DIRECTED DIFFUSION: AN APPLICATION-SPECIFIC AND DATA-CENTRIC COMMUNICATION PARADIGM FOR WIRELESS SENSOR NETWORKS

by

Chalermek Intanagonwiwat

A Dissertation Presented to the

FACULTY OF THE GRADUATE SCHOOL

UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillment of the

Requirements for the Degree

DOCTOR OF PHILOSOPHY

(COMPUTER SCIENCE)

December 2002

Acknowledgments

First of all, I would like to express my deepest gratitude and appreciation to my advisors, Professor Deborah Estrin, Professor Ramesh Govindan, and Professor John Heidemann for their unlimited patience, constant encouragement, vision, inspiration, and guidance throughout my study. I was extremely fortunate to have three advisors who have been the endless source of knowledge and ideas. Without them, this thesis would not have been possible. I would also like to thank the other members of my dissertation committeee, Professor Maja Mataric and Professor Daniel Lee for their valuable time, comments, feedbacks, guidances, and interest in this work.

It has been a great pleasure to work with smart and nice people at USC/ISI. I had the privilege of interacting with members of *dgroup* and *ngroup* at the USC Computer Networks and Distributed Systems Research Laboratory, especially Nirupama Bulusu, Alberto Cerpa, Xuan Chen, Jeremy Elson, Lewis Girod, Deepak Ganesan, Xinming He, Ahmed Helmy, Eddie Hsu, Bau-Yi Polly Huang, Amit Kumar, Satish Kanna Kumar, Khun-Chan Lan, Art Mena, Graham Phillips, Pavlin Ivanov Radoslavov, Reza Rajaie, Anoop Reddy, Puneet Sharma, Fabio Silva, Mohit Talwar,

Hongsuda Tangmunarunkit, Ya Xu, Shuqian Yan, Wei Ye, Haobo Yu, Yan Yu, and Yonggang (Jerry) Zhao. I thank them for their advice, feedback, and friendship.

I am grateful to Jongsuk Ahn, Cengiz Alaettinoglu, Steven Berson, Ted Faber, Ashish Goel, Padma Haldar, Dongho Kim, Robert Lindell, John Mehringer, Christos Papadopoulos, Cauligi Raghavendra, Craig Rogers, Andrew Swartzbaugh, and Rui Wang, members of the Computer Networks Division at ISI, for their comments, time, and office space used in my testbed experiments.

I was very fortunate to have the opportunity to learn from many other people as well. I have learnt a lot about system perspective from Son Dao, Dante De Lucia, and other HRL research staffs during my summer internship in 1998. Our collaborations were my wonderful experiences professionally and educationally. My gratitude extends to Professor George Bekey for his advice and time during my master year. I would also like to thank Van Jacobson for his vision for the future. His suggestion during a DARPA ISAT meeting in 1998 has tremendously inspired this work.

Also, I wish to thank Arnold Diaz, Sandy Ginoza, Dawn Johnson, Joe Kemp, Lisa Moses, Sungita Patel, Alba Regalado-Palaci, Melissa Snearl-Smith, and Amy Yung for their assistance and friendship.

I am thankful to the Defense Advanced Research Projects Agency for their financial support through the SCADDS project at ISI under Cooperative Agreement No. DABT63-99-1-0011.

