shows markedly different behavior between the AODV and SHR protocols. For instance, Figure 4.9(a) shows that as the transient failure rate increases, AODV's delay significantly increases to well above 2s. On the other hand, both SHR-M and SHR maintain much lower delays while SHR-M's delay is nearly constant for all data points. This highlights SHR-M's and SHR's ability to sustain tolerable delay in the midst of increasing network faults. Figure 4.9(b) shows that both SHR-M and

SHR do not perform as well as AODV regarding packet delivery rate, with SHR performing better than SHR-M due to its route repair routine. However, the results in Figure 4.9(b) must be compared with those in Figure 4.9(c) in order to make a keen observation. While AODV exhibits a better packet delivery rate, it requires more than the order of magnitude number of MAC packets of SHR and SHR-M to achieve this performance. Again, this is due to AODV's inefficient route repair behavior which causes many broadcast floods throughout the simulation. SHR can easily be adjusted to allow more retransmission attempts before resorting to route repair and match the performance of AODV, yet it would still require less packets than AODV. This shows the clear benefit of SHR. The data in Figures 4.9(d) and 4.9(e) compliment the data describing the total number of MAC packets sent.

Finally, we examine the behavior of the protocols as the network experiences permanent node failures. Figure 4.10(a) illustrates data that is somewhat similar to that associated with increasing traffic rates. SHR again exhibits the highest delay while AODV exhibits the lowest delay, with the exception of one outlying data point at a permanent failure rate of 45%. However, we do note that SHR-M's delay remains relatively stable while AODV's increases along with the permanent failure rate. SHR does not exhibit as strong performance regarding packet delivery rate though. Figure 4.10(b) shows that SHR's performance dramatically decrease as failures increase, but again highlights the benefit of SHR's route repair algorithm. Again, Figures 4.10(c), 4.10(d), and 4.10(e) reveal that while AODV exhibits better packet delivery performance, it does so at a higher cost of communication overhead.

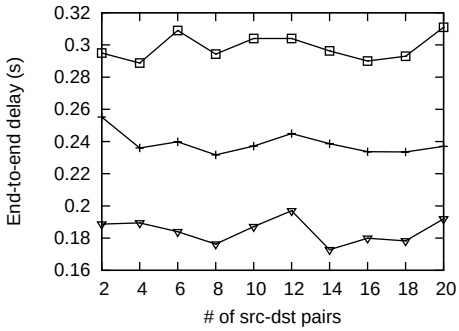
4.4 Concluding Remarks

This chapter describes SHR and SHR-M and identifies their respective benefits in different networking scenarios. We can conclude that for settings in which there exists a negligible amount of failures, SHR-M provides a good communication solution since its performance is better than SHR's, yet more efficient than AODV's. On the other hand, when failures are present (a phenomenon that is expected in wireless sensor networks), SHR provides a much better solution, however at the cost of a slightly higher communication overhead than SHR-M. Regardless,

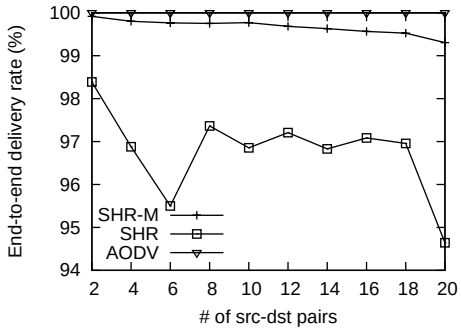
the communication overhead is still much less than that required by AODV. Also, according to the earlier small-scale tests, SHR is beneficial when there may exist some slight radio power fluctuation. Overall, this shows the novelty and benefit of the SHR protocols developed in this thesis study. The SHR protocols also differ from other works described in Chapter 2 since it does not require GPS or an additional RTS-CTS hand-shaking routine to facilitate packet forwarding. Comparatively, it is rather lightweight and still performs effectively. In the last chapter, we describe future research directions that can make SHR even more effective.

We did not conduct a theoretic study of how parameters such as λ and the number of times the retransmission timer should be invoked affects routing protocol performance. In a previous work, we did theoretically describe how using slotted back-off timers reduces the probability of collisions [14]. However, a further theoretical study of how the previously mentioned parameters affect delay, delivery rate, collisions, and energy usage would be helpful. These types of studies have been conducted for more traditional unicast WSN routing protocols, but to the best of our knowledge, have not been conducted extensively for broadcast-based opportunistic protocols such as SHR. However, we also believe that more hardware-based, as opposed to simulation-based, studies should be conducted in order to evaluate the real-world usefulness of opportunistic routing protocols. Our future work includes these types of studies.

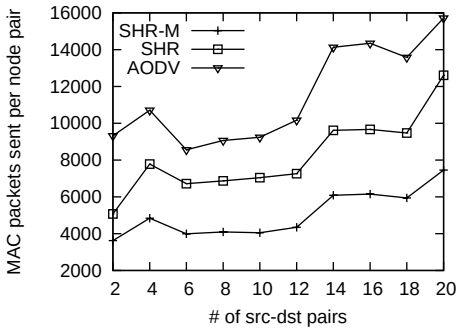
We also summarize the conditions under which the SHR protocols are most applicable. First, SHR and SHR-M do not apply to data-centric query-based networks. There is no way to use these protocols to query for data with specific properties. Currently, one can only query a node by location or network address. Second, SHR and SHR-M do require a large percentage of symmetric links and large amount of network density to deliver their indicated benefits. Both protocols will operate fine in sparse conditions, but they were designed for networks with large density all the same. This is not an unreasonable assumption since for many hardware platforms, the range of sensing is much smaller than the range of communication. Hence, large network density often must be enforced to guarantee adequate sensor coverage.



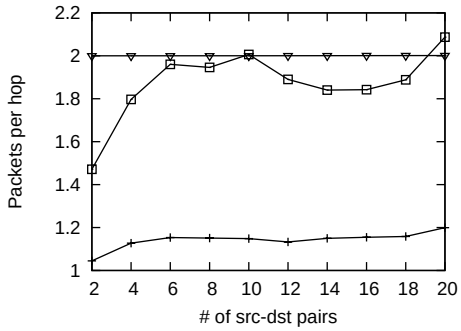
(a) Average end-to-end delay.



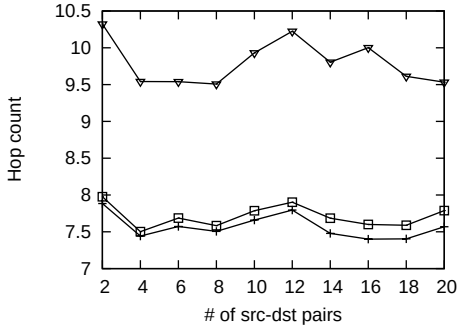
(b) Average delivery rate.



(c) Average total MAC packets sent.

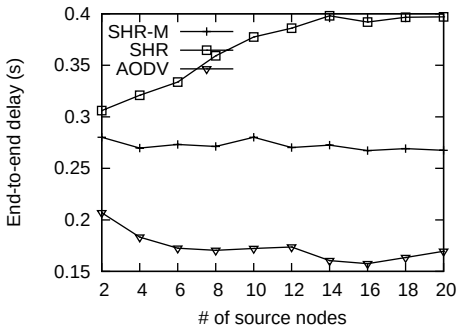


(d) Average packets transmitted per hop.

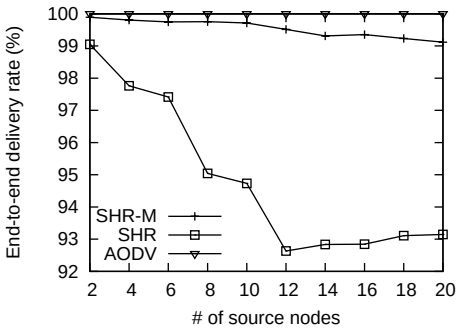


(e) Average route length.

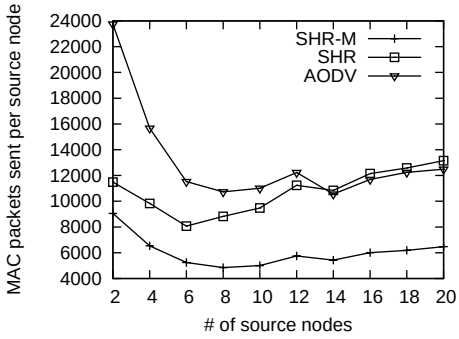
Figure 4.7: Large-scale test performance results versus number of source-destination pairs.



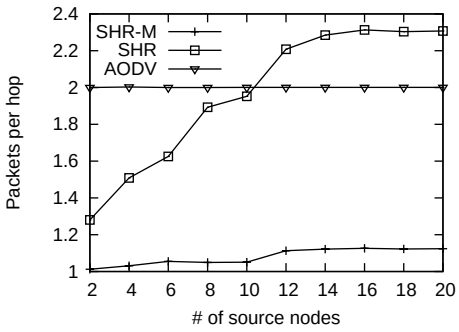
(a) Average end-to-end delay.



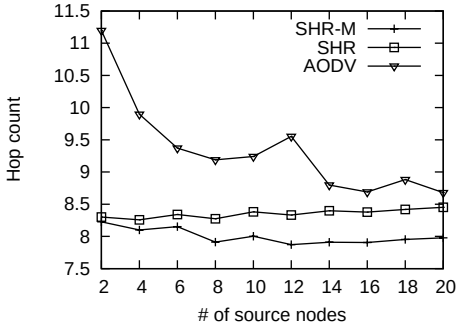
(b) Average delivery rate.



(c) Average total MAC packets sent.

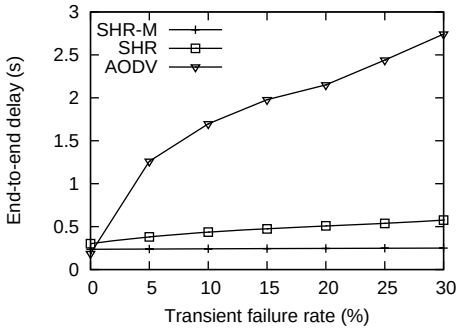


(d) Average packets transmitted per hop.

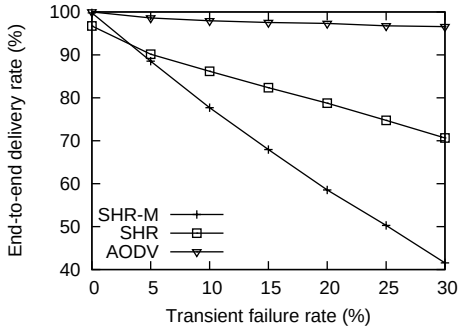


(e) Average route length.

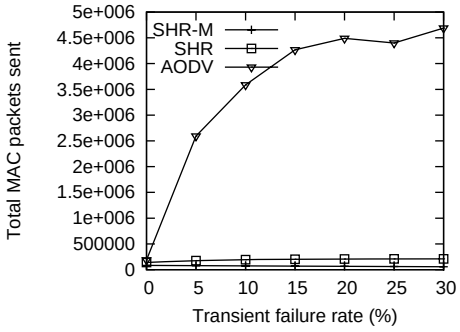
Figure 4.8: Large-scale test performance results versus number of sources (one destination).



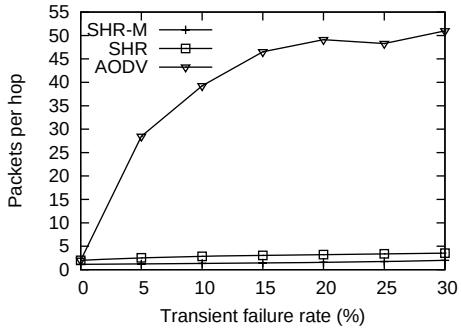
(a) Average end-to-end delay.



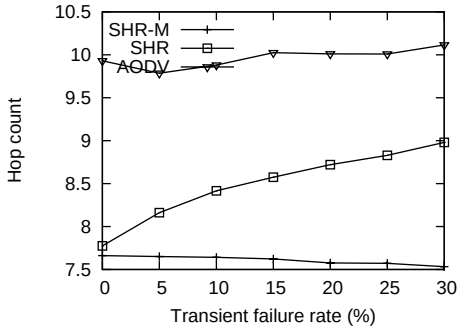
(b) Average delivery rate.



(c) Average total MAC packets sent.

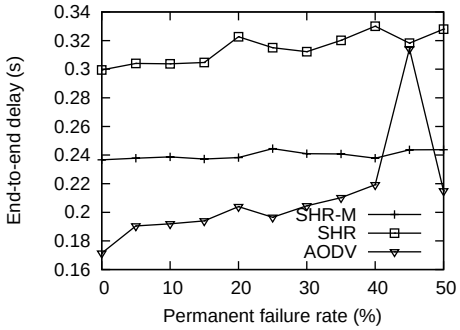


(d) Average packets transmitted per hop.

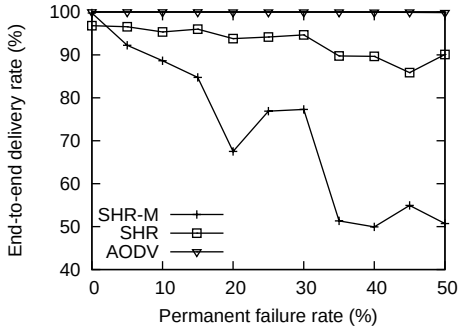


(e) Average route length.

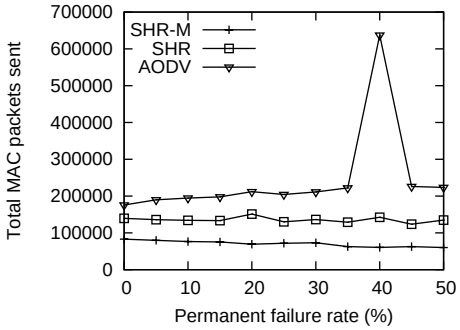
Figure 4.9: Large-scale test performance results versus transient failure rate.



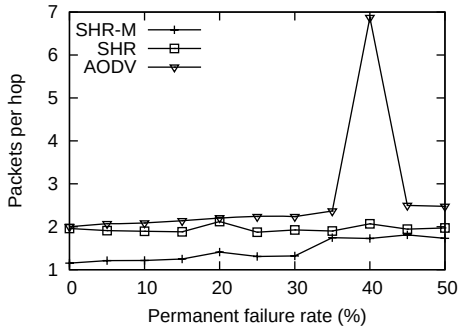
(a) Average end-to-end delay.



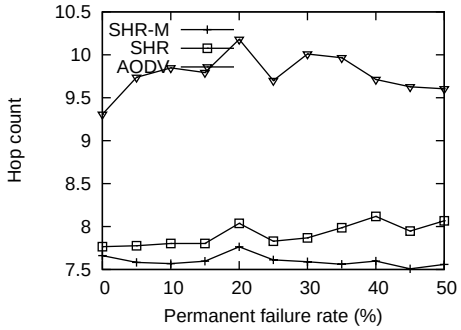
(b) Average delivery rate.



(c) Average total MAC packets sent.



(d) Average packets transmitted per hop.



(e) Average route length.

Figure 4.10: Large-scale test performance results versus permanent failure rate.

CHAPTER 5
Sentire: A Middleware Framework for Sensor and Actuator Networks

5.1 Introduction

This chapter describes *Sentire* [12][13]. The Sentire framework facilitates building extensible high-level application middleware that uses SANETs for environmental control. Sentire addresses the process of building such a middleware, rather than addressing a specific *instantiation* of a SANET middleware optimized for some specific purpose. Hence, the Sentire framework does not preclude the use of existing sensor middleware. Sentire has been implemented using the Java programming language.

Sentire's driving goal is to ease the process of developing middleware by providing the following contributions:

- a high-level framework consisting of extensible and reusable components that (1) facilitate the flow of queries, commands, and data between applications and underlying SANETs and (2) partitions middleware development into logically related sub-tasks;
- programmable middleware primitives that shield solution developers away from intimate details of interacting with underlying SANETs and provide scalable control over underlying sensors and actuators;
- a lightweight budgeting and auction-based policy for coordinating a set of actuators that are distributed among different Sentire middleware instantiations and execute interfering tasks.

Figure 5.1 illustrates our view of the functionality of SANET middleware and the place where it logically resides in relation to SANETs. We acknowledge that an application layer is not depicted in Figure 5.1; if it were, it would be placed to the left of the middleware layer. This is because the scope of our research is focused

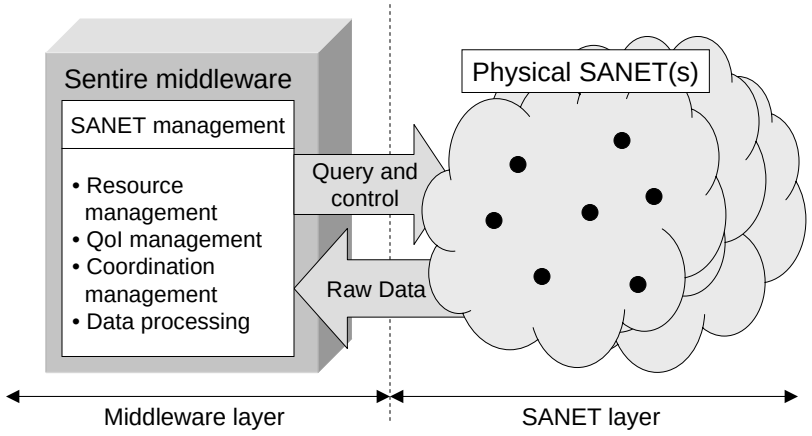


Figure 5.1: SANET system layers.

on the middleware layer and its interaction with underlying SANET resources. We explain in Chapter 6 that future research will involve a close investigation of how Sentire will *formally* interface with applications.

In this chapter, we describe the details of the Sentire middleware framework and how it can be used to compose a sense-and-respond system. This specifically includes a description of Sentire's communication architecture and various manager components. We also describe two usage cases, one in which Sentire is used to compose an example sense-and-respond system and the other in which the framework is used to coordinate actuators in a simulated distributed climate control system.

A major assumption regarding our research in Sentire is that we do not assume the use of a specific SANET platform (e.g., TinyOS, National Instruments, etc.). Sentire focuses on supporting and abstracting essential "development composition tasks" and not providing standardized interfaces with underlying platforms. We describe this in more detail, as well as state any further assumptions, as the chapter progresses.

5.2 Sentire Messaging

Before delving into the details of Sentire's communication facilitates, we give a short overview of Sentire's physical configuration. Figure 5.2 describes where Sentire resides in relation to sensors and actuators. We envision the primary Sentire devel-

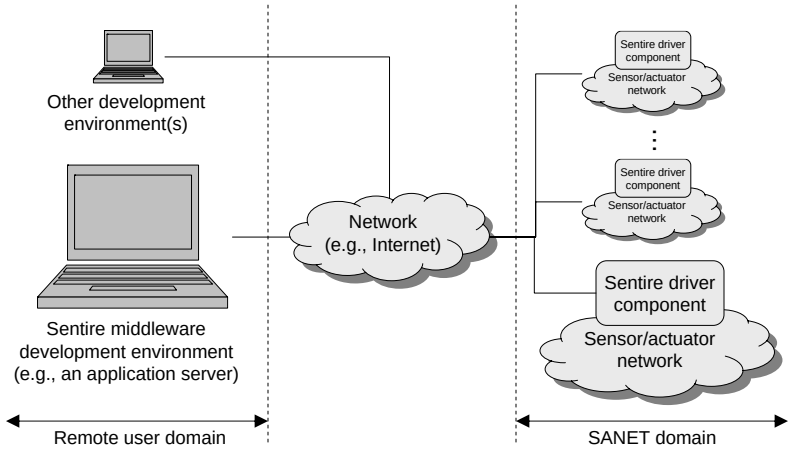


Figure 5.2: Physical configuration of the Sentire development framework.

opment environment (i.e., Java runtime with Sentire library) residing close to the middleware developer, who will most likely be in a remote location from the actual sensors and actuators to be used for development. However, there is no reason why the development environment can not reside directly within the SANET domain. Communication with SANETs or other Sentire instantiations is implemented using Java network sockets. This was chosen primarily for convenience, as sockets are a fundamental and well-understood element of Java communication. Figure 5.2 also reveals the existence of a Sentire driver component. This serves as a translator between the language of Sentire and that of the attached sensors and actuators. The decision to place this component in the SANET domain was motivated by one of the core goals of our research regarding middleware development, to abstract away cumbersome details of how to interface with a specific underlying SANET platform. Therefore, our research assumes the availability of a Sentire driver component on the server (e.g., base station) of a sensor or actuator network that the developer wants to integrate into his or her system.

Sentire uses a publish-subscribe messaging architecture to provide communication between manager components. This is implemented using the Java `EventListener` interface for multi-threaded applications. As mentioned previously, communication between the Sentire middleware and remote SANETs is supported using Java sockets. Sentire employs one message base type, `SentireMessage`, which is a base

format used to implement many different message types. The `SentireMessage` uses the following header fields:

- *destination*: the Sentire manager type to which this message is addressed to;
- *message type*: defined as a `QUERY`, `COMMAND`, `DATA`, `RESOURCE`, or `BID` (whose functions will be described throughout the chapter);
- *data type*: describes the type of sensor data requested if the message type is a query;
- *query type*: describes the type of query (i.e., *one-time* query, *subscription*, or *threshold*) if the message type is a query (query types will be described more later);
- *query ID*: numerical identifier used if this message is a query;
- *query accept bit*: indicates if a query has been rejected or accepted by Sentire;
- *action type*: describes the type of action requested if the message type is a command for an actuator;
- *action strength*: indicates the strength of actuation needed if the message type is a command;
- *action direction*: indicates in what direction actuation should occur (e.g., point actuator right, left, etc.) if the message type is a command;
- *location*: describes the location(s) of the sensor data requested or location at which the action requested;
- *bid amount*: amount of bid for a resource if the message is a bid.

The `SentireMessage` also has a data payload, which is implemented using XML and the Document Object Model (DOM) [28][26]. The `SentireMessage` format was designed as such because of the following. `SentireMessage` headers enable the middleware to route messages to the appropriate managers for further processing.

The usage of XML DOM affords the middleware developer both a flexible and standardized way of specifying, storing, and reading message contents. As we will explain further, a `SentireMessage`'s contents can be used to store data for processing (e.g., sensor data) or to further define the purpose of the message itself. Also, using XML is beneficial since it is a standard and well-used format for processing documents having structured information. Therefore, its inclusion promotes Sentire's formal connectivity with other external applications in the future.

5.3 Sentire Managers

The core of Sentire's framework is its collection of manager components. Sentire managers serve two primary purposes: (1) to facilitate the flow of information between Sentire and SANETs *and* between multiple Sentire instantiations, and (2) to organize the development tasks of SANET middleware. The six following manager types define the core of the Sentire framework: *External Interface*, *Resource*, *Sensor*, *Actuator*, *Data*, and *Coordination*. Figure 5.3 illustrates how these managers are connected, the fundamental data that is passed between them, and what data is passed between Sentire and underlying sensors and actuators. Figure 5.3 represents sensors and actuators as clouds, lending some ambiguity as to the configuration of their connectivity. We state clearly that this cloud may represent a network *or* a single device. Noting this distinction is important for describing the details of the Sensor and Actuator Managers later. For the remainder of this chapter, we will often refer to either a sensor device or network simply as a *sensor* (or simply *actuator* for actuator networks) to avoid confusion. Continuing, Sentire managers are implemented as Java abstract classes that implement the `Runnable` interface for multi-threaded operation. The remainder of this section describes the details of each manager, including what types of SANET abstractions, programming primitives, and level of extensibility each manager offers.

5.3.1 External Interface Manager

The External Interface Manager (EIM) serves as the interface for external sensors and actuators and other Sentire instantiations to send messages to the Sentire

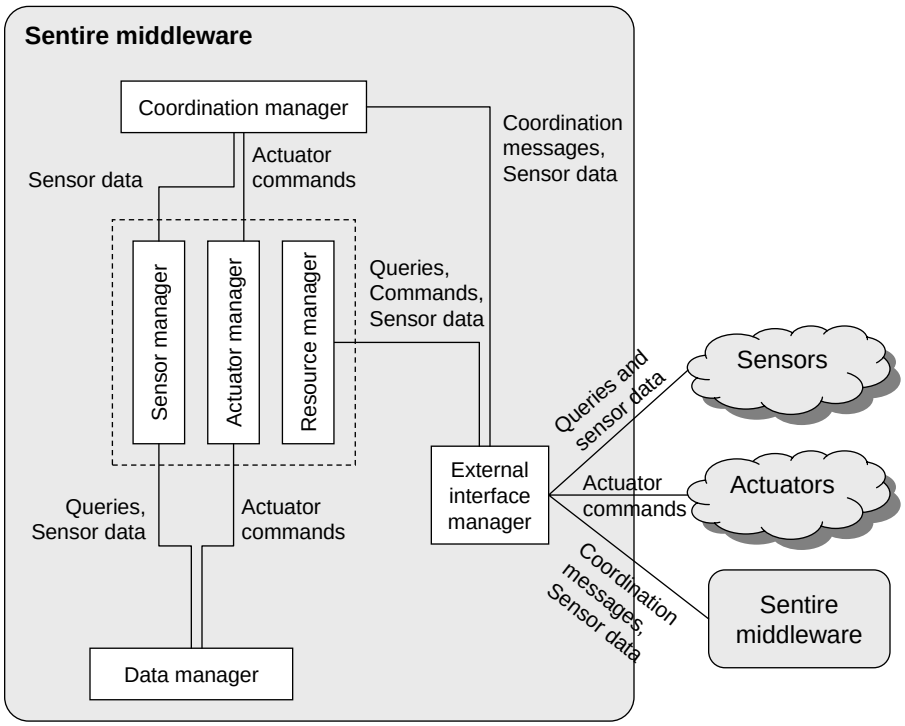


Figure 5.3: Data flow in the Sentire framework.

middleware. The EIM can expect three types of messages: `DATA`, `RESOURCE`, and `BID`. Most `DATA` messages arrive from sensors and hence, are sent to the Sensor Manager, as indicated by the message's destination field. However, some `DATA` messages may also arrive from other Sentire instantiations. This is associated with Sentire's support for distributed actuator coordination. These `DATA` messages will be sent to the Coordination Manager.

5.3.2 Resource Manager

The Resource Manager (RM) is responsible for associating all sensors and actuators with Sentire and policing Sentire's access to those entities. Involving association, the RM maintains a list of an entries (or abstractions) for each sensor or actuator that is available for use by Sentire. Each entry consists of the following information:

- *data type*: describes what type(s) of data a sensor can produce;

- *action type*: describes what type(s) of action an actuator can perform on the environment;
- *location*: describes the physical location(s) in the environment a sensor can sense or an actuator can influence;
- the *socket address* and *port* to which Sentire can connect to send messages to SANETs;
- the *current level of available resource(s)* associated with the sensor or actuator.

All communication from Sentire to SANETs happens via the RM while the opposite flow happens via the EIM. This design decision was chosen to simplify any issues with multiple threads (or our case, managers) reading and writing to and from the same sockets.

The *current level of available resource* is used to describe things such as the estimated remaining lifetime of the network, remaining treatment resources available for actuation, or perhaps available network bandwidth. This information is periodically updated via `RESOURCE` messages from SANETs and enables the RM to control SANET usage via developer-defined policies. Sentire currently stores only residual energy information; adding space for other information is trivial. We prefer updating these fields from the SANET side in order to avoid implementing any programming logic to pause Sentire operation while waiting to retrieve resource information. We acknowledge that SANET-side updating may result in periodically outdated resource information, but we assume that the frequency of updates will be adequate for most application purposes, or at least will increase in frequency as critical resources become more scarce. All queries and commands that are issued to available sensors and actuators arrive at the RM before being sent external of Sentire. The RM enables a developer to write his own code to decide to accept or reject messages addressed to a particular sensor or actuator based on the level of currently available resources. This grants a developer the flexibility to determine how available resources should guide the decision-making of the middleware and hence, also guide application behavior. This also helps organize the middleware de-

velopment process by allowing a developer to manage resource usage independently of other aspects of the middleware.

5.3.3 Sensor and Actuator Managers

In this section, we focus most of our discussion on the operation of the Sensor Manager and then cover the significant differences regarding the Actuator Manager since many aspects of both managers are similar. The Sensor Manager's (SM) two primary responsibilities are to link sensor queries with the most appropriate sensor(s) and to provide a programming space where query decomposition can occur. All queries for sensor data are sent to the SM. Ideally, these queries would arrive from an overlying application, but for our purposes, all queries are currently issued from the Data Manager, which will be explained in further detail later.

When a query arrives at the SM, Sentire will attempt to establish a *virtual sensor* for the query. A virtual sensor is a component that binds queries to sensors in up to many-to-many relationships. Sentire uses virtual sensors to support efficiency and scalability and to abstract away the process of binding queries to sensors for the developer. When a query arrives at the SM, it first determines if an existing virtual sensor can be used to bind the query to the data source (we discuss details regarding this shortly). If no such virtual sensor exists, the SM determines which sensor to pass the query to by inspecting the query's data type and location. Once it has this information, it will query the SANETs listed in the RM for an appropriate sensor data source. We note that it is quite possible that a query will indicate a *set* of locations (e.g., requesting sensor data from multiple rooms in a building) covering a total area that might be larger than any one sensor data source registered with the RM can supply. In this case, the SM will find the most sensor data sources that can cover the span of locations indicated in the query. The SM will then create a new virtual sensor object describing the query's data type, location, ID, query type, and associated sensor data sources and then will store the object in its *virtual sensor cache*, as shown in Figure 5.4. Giving the SM the responsibility of building a one-to-many query-to-sensor relationship relieves the developer of the burden of having to issue multiple queries to sensor networks, which promotes usability and

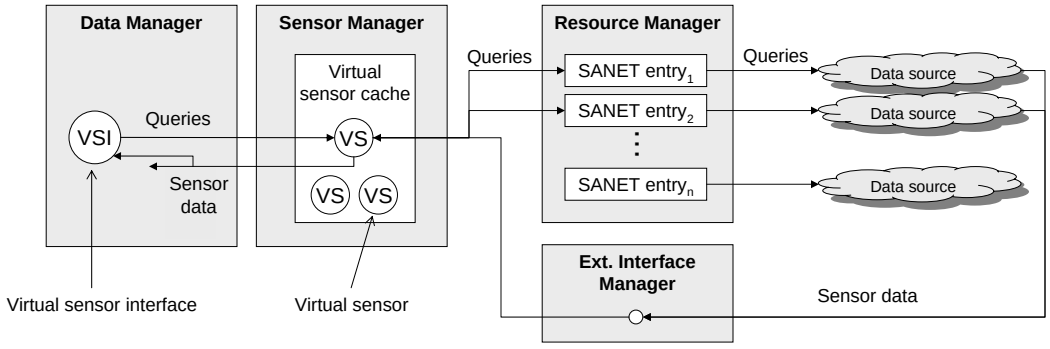


Figure 5.4: Sensor Manager's virtual sensors.

scalability. The virtual sensor cache is also used for efficiency. There could exist multiple queries either from inside the middleware or, in the future, from external applications. If the SM receives a query which it determines can be fulfilled from an existing virtual sensor in its cache, it will associate that query with the matching virtual sensor. The SM makes this decision by first checking its virtual sensor cache for a virtual sensor that describes the same data type and location(s) of the query. If one or more matching virtual sensors exist, the SM chooses (the first) one based on the query's query type (as described by the `SentireMessage` format). If the query's query type is a subscription, the SM will attach the query to an existing virtual sensor only if its associated with a subscription *and* the virtual sensor's subscription rate is higher or equal to that required by the query. We assume that the querier will not mind if data arrives faster than required. If the query's query type is a *one-time* query or a *threshold* (indicating that only data above or below a particular value should be returned), the SM will attach the query to an existing virtual sensor with a matching query type and, in the case of *threshold*, matching parameters. Overall, this reduces the number of redundant queries that might be sent to sensor data sources, thereby saving energy or bandwidth in a wireless sensor network.

As shown in Figure 5.4, sensor data (contained in a `DATA` message) is returned to Sentire via the EIM (as explained earlier), which routes it back to the original virtual sensor which issued the query on behalf of its queriers. The figure shows these queries to have originated from the Data Manager. The virtual sensor forwards

the sensor data to the Data Manager using the ID for each query attached to the virtual sensor so that the developer knows which sensor data can be used for which purpose. Another solution would be to simply broadcast the sensor data back to Data Manager with meta-data describing the details of the query which requested the data to begin with. This solution is inspired by the efficient broadcast-based mechanism of SHR (described in Chapter 4). However, this solution would probably find better use if Sentire focused on interfacing with external applications, whose presence may increase the magnitude and complexity of the overall system. For now, using query IDs is adequate. We describe an alternative method of returning sensor data to the querier in the next subsection. Continuing, if the query called for data over multiple locations, the SM simply waits until all the required `DATA` messages arrive, stores the `DATA` messages payloads in a list, and creates a new `DATA` message with the attached list to send back to the Data Manager. Last, we note that if a query can not be satisfied due to the absence of an appropriate data source attached to the framework *or* a rejection from the RM, the SM will return a `DATA` message with the query accept bit set to zero.

The SM also provides a programming space where a developer can implement query decomposition, which primarily supports the implementation of *quality of information* (QoI) policies. QoI, as we define it, describes how well sensor data meets the requirements of the application. There is no standard that specifies the exact metrics comprising QoI. Possible metrics could include sampling rate or sensor data redundancy, both of which are popular methods in the literature. However, another metric could be the sensor data type. For instance, in an enemy surveillance application, collecting motion data would provide a low QoI since motion does not *identify* an event as being intruder-related. Monitoring for sound might be better since characteristics of foot steps or vocal sounds can often differentiate between things such as animals and humans. This would provide medium QoI. The best QoI, however, would most likely come from visual sensors like infrared cameras. Notice that using different types of sensors also require different costs such as energy or network bandwidth. This scenario also points out that defining QoI can be very application-specific. Hence, we leave the definition of QoI to the system developer

and this serves as partial motivation for the SM's existence.

In Sentire's current incarnation, implementing query decomposition requires bypassing the operation of the virtual sensor. Also, since there is no header field in the `SentireMessage` format for defining the value of QoI, it is required that the developer add the header field to the message's payload. In this configuration, upon the SM receiving a query, the developer can simply query the RM for the appropriate data sources and issue specific queries on the Data Manager's behalf in order to best fulfill the QoI as the application (or developer) defines it. Recall previous mention that a sensor data source may be a network of devices, or a single device. Having access to single devices is especially helpful in the case of QoI management since a developer may need access to specific sensors in order to support QoI processing. For instance, in a surveillance application, querying sensors only in sensitive areas (e.g., hallways nears a server room where a power breaker is located) will mostly likely provide the highest QoI. As before, the developer can set her code to return a `DATA` message with the accept bit set to zero if adequate data sources or resources are not available. To relieve any complications, managing QoI and providing virtual sensor operation are mutually exclusive, either one or the other can be used at any one time.

The operation of the Actuator Manager (AM) is largely the same as the SM, including management of command decomposition. However, the AM receives `COMMAND` messages (for actuators) instead of `QUERIES`. Also, the operation of the similarly designed virtual actuator cache is different. First, the AM defines virtual actuators by location and action type instead of the fields used for virtual sensors. Second, since actuation affects the environment, simultaneous access to an actuator is currently prohibited. Therefore, only one-to-many query-to-actuator relationships are currently supported by Sentire. Furthermore, an actuator registered with a given Sentire middleware framework can only be associated with one virtual actuator at a time. This further restricts the possibility of conflicting commands being issued to the same actuator. These issues point out the significant challenge of controlling access to actuators in a large-scale environment. However, we expect that Sentire's method of controlling access is reasonable since sharing actuators can reveal critical

safety issues depending on the environment and application. This is why, where as actuation is concerned, we choose to focus Sentire's attention on distributed actuator coordination.

5.3.4 Data Manager

The Data Manager's (DM) responsibility lies in providing programming space for implementing the SANET system's control logic. In essence, all sensor queries and actuator commands are issued from the DM and all sensor data ultimately arrives here as well. Implementing the system's control logic is the responsibility of the middleware developer. Sentire offers several major components to aide the developer in this task. Two components, which are strongly related to the SM's and AM's operation, are the availability of *virtual sensor* and *virtual actuator interfaces*. When the developer wants to query for sensor data, she will make a new virtual sensor interface call specifying (in the parameter list) the query ID, query type, data type, and location. If the query type is a *one-time* query, these parameters are all that is required. If the query type is *subscription*, two additional parameters are required: the data refresh rate, and the number of data samples required. If the query type is *threshold*, additional parameters must specify the threshold value and whether values above or below the threshold should be returned. It is this programmer method call in the DM that causes the transmission of a `QUERY` message and hence, causes the processing of virtual sensors in the SM. Similar operation applies for the virtual actuator interface. However, only one method call is used requiring the following parameters: action type, location, strength, and direction. As eluded to earlier, sensor data may be returned to the DM in two ways. The first, which was described in the last subsection, is asynchronous. After executing the virtual sensor interface call, `DATA` messages will arrive at the DM via the `messageReceived` method, which must be implemented by the developer. This is helpful in the case that the developer wants to tend to other tasks while waiting for data to return. However, a developer may want a simple way to have all her required sensor data available before making any other decisions. To accommodate this, there is a second virtual sensor interface method call which is blocking. The method call, which

is similar to that for the first virtual sensor interface, will return the sensor data as specified in the method call's parameters before continuing with the rest of the program. This method can only be used for *one-time* queries.

As described in the previous subsection, not all queries and commands from the DM will require the intervention of virtual sensors/actuators because a developer may wish to employ query/command decomposition. In this case, the developer can execute a *simple query/actuator method* call that will simply pass data to the SM or AM so that more customized processing can occur. Last comes the issue of actuator coordination, which will be described next.

5.3.5 Coordination Manager

The Coordination Manager (CM) embodies the logic for distributed actuator coordination. Again, Sentire focuses on coordination that addresses resource contention. Specifically, the CM helps manage behavior of actuators that are distributed among different Sentire middleware instantiations, but still use resources from a shared entity as illustrated in Figure 5.5. This scenario can apply to large-scale control systems such as climate control (or HVAC) systems. Here, a straightforward example would include a system configuration where each office (or general room such as that used for storing servers) in a building has a Sentire instantiation that is charged with managing the climate for that office. Sentire would independently control a local heater unit (or units) using sensors deployed throughout the office. Coordination between Sentire instantiations of multiple offices becomes necessary if there is an energy budget imposed that prohibits all offices from simultaneously being heated to their desired user preferences. This restriction might be imposed to keep energy costs low, reduce the probability of a grid brownout, or prolong the operation of a back-up generator supplying the building's power.

The CM facilitates actuator coordination using a distributed bidding scheme where each Sentire instantiation bids a percentage of its *budget* for the use of a common resource. In general, the higher the bid is against other bidders, the greater amount of resource that is allocated to the bidder. This scheme is beneficial for various reasons. One, it is simple, not requiring a great amount of computing

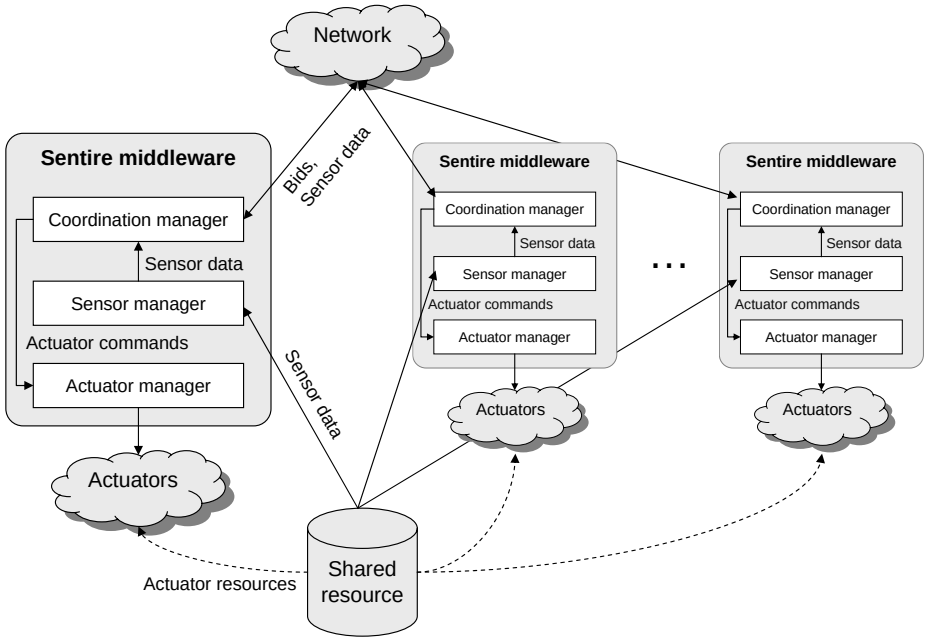


Figure 5.5: Sentire's actuator coordination setup.

power, yet is effective. Two, it is scalable, since no central entity is required and computation does not increase along with the number of participants. Three, it has roots in microeconomic and auction theory, enabling the potential for including powerful analytical methods and sophisticated algorithms for optimization in the future. We imagine that in some cases, this bidding behavior (implemented by the CM) will guide the entire behavior of the Sentire middleware. Such a scenario may include a winter setting where there are frigid outdoor temperatures and all occupants of an office complex want their offices constantly at a well-heated state. However, in other cases, coordination may only be needed periodically. We will explain these details next.

The CM's operation is triggered by the DM. If a developer determines that coordination is needed, perhaps due to numerous rejections from the RM to perform actuation, he can send a `BID` message to the CM including the bid amount as well as the action type of the actuator. For simplicity, we assume that a given action type corresponds to only one type of resource. When the CM receives the `BID` message, it

will determine the correct allocation of resources according to the following equation:

$$\left(1 + \frac{1}{2^n} - \frac{1}{2^a}\right) \times \left(\frac{b_i^2}{\sum_{j=1}^{n} b_j^2}\right) \times te \tag{5.1}$$

where

- n = total number of known Sentire instantiations comprising the entire control system;
- a = total number of known Sentire instantiations participating in coordination (i.e., actively bidding);
- b_i = Sentire instantiation i's current bid;
- te = total amount of resource made available to the entire system.

Both n and te are assumed to be known by all participants in the system. Figure 5.5 indicates that te (or any other data associated with the shared resource) may be collected in the form of sensor data. We will shortly explain how all participants learn the value of a, which is a simple process. Overall, Equation 5.1 will give each participant an equal allocation of resources if bids are the same, no matter what the value of the bid is. The equation is also designed so that if there is only one participant that is bidding (i.e., $a = 1$), then the entirety of the resources (te) will not be consumed by that sole individual, but only a fraction. This fraction of te will asymptotically approach 0.5 as n grows. This should be acceptable since we expect the value of te to be chosen such that $te/2$ should be enough to satisfy the requirements of any one actuator acting alone in the system. Furthermore, this fraction can be acquired using a very low bid. Overall, this is beneficial for those systems that may always operate in a state of coordination, as previously mentioned.

Continuing, when the CM receives the bid from the DM, it will send a `BID` message (specifying the bid value and action type) to all other participants of the system. We do not specify how this message is propagated, but assume that either

the CM knows the network addresses of all other Sentire instantiations, or a broadcast, multicast, or flooding policy has been implemented in the network. When the CM receives a bid from an external Sentire instantiation, it will decide to join in coordination only if it knows that its own host is currently attempting to actuate using the same action type. This information is determined simply by querying the Actuation Manager. If the CM decides to participate in coordination, it will send its bid to all other members of the system as well. If the CM decides not to participate, it will simply transmit a bid with a null value. Using this process, each member of the system participating in bidding will learn the value of a as well as the total of everyone's bids and hence, can use Equation 5.1 to determine their allocation of resources. While this describes how the CM manager operates, we do not intend to indicate that bidding is a continuous process. The frequency at which bidding occurs is left up to the developer to decide. In the usage scenario described in Section 5.4.2, we describe several possible ways to control bidding.

A caveat of the above process is that continuously identical bids will only result in each participant gaining an equal share of resources. If this occurs, it is possible that no actuator will ever have an opportunity to fulfill its local goal. Hence, as a rule of thumb, there should be some factors that differentiate when or how long a Sentire instantiation is able to acquire all the resources it needs so that other instantiations will have the chance to meet their respective goals. It is very challenging, and perhaps not appropriate, for the CM to enforce how bidding is differentiated because this is a very application-oriented feature. The amount of a bid or when a bid is placed may be a factor of any number of features (e.g., task, priority, time elapsed since last time the actuator's goal was reached, etc.) that are specific to the Sentire instantiation. This information is straightforward for the CM to obtain. However, our investigation of example applications reveals that some knowledge of other participants' sensor data could be useful in guiding bidding behavior. We show an example of this in Section 5.4.2. The CM supports this by being able to collect the sensor data of other Sentire instantiations, as illustrated in Figure 5.5. When a `BID` message is sent out, it can include any sensor data that the developer chooses to place in the message. One might suggest more flexible ways

of sharing sensor data. A major one might be to allow the CM to query the sensor data of other Sentire instantiations much in the same way as using the virtual sensor interface. However, the querier must have intimate knowledge of what sensor data to collect. This is non-trivial because only certain data guides an actuator's tasks. For instance, knowing what *type* of sensor data to collect is fine, but knowing the *location*, which strongly guides an actuator's behavior, is difficult. Hence, allowing the bidder the opportunity to include whatever sensor data he sees fit for other bidders to make informed decisions in the most straightforward option.

5.4 Usage Scenarios

In order to demonstrate the potential of the Sentire framework, we used it to compose two example SANET control systems. The first dealt with using a small-scale SANET hardware platform to conduct object identification. The second dealt with using a simulated HVAC system to demonstrate the benefits of Sentire's actuator coordination features.

5.4.1 Object identification

We developed a test application that exemplifies the task of object identification and demonstrates the collaborative nature of heterogeneous sensing, both of which are features that may be found in a SANET system. At the time of this demonstration, which was actually conducted at the IBM T.J. Watson Research Center in Hawthorne, NY, actual Berkeley mote hardware was not available for our use. As a substitution, we opted to implement our SANET system using the LEGO®Mindstorms^{TM}kit [42], which provides a convenient, yet powerful SANET experimentation platform. The following LEGO components were used to build our test platform: 2 light sensors with adjustable sensitivity, 2 LED lamps, 1 USB camera capable of motion detection, and 1 RCX micro-controller brick.

The RCX brick was used to connect the light sensors and LEDs to the SANET server (labeled SANET *gateway* in Figure 5.6), while the motion detector communicated directly with the gateway, as shown in Figure 5.6. The SANET server, which hosted a Sentire driver component to convert Sentire commands into the language

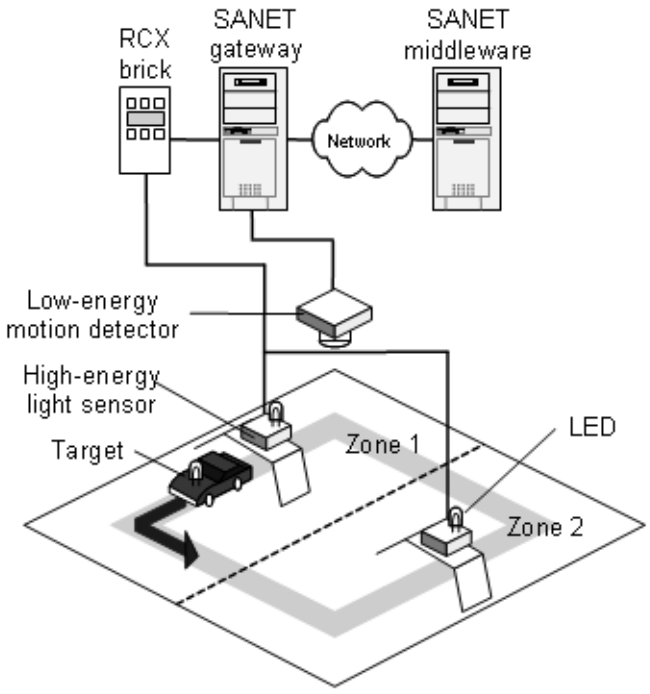


Figure 5.6: Object identification usage scenario schematic.

of the LEGO Mindstorms platform, connected all sensors and actuators through an Ethernet network for interaction with the middleware. We also note that the Mindstorms's programming language was Java, which made it easy to control the platform via the Sentire framework.

The purpose and behavior of the SANET system was set up in the following manner. The system was used to detect a small model car with a lamp attached to it. The motion detector, representing a low-energy and low QoI sensing device, was used to detect the car's presence in either of two zones an depicted in Figure 5.6. When motion was detected in a zone, the zone's respective light sensor, representing a high-energy and high QoI sensing device, was triggered to inspect if the car was illuminated. If a car's lamp was detected, the zone's respective lamp was actuated, indicating successful target identification. In essence, this represents an application scenario where the first step is to perform detection in a low-power manner and then switch to high-power detection only when needed in an attempt to use the SANET's energy wisely.

By using the Sentire development framework, the development of the SANET

system was a straightforward, systematic process. From the perspective of middleware developer, we used four Sentire manager components to instantiate the middleware for the test application. We used the Resource Manager to periodically store the RCX brick's residual battery charge. Additionally, we wrote a simple policy which suppressed sensor and actuator instruction execution after the residual battery charge decreased below a prescribed threshold. We extended the Sensor Manager to select a particular sensor (i.e., motion detector or light sensor) based on the QoI value indicated in the sensor query. Since this demonstration did not represent a SANET *control* system, the Actuator Manager was used simply to actuate the LED if the light sensor detected the car. The Data Manager was used to issue all queries and commands.

The flow of information through the framework proceeded as follows. One, the DM submitted a request, to the SM, to collect sensor data of type *car* in locations (or zones) 1 and 2 with a QoI value of one. Upon receiving the query, the SM queried the LEGO system for data type *motion* as a response to QoI being one. Upon receiving a positive data report indicating the zone in which the movement was detected, the DM then immediately queried for sensor data of car again, only this time with a QoI value of 2. Using this information, the SM queried the SANET for *light* in the respective zone. This second query's type was a *subscription* query for 20 data samples at an interval of 0.5s. A subscription was used because the light sensor had a narrow field of detection in comparison to the motion sensor; the illuminated car had to be almost directly underneath the sensor in order for its presence to be acknowledged. Therefore, as soon as the car entered a zone, the sensor periodically measured for light to increase its chance of detecting the car. Upon receiving a positive report from the SM in response to its second query, the DM sent a command to the AM to actuate a device of type *light* in the appropriate zone, which caused the correct LED lamp to illuminate.

For the majority of times, the composed system successfully completed its goal. In order for the light sensor to operate properly, the ambient lighting in the room had to be at a very dim state. For some tests, we also adjusted the light sensor's sensitivity as a function of QoI, using a low sensitivity level for a low QoI and the

opposite for high QoI. This also proved to be effective, as detection was more reliable for higher sensitivity levels. However, past a certain sensitivity, performance again decreased because the sensor became so sensitive to light that it responded either to the reflection of light from the car, or ambient lighting in the room.

5.4.2 HVAC Control

This subsection describes a usage scenario involving a simulated HVAC system to demonstrate the usefulness of Sentire's actuator coordination facilities. We used the SimJava Java-based simulation framework [69] to implement a model of a small office complex's climate change behavior. The virtual complex consisted of four separate rooms with independent climate, or heat loss, properties. A survey of the literature reveals numerous ways to calculate the heat loss of a room (defined as the rate at which heat leaves a room). For our purposes, we used the following set of equations to define a room's heat loss:

$$heatloss = heatloss_{walls} + heatloss_{windows} + heatloss_{infiltration}, \tag{5.2}$$

$$heatloss_{walls} = \Delta t \times \left(\frac{a_{outsidewall}}{r_{outsidewall}} + \frac{a_{insidewall}}{r_{insidewall}} + \frac{a_{ceiling}}{r_{ceiling}} \right), \tag{5.3}$$

$$heatloss_{windows} = \Delta t \times a_{windows} \times tf_{windows}, \tag{5.4}$$

$$heatloss_{infiltration} = \Delta t \times \rho \times ER \times c \times v_{room}. \tag{5.5}$$

The variables in the above equations are defined as follows:

- Δt = indoor temperature minus outdoor temperature (°F);
- a_x = the area of x with x being outside walls, inside walls, windows, or ceilings (ft^2);
- r_x = the thermal resistance of x with x being outside walls, inside walls, or ceilings (hr·ft^2·°F/BTU);
- $tf_{windows}$ = the transmission factor of the windows (BTU/(hr·ft^2·°F));
- ρ = air density (lb/ft^3);

- ER = the exchange rate of air ((air change)/hr);
- c = *the specific heat of air* (0.24 BTU/(lb·°F));
- v_{room} = the volume of the room (ft^3).

Table 5.1 lists the variable values of the heat loss equations above that we used in our simulation; these values were consistent for each room. Heat loss determines the rate of heat a heater needs to generate in order to sustain a specific indoor temperature. So, if a room experiences a heat loss rate of x when the indoor temperature is 70°F and the outdoor temperature is 30°F, then a heater needs enough power to generate x BTU/hr to keep the room's temperature at 70°F. As a note, the following equation, converts BTU/hr to kw (power):

$$kw = 2.931 \times 10^{-2} \cdot \frac{\text{BTU}}{\text{hr}}. \tag{5.6}$$

We used this equation in the HVAC simulation to determine the heat output of a virtual heater given a certain power setting. The architecture for this demonstration was similar to that depicted in Figure 5.5. A separate Sentire instantiation was used to connect with each virtual room, where sensor data was represented by temperature readings and actuator commands were represented as commands to change the power setting (in kw) of the room's heater. We only adjusted heater settings (and not air conditioner settings) because we assumed a winter season. The temperature of the rooms changed in accordance with the supplied power (which affected the heater output) and the outdoor temperature. The indoor temperature was not allowed to fall below the outdoor temperature. Four virtual offices were used for the demonstration.

The goal of the system was to coordinate heaters' usage of a shared power source given their individual heating requirements. This is representative of the previously described scenario in which an HVAC system must operate efficiently given a limited and shared power source. For this usage scenario, we set te to 3500kw. This value was chosen because it is well below the power needed for all rooms to simultaneously sustain their temperature goals, forcing the need for coordination. Each room's temperature goal was set to 70°F, however, not all rooms

$a_{outsidewall}$	68ft^2
$a_{insidewall}$	564ft^2
$a_{ceiling}$	140ft^2
$a_{windows}$	16ft^2
v_{room}	1680ft^3
$r_{outsidewall}$	16.97 hr·ft$^{2\cdot\circ}$ F/BTU (for typical building materials)
$r_{insidewall}$	0.45 hr· ft$^{2\cdot\circ}$ F/BTU (for typical building materials)
$r_{ceiling}$	49 hr· ft$^{2\cdot\circ}$ F/BTU (for typical building materials)
$tf_{windows}$	1.13 BTU/(hr· ft$^{2\cdot\circ}$ F) (for single pane glass)
ρ	0.0741 lb/ft^3 (for normal pressure and moderate temperature conditions)
ER	0.4 (air change)/hr (for well-insulated new construction)

Table 5.1: Heat loss variable values used for HVAC simulation.

required the same amount of power to reach this temperature. To enforce this, we gave each room its own outdoor temperature. We realize that this phenomenon in itself is not realistic. In reality, a room's heater's output required to sustain a particular temperature could be based on things such as the number of people in a room, whether a door or window in the room is open, the heat output of computer equipment or machinery, or simply if the room is receiving direct sunlight through its windows. We found that simply adjusting the outdoor temperature adequately represented any of these factors for purposes of simulation. We set room 1, 2, 3, and 4's outdoor temperature to 60°F, 40°F, 40°F, and 30°F respectively.

Now we describe the activity of the Coordination Managers in the Sentire middleware. First, this demonstration was configured such that the CM always drove the operation of the Sentire middleware. Hence, there were no messages sent from the Data Manager. We chose this setup because the purpose of this demonstration was to focus on actuator coordination and also consistent operation of the CM allowed for more meaningful data plots (i.e., without anomalous data points representing the absence of any coordinated behavior) which we present later. As mentioned previously, each `BID` message included the bid value as well as the pertinent sensor data value. We extended the CMs so that each had a maximum budget of 50. When the budget was depleted, we used sensor data to help differentiate when the rooms' Sentire instantiations were allowed to bid by controlling the rate

at which each room gained its budget back. CMs were only allowed to bid for power after their budgets increased back to the full amount. Each CM used the following three rules to guide the rate at which its budgets increased:

B1. `IF` (<room's temperature> is within 10% of <mean temperature for all rooms>) `AND` (room's heater is turned off)
`THEN` increase budget at a rate of $[2 + 0.001 \cdot (te -$ <current power usage for all rooms>) every 10 seconds;

B2. `IF` (<room's temperature> is greater than 10% of <mean temperature for all rooms>) `AND` (room's heater is turned off)
`THEN` increase budget at a rate of 1 every 10 seconds;

B3. `IF` (<room's temperature> is less than 10% of <mean temperature for all rooms>) `AND` (room's heater is turned off)
`THEN` increase budget at a rate of $[3 + 0.001 \cdot (te -$ <current power usage for all rooms>) every 10 seconds.

These rules were designed in accordance with the following motivations. The major causes for a room being in the state indicated in Rule B2 are either the room having some ambient source of heat (e.g., sunlight or people), or the room just happening to heat itself higher than all other rooms. Therefore, there is likely no need to increase budget quicker than all other rooms' CMs. Other rooms should have a higher priority of gaining budget so that they can power their heaters. As for Rule B3, the major causes for this state are likely to be that (1) this room's heater stopped before all other rooms' heaters; (2) this room looses heat at a faster rate than other rooms; or (3) on average, this room's temperature is lower than that of the other rooms. For cases (1) and (2), it is justified that this room's CM increase its budget quicker than other rooms' CMs so that it can start the heater sooner if necessary. If case (3) is not true, then the penalty is small. Even though the room's CM increases its budget quicker, it will not require much power for heating in comparison to other rooms. Continuing, Rule B1 might be followed because every room's temperature is about the same, or this room's temperature simply falls near the average of all temperatures. Here, the room's CM should simply store budget at a medium (or

default) rate in comparison to the other rules. We assume that the *current power usage for all rooms* can easily be obtained as sensor data from the shared power source. Another way to learn this data is for a room's CM to include it in its `BID` message.

5.4.2.1 Evaluation

We used the Sentire framework and the CM's budgeting policies described above to control a simulated HVAC system. For evaluation purposes, we collected some application performance metrics using several CM bidding strategies. These strategies are described below:

S1. When the room's temperature falls below 70°F, bid the entire budget (50) and use the full amount of the allotted power to power the heater;

S2. When the room's temperature falls below 70°F, bid the entire budget (again using the full amount of allotted power) and when the temperature passes above 75°F, deactivate the heater and receive budget back;

S3.
- All CM's begin by bidding 1 unit of budget;
- `IF` (the previous bid yielded a temperature gain) `AND` (room's temperature is below 70°F) `AND` (some budget remains)
  `THEN` keep the previous bid invested;
- `IF` (the previous bid yielded a temperature loss) `AND` (room's temperature is below 70°F) `AND` (some budget remains)
  `THEN` bid an additional unit of budget;
- `IF` (the room's temperature is greater than 75°F) `AND` (heater is on)
  `THEN` turn off heater, reduce bid to 0, save remaining budget.

S4. Same as [S3.], only bids of 3 units are used;

S5. Same as [S3.], only a bid of 3 units is used if three consecutive bids of 1 consecutively yield temperature losses.

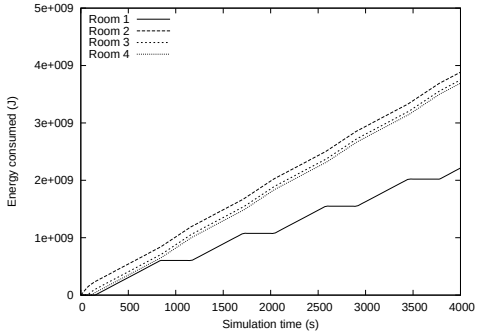
As a reminder, the CM turns the heater off any time there is no budget remaining. Also, any time a new bid is received, the CM will adjust its power allocation via a re-evaluation of Equation 5.1. Strategies S1 and S2 uses timers initialized to 500 seconds to control how long a heater is allowed to stay on while the room's temperature is above 70°F. When the timer runs out, the CM will turn off the heater, set its budget to 0, and start to increase its budget according to the rules stated earlier. This helps to give all CMs the opportunity to gain adequate power to heat their respective rooms past the target temperature. For strategies S3-S5, bidding only occurs every Δ seconds. We used a Δ value of 10s. All simulations were run for 4000s.

For each strategy tested, we plotted and compared the following data:

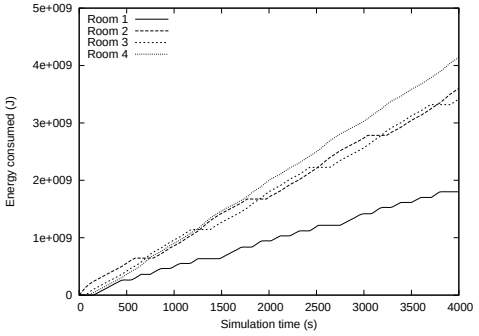
- energy consumed per room (J),
- power consumed per room (kw),
- temperature per room (°F),
- average and cumulative average percent of power savings.

The average percent of power savings was calculated using the average power usage of all rooms and the total power usage needed for all rooms to constantly sustain a temperature of 70°F.

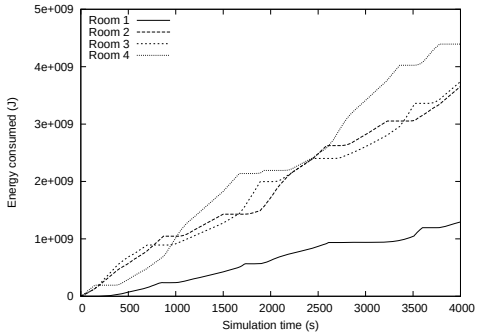
Figure 5.7 shows the plots for each room's energy consumption during the simulation. All strategies show that Room 1 uses the least amount of energy, which is expected since its outdoor temperature is set the closest to 70°F. Strategy 1, shown in Figure 5.7(a), induces a small amount of differentiation between the remaining rooms' energy consumption in comparison to the other strategies. The other strategies, 2-5, serve the application better since, according to Figures 5.7(b)-5.7(e), room 4 uses the greatest amount of energy. This is desired since room 4's outdoor temperature was set to the lowest value among all the rooms. Where as strategy 2 induces fairly consistent energy consumption behavior, Figures 5.7(c)-5.7(e) illustrate that strategies 3-5 induce less steady behavior. This is not necessarily good or bad. The remaining data plots for other performance metrics do a better job at revealing the advantages (or disadvantages) strategies 3-5 may present.



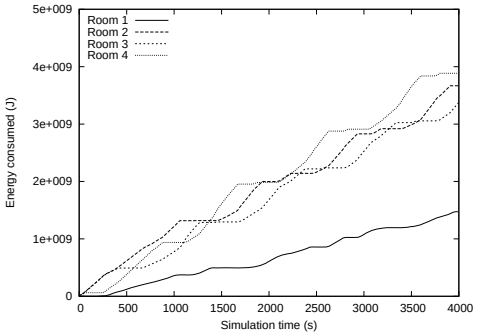
(a) Energy consumed for strategy 1.



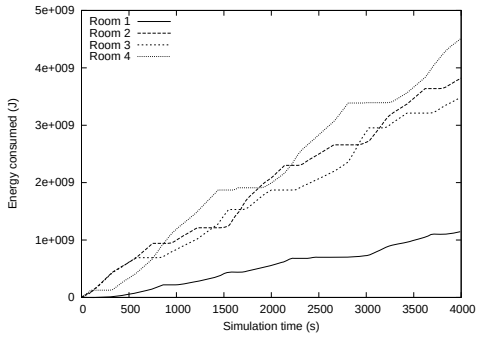
(b) Energy consumed for strategy 2.



(c) Energy consumed for strategy 3.



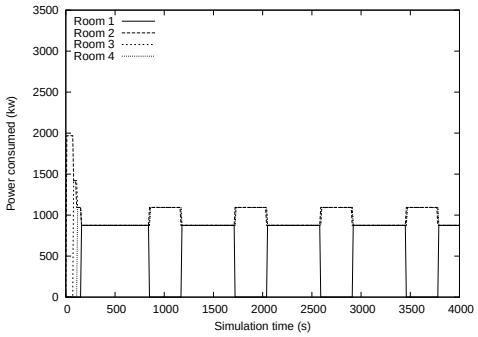
(d) Energy consumed for strategy 4.



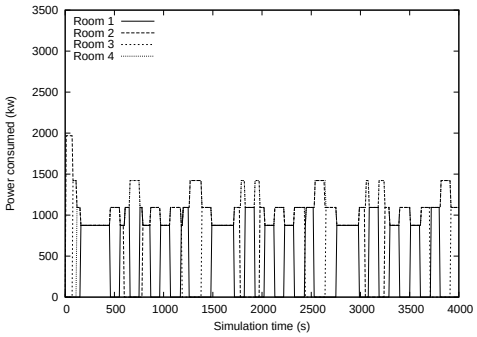
(e) Energy consumed for strategy 5.

Figure 5.7: Energy consumed per room.

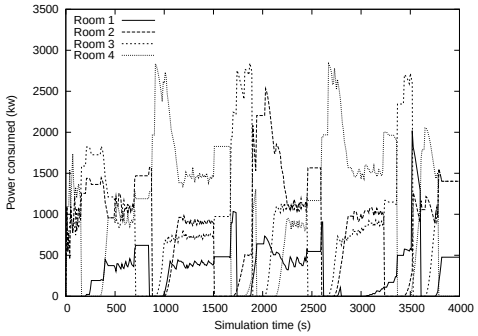
Figure 5.8 shows how the bidding strategies affect the power consumed by the rooms' heater units. Figures 5.8(a) and 5.8(b) show that strategies 1 and 2 limit the rooms' power usage to a rough average of 1000kw for most of the simulation. The behavior for strategy 2 shows a good amount of oscillation where heaters are turning off as the temperatures increase past 75°F and then turn on again when temperature decrease past 70°F. This surely reduces power consumption in the long run. However, it is not yet apparent that target temperatures are being met using



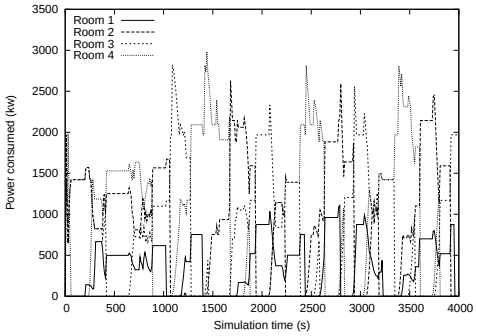
(a) Power consumed for strategy 1.



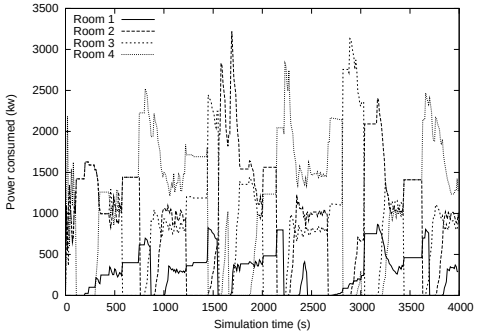
(b) Power consumed for strategy 2.



(c) Power consumed for strategy 3.



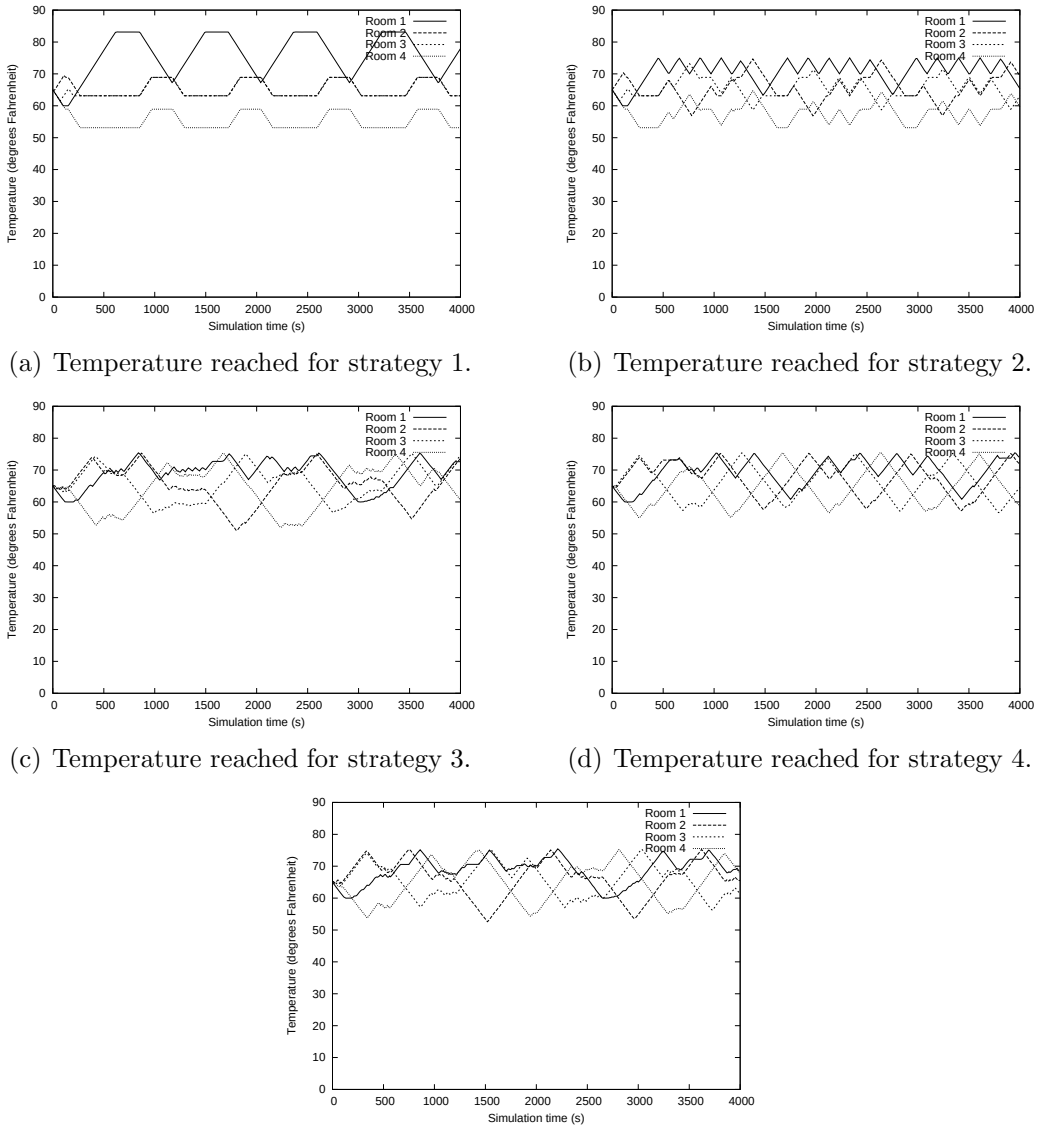
(d) Power consumed for strategy 4.



(e) Power consumed for strategy 5.

Figure 5.8: Power consumed per room.

this strategy. Figures 5.8(c)-5.8(e) are particularly interesting. As in the previous energy plots, it is not difficult to see that room 1 uses the least amount of power on average. It is difficult to decipher the behavior of the other rooms' heaters. However, one can see a general trend that on average, a room's power usage goes through intervals in which either there is a peak followed by a gradual slope downward, or usage slopes upward to a peak, and then sharply drops. This illustrates the adaptive behavior of bidding strategies 3-5. Figures 5.8(c) and 5.8(e) in particular



(a) Temperature reached for strategy 1.

(b) Temperature reached for strategy 2.

(c) Temperature reached for strategy 3.

(d) Temperature reached for strategy 4.

(e) Temperature reached for strategy 5.

Figure 5.9: Temperature reached per room.

show some evidence of periodic convergence of power consumptions roughly around 1000kw, which is the almost the same average value shown in Figures 5.8(a) and 5.8(b). So while these averages are roughly the same, strategies 1 and 2 cause all the rooms to almost consistently exhibit this average power consumption value. We look to other plots to see if there are any advantages or disadvantages to this behavior.

Figure 5.9 illustrates the temperatures the rooms reach as a result of the five tested strategies. Here, we clearly see the disadvantages of employing strategies 1

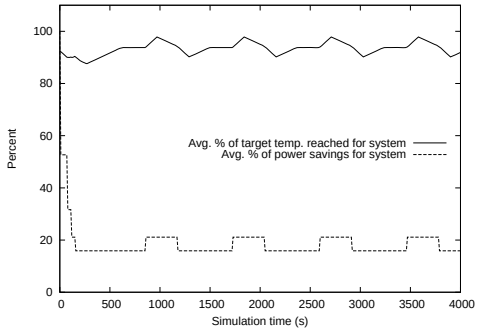
and 2, as Figures 5.9(a) and 5.9(b) show that room 1 is the most frequently heated above 70°F while room 4 largely suffers. This shows that while room 4 consumes more energy than room 1 for strategies 1 and 2, it is not enough to serve room 4's temperature needs. However, the remaining strategies, whose behavior is shown in Figures 5.9(c)-5.9(e), induce a much larger degree of fairness in the system, as all rooms are periodically heated to at least 70°F. A particular advantage of strategy 4, which uses a more aggressive bidding strategy (i.e., 3 units as opposed to 1) is that no room's temperature drops far below about 55°F, as it does with strategies 3 and 5.

Figure 5.10 shows the power savings that a strategy is able to achieve in relation to the percent of target temperature reached. All strategies help the entire system reach the target temperature roughly 90% of the time. However, in pairing these plots with those shown in Figure 5.9, we know that this average does not reflect the starvation of room 4 for strategies 1 and 2. Strategy 1's behavior, depicted in Figure 5.10(a), is not as efficient as the other strategies since not only does it yield poor temperature performance (according to previous plots), but it only saves 20% of the system's power consumption on average. The other strategies do much better. Strategy 2 periodically induces larger power savings, again because the heater cuts off at a temperature beyond 75°F. Strategy 4, shown in Figure 5.10(d), appears to yield the greatest amount of power savings throughout the simulation. Figure 5.11 shows the *cumulative* values for the data shown in Figure 5.10. Inspecting this figure in conjunction with all the previous one helps solidify the strong performance of Strategy 4. This strategy helps room 4 avoid starvation of resources, yields the second greatest overall percent of target temperature reached for the system (shown as percent *not* reached in Figure 5.11), and yields the largest percent of system power savings. Strategy 5 is also a rather efficient strategy, helping the system reach an even larger percentage of its temperature goal and yielding nearly 25% power savings. This shows that a developer might choose either strategy 4 or 5 depending on the goal of system being either power savings or actuation priority.

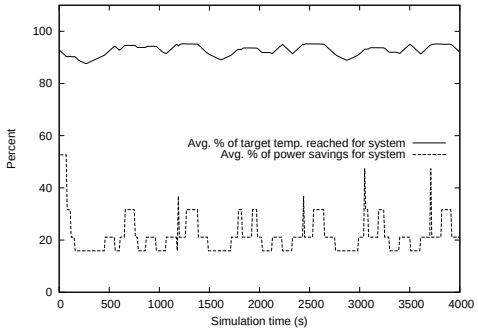
5.5 Concluding Remarks

This chapter described the details of the Sentire middleware framework and gave some examples of how it can be used to build meaningful SANET control systems from a high-level perspective. In Chapter 6, we describe some future research objectives regarding Sentire. Investigating Sentire's usefulness in facilitating actuator coordination also lead to an insightful way of managing power consumption for distributed HVAC control systems. This is a problem of growing interest and importance to the government as it searches for all ways to reduce energy consumption. Therefore, Sentire shows promise of addressing real-world and timely problems regarding sensor and actuator control.

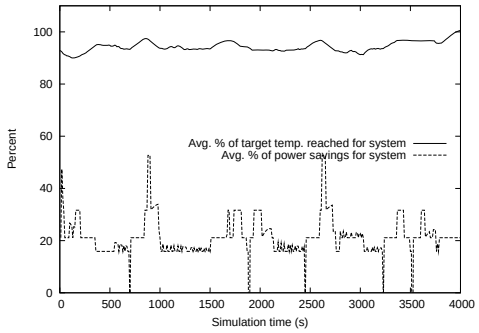
Here, we make a note that Sentire mostly applies to delay tolerant applications, especially those in an enterprise context. There are no system-defined policies that consider such things as the time it takes to get a message to an actuator or the time it takes for an actuator to change an environment to a desireable state. This could well be used for coordination purposes, but we leave that for future work. We note that this also brings in the issue of safety. Sentire does not focus on safety-critical applications, which would largely require the consideration of delay and a fine-grained analysis of how actuators' treatment tasks conflict in the environmental sense, instead of just that regarding resource usage. We leave these concerns for future research.



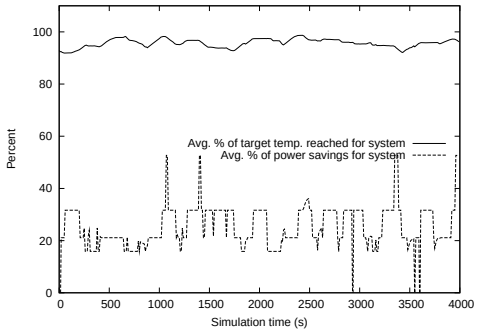
(a) Percent of goal reached and power saved for strategy 1.



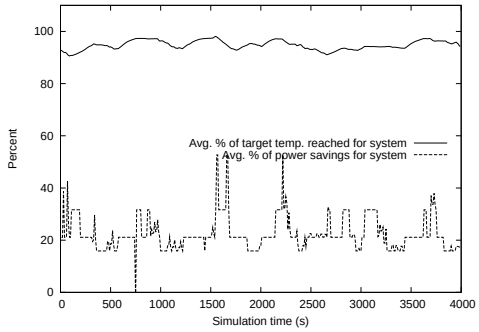
(b) Percent of goal reached and power saved for strategy 2.



(c) Percent of goal reached and power saved for strategy 3.



(d) Percent of goal reached and power saved for strategy 4.



(e) Percent of goal reached and power saved for strategy 5.

Figure 5.10: Average percent of temperature goal reached and power saved for the entire system.

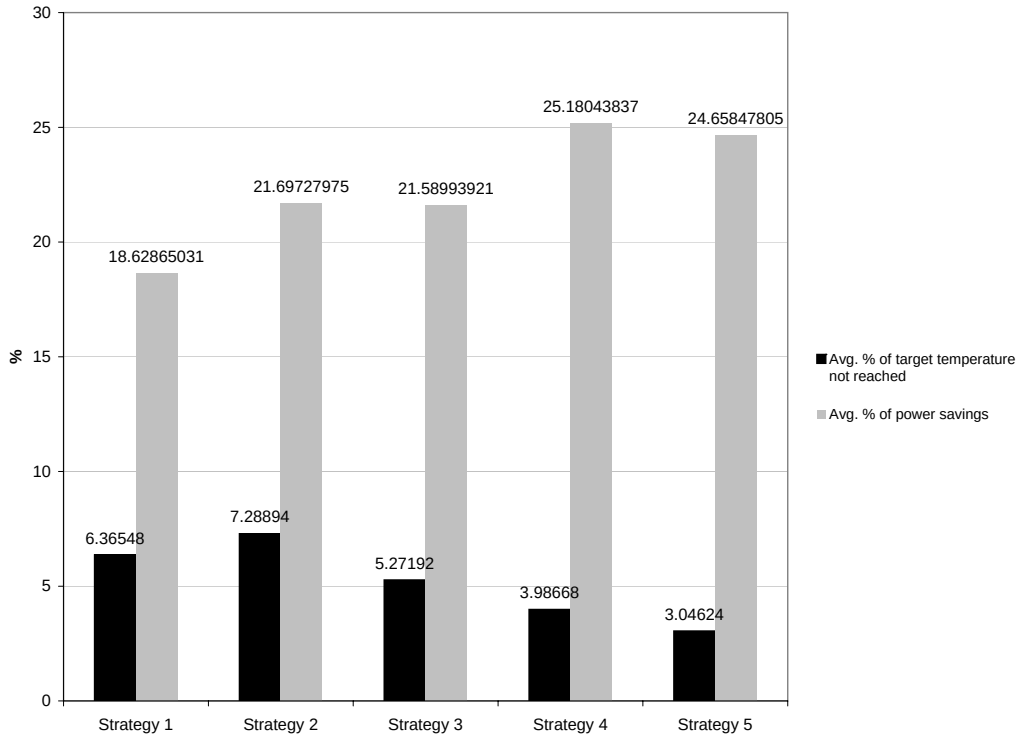


Figure 5.11: Cumulative average percent of temperature goal not reached and power saved for each strategy.

CHAPTER 6
Thesis Contributions and Impact

6.1 Introduction

This chapter summarizes this thesis' contributions to and impact on the wireless ad hoc networking and sensor and actuator network research communities and application domains. In regards to thesis impact, we specifically discuss applications that may be impacted by this thesis's contributions as well as future research opportunities that have been identified as a result of conducting this thesis research.

6.2 Contributions

This thesis has made three major research contributions to the field of sensor and actuator networks. In Chapter 3, we described ESCORT, an energy-saving topology-control algorithm for wireless sensor networks. ESCORT which operates directly below the networking layer in a WSN protocol stack, forms communities of nodes based on link quality and received signal strength, which is a particular novelty. Using these factors reduces the chances of communities forming around physical obstacles or other sources of wireless signal interference that may cause dropped messages needed for coordination. Additionally, using wireless signal quality metrics eliminates the requirement of using GPS or related location-tracking technology to form communities. This is beneficial since such technology may easily escalate the energy and monetary costs of WSN deployment. Members of a community coordinate to establish radio duty-cycles, which, as we showed, significantly reduces energy consumption while minimally reducing routing performance. ESCORT's placement below the routing layer and requirement that all community members assume a shared ID helps provide a measure of transparency to the network layer. Therefore, a major contribution of ESCORT is a topology-control algorithm that reduces network energy consumption while giving the network designer the freedom to choose the best wireless multi-hop protocol for the target application.

Chapter 4 described SHR, an opportunistic routing protocol for WSNs. SHR

abandons the traditional method of multi-hop routing requiring nodes to choose which neighbor to send packets to (in either a static or dynamic manner) in order to communicate with a destination. Instead, SHR relies exclusively on each node knowing only their routing cost to a packet's destination. In SHR, a node essentially broadcasts packets out to all nearby neighbors, without knowledge of their IDs or even their existence. Receivers of the packet autonomously make the decision to forward the packet based on their routing cost compared with the sender's; the lower the cost, the higher the probability that the receiver will forward the packet. This is supported by SHR's self-selection algorithm. If a node overhears the packet forwarded before the end of its own delay, it will drop the packet knowing that a node with a lower cost to the destination exists. Chapter 4 describes further in detail how this algorithm reduces the probability of multiple nodes self-selecting themselves to forward a packet and reduces collisions. SHR and its variant, SHR-M, are novel in several aspects. One, they primarily define routing cost based on hop distance to the destination. This again eliminates the need for costly GPS technology and enables both protocols to locally and in a greedy manner find the shortest available paths to a destination while balancing energy usage. Two, SHR and SHR-M can easily be extended to implement their own topology-control protocols that probabilistically enables nodes to stay awake only when they are needed to route packets and reduce energy consumption due to overlistening. Three, SHR also provides a local route repair algorithm that adjusts nodes' costs in order to seamlessly route packets around holes, which occur when a node has no neighbors of lower cost than itself. All of these novelties help solidify SHR's and SHR-M's contributions as efficient protocols that are significantly fault-tolerant (i.e., able to accommodate faulty wireless links and nodes), energy-efficient and generally accommodative of unsynchronized topology-control algorithms, and extensible. We describe how the protocols can be extended in the Future Research section of this chapter.

Last, Chapter 5 described Sentire, a framework for developing high-level middleware for SANET-enabled systems. Sentire offers a collection of several extensible inter-connected manager components for facilitating the flow and processing of queries, data, and control messages between applications and underlying sensor

and actuator resources (i.e., devices or services). Each manager provides particular capabilities and abstractions supporting commonly anticipated SANET middleware functions. For instance, the *Resource Manager* tracks what sensor and actuator networks are currently associated with the middleware and provides descriptors (e.g., data type, energy level, etc.) to facilitate the selection of sensor and actuator networks based on query and command requirements. The *Sensor Manager* supports the decomposition of application-level queries in order to configure underlying sensors to provide a particular quality of sensed information as specified by the application. This is an extensible feature since quality of information is largely an application-dependent metric and hence may describe any number of sensor aspects such as sampling rate, number of similar data streams, etc. The Sensor Manager also maintains the logical bindings between application queries and underlying sensor networks. Multiple networks may be bound to single queries, helping to provide sensor data over larger geographic areas, and multiple queries may be bound to the same network, stopping Sentire from sending redundant queries to a network and hence reducing communication overhead and potentially related energy consumption. The characteristics of the other managers, *Actuator*, *Data*, and *External Interface*, are also described in Chapter 5. A particular novelty of Sentire is its support of decentralized actuator coordination, which is handled by its *Coordination Manager*. This manager uses a lightweight budgeting approach that allows a Sentire instantiation to locally control the extent to which an associated actuator may use a resource that is shared by actuators belonging to other Sentire instantiations. In essence, an actuators stores budget when not using the shared resource and spends it otherwise. Global sensor data determines the rate at which budgets grow, probabilistically giving actuators with either more urgent tasks or have been waiting the longest to use a resource higher priority to use the resource in the near future. We showed how this simple approach was effective in coordinating the energy usage in a simulated HVAC application. Sentire's actuator coordination policy represents a firm contribution to the community since very few SANET middleware frameworks support this feature. The entire manager-based framework also represents a notable contribution in that it (1) supplies extensible and reusable components

for composing SANET middleware; (2) provides intuitive SANET resource abstractions to shield solution developers away from intimate details of underlying SANET resources; and (3) logically separates SANET middleware development tasks based on system functionality (e.g., data processing, resource management, and query translation).

6.3 Applications

We begin this discussion by addressing applicable uses for SHR. SHR was not designed with any *specific* applications in mind, as in many ways, it is a general purpose routing algorithm. However, as was previously mentioned, we can claim that SHR is highly useful for sensor network deployments in which the availability of wireless links may be transient and unpredictable and sensor nodes may be unexpectedly destroyed or added to the network. Given this, *military battlefield* scenarios are a befitting domain for SHR. Here, one can easily expect sensor nodes to be destroyed at any point in time. Following this expectation, it is conceivable that replacement nodes may be dropped from aircraft (either manned or unmanned) onto a battlefield to continue to collect data. SHR can readily accommodate such scenarios since it does not require nodes to rely on maintaining their neighbors' statuses to make fault-tolerant routing decisions. Following in the spirit of this example, we claim that SHR is equally useful for *emergency response* scenarios. One such example includes detecting and monitoring forest fires. If a significant fire occurs, we can expect elements such as extreme heat, smoke, debris, and the fire itself to contribute to weakening wireless connectivity and destroying sensor nodes. Again, this is a situation in which nodes have to quickly coordinate to find the shortest and most resilient routes back to a base station. SHR is applicable to a large number of further applications exhibiting similar characteristics.

We have already described a compelling application that can be developed using Sentire, *distributed HVAC component management*, and demonstrated how Sentire's lightweight actuator coordination mechanism can be used to increase the efficiency of this application. A similar application includes *distributed vehicle traffic management*. Many current vehicle navigation systems provide drivers with routes

based on both their destinations and the collection of live traffic data (e.g., reports of congestion and accidents) using roadside sensors and other data sources [54]. We envision next-generation traffic systems to evolve with two characteristics. First, we expect the development of distributed traffic management systems as a solution to administrative domains of control (e.g., the separate control of adjacent municipalities), in support of reduced and balanced network traffic load, and due to physical network partitioning. Furthermore, distributed control will avoid single points of failure in the face of large metropolitan areas with growing traffic systems. Second, we expect these systems to aim at globally reducing aggregate trip delay for a population of users rather than acting in an individualistic and greedy manner. For instance, if congestion is encountered on a roadway, such traffic should be re-routed so as to reduce the probability of creating new sources of congestion elsewhere in the system. Hence, similar to HVAC management, the separate entities composing the system (in this case, municipal jurisdictions) may exhibit *actuation* decisions that affect others' domains of control. A coordination solution similar to that offered by Sentire could be effectively applied to such an application. Beyond actuator coordination, the overall Sentire framework is a strong basis for developing such an application, although we concede that upgrades to the framework might be required.

Another application that Sentire may strongly influence is *precision agriculture*. Precision agriculture is a great application for WSNs alone. Small disposable sensors can be used to measure soil for levels of moisture, temperature, and chemicals (e.g., pesticides and fertilizers) and transmit this data back to a user. When directly mated with nearby actuators however, a more complete system can be formed in which sensor data can be used to automate decisions about when and where to treat soil with water, fertilizers, pesticides, etc. Such actuators may be in the form of mobile or strategically placed feeders or even mobile sprinkler head units connected to an irrigation system. Note that in such a control system, actuators' decisions may interfere with each other in multiple ways. For instance, sprinklers' treatment areas may overlap, causing accidental over-watering of some crops over time. Also, in many irrigation systems, there is a finite amount of water that must be effectively shared for distribution among crops or patches of soil. This poses a

similar coordination problem as that of HVAC component management, since any actuator's use of a finite resource (in this case water) will affect other actuators' abilities to perform their own control tasks. Therefore, Sentire may also apply to precision agriculture.

6.4 Future Research

Here, we discuss some major future research opportunities which have been identified in the course of this thesis study.

6.4.1 Data-centric Query Dissemination

SHR's opportunistic and priority-driven behavior has been shown to be effective for balancing and reducing WSN energy-usage and accommodating error-prone wireless links and faulty nodes. However, SHR is currently an address-centric protocol. This means that in order for data to be transmitted, a user must know the address of the exact sensor to query. This may not be a problem for small networks, but it can be for large networks. Requiring a user to maintain all sensors' addresses is not a scalable solution. Furthermore, for many WSN applications, a user will not care to query particular sensors. Instead, they will want to query the network for particular events or historical data characteristics. For instance, an architect or engineer may simply want to know where on a building's structure is there a level of stress beyond a certain force threshold. Also note that stress information will most likely be correlated among multiple sensors, with a sensor registering high stress being surrounded by sensors registering gradually weaker stress levels. Since there is the phenomenon of a *data gradient* in the network, which occurs in any application involving attenuating signals (e.g., those describing sound, heat, gas, etc.), SHR's techniques can be applied to efficiently guide queries to proper places in the network. For instance, a node can determine its priority to forward a query based on how close the characteristics of its sensor data is to that described in the query packet. Another solution can involve nodes using historical data attributes and trends to help probabilistically determine if they should forward a query on further even if their current sensor values may not indicate that they offer a good path to the data

in question. Yet another solution could be to follow the path of largest gradient, that is to have a node define its priority to forward a query based on having the largest (when the maximum is sought) or smallest (when the minimum is sought) data value of interest. In such a case, a query can ask for an extreme value, without having to know what the value actually is. Overall, extending SHR in this manner will help to intelligently guide queries within the network instead of blindly flooding them to all nodes. This increases the usability of SHR for applications that are more data-centric. This also helps reduce network traffic and energy consumption due to global floods of queries.

6.4.2 Game-theoretic SANET Actuator Coordination

This thesis's primary goal for supporting distributed actuator coordination was to provide a very simple and lightweight technique through which actuators could make decisions so as not to overuse a particular shared resource. We justify the use of this technique by our goal which was to use a solution that could accommodate the presence and behavior of many conflicting actuators, such as in the HVAC component management example. The major limitation of this technique is that it only provides a heuristic approach to determining how actuators can simultaneously perform tasks which can interfere with each others'. We believe that this technique is suitable for an application such as HVAC management, in which how an equilibrium point at which all actuators converge to a fair and *optimal* point is reached is often not critical to an application's goal(s). However, we concede that there may be other SANET applications which do (or *will*) require such an equilibrium point that can be reached and verified using a rigorous and proven method. Such applications might involve managing the movement and tasks of robotic drones, which often require very precise and coordinated commands. To address this in future research, we advocate the use of game theory, which is traditionally used to coordinate the strategic behavior of selfish, rational agents in conflict with each other. A central concept in game theory, and one which addresses the limitations of this thesis's approach, is the *Nash equilibrium*. The Nash equilibrium solution concept is used to find an application equilibrium point at which each conflicting agent is able to

execute its best strategy toward a specific goal without making other agents worse off. This is done primarily through an analysis of payoffs associated with each agent's set of available strategies. We believe game theory can be mapped to the actuator coordination problem and used by Sentire, however, there are important problems that must be addressed. For instance, SANET applications represent dynamic real-time systems. When game theory is applied, one must determine for how long a Nash equilibrium point is valid for the application. Also, any given game may possess zero, one, or many Nash equilibrium points. If none exist, an alternative method must be used, perhaps one similar to that presented in this dissertation. However, if many exist, there must be a policy such that every agent selects the same equilibrium point, and this is non-trivial. Another major challenge is that as the number of participating agents and available strategies grows, the Nash equilibrium becomes increasingly difficult to compute. Hence, an efficient technique must be employed to divide larger games into smaller ones and to iteratively find equilibrium points. Another method might involve identifying similarities between agents' (or in this case, *actuators'*) goals and establishing coalitions in order to shrink the number of unique players in a game.

6.4.3 Service-oriented Architectures for SANETs

We have demonstrated how Sentire can be used to develop interesting SANET applications and coordinate their actuator decisions in a lightweight manner. Sentire can have an enormous impact on the SANET software development community. However, its current implementation uses a very simplified technique to associate and interact with underlying sensors and actuators. This minimalist technique, which was not a significant focus of this thesis, was adequate in order to compose demonstrations exhibiting some of Sentire's more salient features. However, Sentire's, as well as other frameworks', usability would greatly benefit from a *standardized* model of interactivity with heterogeneous sensors and actuators. This involves addressing several notable challenges. One, there must be a standardized way of expressing what type of data a sensor can produce and what are the specific capabilities of a sensor or actuator. The former is self-explanatory, while the latter

may describe the quality of sensed data, available CPU processing power, or the amount of time or energy needed for an actuator to change an environment to a particular state. Two, there must be a standardized way of expressing how sensors and actuators can combine their capabilities in order to provide added value to a framework such as Sentire or directly to an end-application. This is critical since sensors must often operate as an aggregate in order to provide certain data transformations and make deeper inferences about environmental events. Just as important, actuators must know how to cooperate in order to treat environmental areas whose sizes exceed their own individual areas of influence. Three, there must a standardized language and protocol for interfacing with sensors and actuators. among other things, the transient existence that many wireless sensors and actuators will exhibit must be considered. To address the previous challenges, we advocate future research in service-oriented architectures (SOAs) for SANETs. Representing sensors and actuators as services can help establish appropriate abstractions and languages for addressing issues regarding representation, composition, and interaction among heterogeneous sensors and actuators. Research in SOA for SANETs also creates a rich basis for developing web services that are tuned for accessing live sensor data and actuator resources. Overall, this can lead to a standardized and systematic method of building Internet-scale sensor and actuator-enabled monitoring and control applications.

LITERATURE CITED

[1] A. Abramovici and J. Chapsky, *Feedback Control Systems: A Fast-track Guide for Scientists and Engineers*. Norwell, Massachusetts: Kluwer, 2000.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," in *Computer Networks*, vol. 38, no. 4, pp. 393-422, March 2002.

[3] I. F. Akyildiz and I. H. Kasimoglu, "Wireless sensor and actor networks: research challenges," in *Ad Hoc Networks Journal (Elsevier)*, vol. 2, no. 4, pp. 351-367, Oct. 2004.

[4] A. Anastasi, E. Borgia, M. Conti, E. Gregori, A. Passarella, "Understanding the real behavior of mote and 802.11 ad hoc networks: an experimental approach", in *Pervasive and Mobile Computing Journal*, vol. 1, no. 2, pp. 237-256, June 2005.

[5] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, and D. Rus, "Tracking a moving object with a binary sensor network," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pp. 150-161, Nov. 2003.

[6] S. Basse and A. V. Gelder, *Computer Algorithms: Introduction to Design and Analysis, 3rd Edition*. Reading, Massachusetts: Addison-Wesley, 2000.

[7] G. Biegel and V. Cahill, "A framework for developing mobile, context-aware applications," in *Proceedings of the Second IEEE Anuual Conference on Pervasive Computing and Communications*, pp. 362-365, March 2004.

[8] S. Biswas and R. Morris, "ExOR: opportunistic multi-hop routing for wireless networks," in *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp. 133-144, Aug. 2005.

[9] B. Blum, T. He, S. Son, and J. Stankovic, "IGF: A state-free robust communication protocol for wireless sensor networks," *Technical Report CS-2003-11*, University of Virginia Computer Science Department, 2003.

[10] G. Borriello and R. Want, "Embedded computation meets the World Wide Web," in *Communications of the ACM*, vol. 43, no. 5, pp. 59-66, May 2000.

[11] J. Branch, G. Chen, and B. Szymanski, "ESCORT: Energy-efficient Sensor network COmmunal Routing topology using signal quality metrics," in

Proceedings of the 4th International Conference on Networking, Springer-Verlag LNCS, vol. 3420, pp. 438-448, 2005.

[12] J. Branch, J. Davis, D. Sow, and C. Bisdikian, "Sentire: a framework for building middleware for sensor and actuator networks," in *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications Workshops (IEEE PerSeNS'05)*, pp. 396-400, March 2005.

[13] J. Branch, B. Szymanski, C. Bisdikian, N. Cohen, M. Ebling, J. Davis, and D. Sow, "Towards middleware components for distributed actuator coordination," in *Proceedings of the 3rd IEEE Workshop on Embedded Networked Sensors (IEEE EmNets'06)*, May 2006.

[14] J. Branch, M. Lisee, and B. Szymanski, "SHR: Self-Healing Routing for wireless ad hoc sensor networks," *unpublished*.

[15] A. Casimiro, J. Kaiser, and P. Verissimo, "An architectural framework and a middleware for cooperating smart components," in *Proceedings of the 1st Conference on Computing Frontiers*, pp. 28-39, April 2004.

[16] A. Cerpa and D. Estrin, "ASCENT: adaptive self-configuring sensor networks topologies," in *IEEE Transactions on Mobile Computing*, vol. 3, no. 3, pp. 272-285, July-Aug. 2004.

[17] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, "Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks," in *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, pp. 85-96, July 2001.

[18] D. Chen, J. Deng, and P. K. Varshney, "A state-free data delivery protocol for multihop wireless sensor networks," in *Proceedings of the IEEE Wireless Communications and Networking Conference*, March 2005.

[19] D. Chen, G. Cao, and L. Zuo, "A multihop data relay scheme for wireless networked sensors," in *Proceedings of the IEEE 6nd Semiannual Vehicular Technology Conference*, Sept. 2005.

[20] G. Chen, J. Branch, M. Pflug, L. Zhu, and B. Szymanski, "SENSE: A wireless sensor network simulator," in *Advances in Pervasive Computing and Networking*, B. Szymanski and B. Yener (editors), New York, Springer, LLC, 2004.

[21] G. Chen, J. Branch, and B. Szymanski, "Self-selective routing for wireless ad hoc networks," in *Proceedings of the IEEE International Conference on Wireless and Mobile Computing*, vol. 3, pp. 57-65, August 2005.

[22] G. Chen, J. Branch, and B. Szymanski, "A self-selection technique for flooding and routing in wireless ad hoc networks," in *Journal of Network and Systems Management*, vol. 14, no. 3, pp. 359-380, 2006.

[23] D. A. Coffin, D. J. Van Hook, S. M. McGarry, and S. R. Kolek, "Declarative ad-hoc sensor networking," in *Proceedings of SPIE Integrated Command Environments*, pp. 109-120, Nov. 2000.

[24] Crossbow Technologies, Inc., http://www.xbow.com.

[25] B. Deb, S. Bhatnagar, and B. Nath, "ReInForM: reliable information forwarding using multiple paths in sensor networks," in *Proceedings of the 28th Annual Conference on Local Computer Networks*, pp. 406-415, Oct. 2003.

[26] W3C Document Object Model, http://http://www.w3.org/DOM/.

[27] *Proceedings of the Distributed Sensor Nets Workshop*, 1978.

[28] Extensible Markup Language (XML), http://www.w3.org/XML/.

[29] Y. Fan, G. Zhong, S. Lu, and L. Zhang, "GRAdient Broadcast: a robust data delivery protocol for large scale sensor networks," in *ACM Wireless Networks*, vol. 11, no. 2, March 2005.

[30] T. He, B. Krogh, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, and J. Hui, "Energy-efficient surveillance system using wireless sensor networks," in *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services*, pp. 270-283, June 2004.

[31] W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo, "Middleware to support sensor network applications," in *IEEE Network*, vol. 18, no. 1, pp. 6-14, Jan./Feb. 2004.

[32] M. Heissenbuttel and T. Braun, "A novel position-basd and beacon-less routing algorithm for mobile ad-hoc networks," in *Proceedings of the 3rd Workshop on Applications and Services in Wireless Networks*, pp. 197-210, July 2003.

[33] M. Heissenbuttel, T. Braun, M. Walchli, and T. Bernoulli, "Broadcasting in wireless multihop networks with the dynamic forwarding delay concept," *Technical Report IAM-04-010*, University of Bern Institute of Computer Science and Applied Mathematics, Dec. 2004.

[34] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, "System architecture directions for networked sensors," in *Proceedings of Architectural Support for Programming Languages and Operating Systems*, pp. 93-104, Nov. 2000.

[35] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," in *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 2-16, Feb. 2003.

[36] D. B. Johnson, D. A. Maltz, and Y. C. Hu, "The dynamic source routing protocol for mobile ad hoc networks (DSR)," *IETF Mobile Ad Hoc Networks Working Group*, Internet Draft, work in progress, April 2003.

[37] J. M. Kahn, R. H. Katz, and K. S. J. Pister, "Emerging challenges: mobile networking for smart dust," in *Journal of Communications and Networks*, vol. 2, no. 3, pp. 188-196, Sept. 2000.

[38] V. A. Kottapalli, A. S. Kiremidjian, J. P. Lynch, E. Carryer, T. W. Kenny, K. H. Law, and Y. Lei, "Two-tiered wireless sensor network architecture for structural health monitoring," in *Proceedings of the 10th Annual International Symposium on Smart Structures and Materials*, March 2003.

[39] L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis, "Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea," in *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, pp. 64-75, Nov. 2005.

[40] B. C. Kuo and F. Golnaraghi, *Automatic Control Systems*. New York, New York: John Wiley and Sons, Inc., 2003.

[41] D. Lal, A. Manjeshwar, F. Herrmann, E. Uysal-Biyikoglu, and A. Keshavarzian, "Measurement and characterization of link quality metrics in energy constrained wireless sensor networks," in *Proceedings of the IEEE Global Telecommunications Conference*, pp. 446-452, Dec. 2003.

[42] LEGO Mindstorms Home, http://mindstorms.lego.com.

[43] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler, "The emergence of networking abstractions and techniques in TinyOS," in *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation*, pp. 1-14, March 2004.

[44] T. Liu and M. Martonosi, "Impala: a middleware system for managing autonomic, parallel sensor systems," in *Proceedings of the 9th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 107-118, June 2003.

[45] T. Liu, C. M. Sadler, P. Zhang, M. Martonosi, "Implementing software on resource-constrained mobile sensors: experiences with Impala and Zebranet," in *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services*, pp. 256-269, June 2004.

[46] K. Lorincz, D. Malan, T. R. F. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, S. Moulton, and M. Welsh, "Sensor networks for emergency response: challenges and opportunities," in *IEEE Pervasive Computing*, Special Issue on Pervasive Computing for First Response, vol. 3, no. 4, pp. 16-23, Oct.-Dec. 2004.

[47] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pp. 88-97, Sept. 2002.

[48] A. Manjeshwar and D. P. Agrawal, "TEEN: a routing protocol for enhanced efficiency in wireless sensor networks," in *Proceedings of the 15th International Parallel and Distributed Processing Symposium*, pp. 2009-2015, April 2001.

[49] A. Manjeshwar and D. P. Agrawal, "APTEEN: a hybrid protocol for efficient routing and comprehensive information retireval in wireless sensor networks," in *Proceedings of the International Parallel and Distributed Processing Symposium*, pp. 195-202, April 2002.

[50] T. Melodia, D. Pompili, V. C. Gungor, and I. F. Akyildiz, "A distributed coordination framework for wireless sensor and actor networks," in *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pp. 99-110, May 2005.

[51] M. Modahl, I. Bagrak, M. Wolenetz, P. Hutto, U. Ramachandran, "MediaBroker: an architecture for pervasive computing," in *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications*, pp. 253-262, March 2004.

[52] Moteiv Corporation, http://www.moteiv.com.

[53] C. Myers, A. Oppenheim, R. Davis, and W. Dove, "Knowledge-based speech analysis and enhancement," presented at the International Conference on Acoustics, Speech, and Signal Processing, 1984.

[54] XM NavTraffic, http://www.xmradio.com/xmnavtraffic.

[55] L. Nachman, R. Kling, R. Adler, J. Huang, and V. Hummel, "The Intel mote platform: a bluetooth-based sensor network for industrial monitoring," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, pp. 437-442, April 2005.

[56] E. Nett and S. Schemmer, "An architecture to support cooperating mobile embedded systems," in *Proceedings of the 1st Conference on Computing Frontiers*, pp. 40-50, April 2004.

[57] E. C. H. Ngai, M. R. Lyu, and J. Liu, "A real-time communications framework for wireless sensor-actuator networks," in *Proceedings of the 2006 IEEE Aerospace Conference*, March 2006.

[58] J. Paek, K. Chintalapudi, J. Cafferey, R. Govindan, and S. Masri, "A wireless sensor network for structural health monitoring: performance and experience," in *Proceedings of the Second IEEE Workshop on Embedded Networked Sensors*, May 2005.

[59] G. Pei, M Gerla, and T-W. Chen, "Fisheye state routing in mobile ad hoc networks," in *Proceedings of the International Conference on Distributed Computing Systems Workshop on Wireless Networks and Mobile Computing*, pp. D71-D78, April 2000.

[60] C. E. Perkins and P. Bhagwat, "Highly dynamic destination sequenced distance vector routing (DSDV) for mobile computers," in *Proceedings of ACM Conference on Communications Architectures, Protocols, and Applications (SIGCOMM'94)*, pp. 234-244, Aug.-Sept. 1994.

[61] C. E. Perkins, E. M. Royer, and S. R. Das, "Ad hoc on demand distance vector routing (AODV)," *IETF Internet Draft*, draft-ietf-manet-aodv-09.txt, work in progress, Nov. 2001.

[62] J. Polastre, R. Szewczyk, C. Sharp, and D. Culler, "The mote revolution: low power wireless sensor network devices," in *Proceedings of Hot Chips 16: A Symposium on High Performance Chips*, Aug. 2004.

[63] R. Poor, "Gradient routing in ad hoc networks," unpublished.

[64] T. Rappaport, *Wireless Communications: Principles and Practice, Second Edition*, Upper Saddle River, New Jersey: Prentice-Hall, Inc., 2001.

[65] C. Santivanez, R. Ramanathan, and I. Stavrakakis, "Making link-state routing scale for ad hoc networks," in *Proceedings of the 2001 ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pp. 22-32, Oct. 2001.

[66] C. Schurgers, V. Tsiatsis, S. Ganeriwal, M. B. Srivastava, "Topology management for sensor networks: exploiting latency and density," in *Proceedings of the Symposium on Mobile Ad Hoc Networking and Computing*, pp. 135-145, June 2002.

[67] R. C. Shah and J. M. Rabaey, "Energy aware routing for low energy ad hoc sensor networks," in *Proceedings of the Wireless Communications and Networking Conference*, pp. 350-355, March 2002.

[68] J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, M. Seltzer, and M. Welsh, "Hourglass: an infrastructure for connecting sensor networks and applications," *Harvard Technical Report TR-21-04*, 2004.

[69] SimJava, http://www.dcs.ed.ac.uk/home/hase/simjava/.

[70] P. Skraba, H. Aghajan, and A. Bahai, "Distributed Passive Routing Decisions in mobile ad-hoc networks," in *Proceedings of the IEEE 60th Vehicular Technology Conference*, vol. 4, pp. 26-29, Sept. 2004.

[71] R. F. Sproull and D. Cohen, "High-level protocols," in *Proceedings of the IEEE*, vol. 66, pp. 1371-1386, Nov. 1978.

[72] L. St. Ville and P. Dickman, "Garnet: a middleware architecture for distributing data streams originating in wireless sensor networks," in *Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops*, pp. 235-240, May 2003.

[73] K. Wasilewski, J. Branch, M. Lisee, and B. Szymanski, "Self-Healing Routing: a study in efficiency and resiliency of data delivery in wireless sensor networks," to appear in *Proceedings of the SPIE Defense and Security Symposium*, April 2007.

[74] M. Weiser, "The computer for the 21st century," in *Scientific American*, vol. 265, no. 3, Sept. 1991.

[75] Y. Xu, J. Heidemann, and D. Estrin, "Adaptive energy-conserving routing for multihop ad hoc networks," *USC/ISI Research Report 527*, Oct. 2000.

[76] Y. Xu, J. Heidemann, and D. Estrin, "Geography-informed energy conservation for ad hoc routing," in *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, pp. 70-84, July 2001.

[77] Y. Xu, S. Bien, Y. Mori, J. Heidemann, and D. Estrin, "Topology control protocols to conserve energy in wireless ad hoc networks," *CENS Techincal Report 0006*, Jan. 2003.

[78] Y. Xu, W. Lee, J. Xu, and G. Mitchell, "PSGR: Priority-based Stateless Geo-Routing in wireless sensor networks," in *Proceedings of the IEEE Conference on Mobile Ad-hoc and Sensor Systems*, Nov. 2005.

[79] E. Yoneki and J. Bacon, "A survey of wireless sensor network technologies: research trends and middleware's role," *Technical Report UCAM-CL-TR646*, University of Cambridge, 2005.

[80] Y. Yu, B. Krishnamachari, and V. K. Prasanna, "Issues in designing middleware for wireless sensor networks," in *IEEE Network*, vol. 18, no. 1, pp. 15-21, Jan./Feb. 2004.

[81] F. Zhao and L. Guibas, *Wireless Sensor Networks: an Information Processing Approach*, San Francisco, California: Elsevier, Inc., 2004.

[82] J. Zhao and R. Govindan, "Understanding packet delivery performance in dense wireless sensor networks," in *Proceedings of the 1st International Conference in Embedded Networked Sensor Systems*, pp. 1-13, Nov. 2003.

[83] ZigBee Alliance, http://www.zigbee.org.

[84] M. Zorzi and R. R. Rao, "Geographic random forwarding (GeRaF) for ad hoc and sensor networks: energy and latency performance," in *IEEE Transactions on Mobile Computing*, vol. 2, no. 4, pp. 349-365, Oct.-Dec. 2003.

[85] M. Zorzi and R. R. Rao, "Geographic random forwarding (GeRaF) for ad hoc and sensor networks: multihop performance," in *IEEE Transactions on Mobile Computing*, vol. 2, no. 4, pp. 337-348, Oct.-Dec. 2003.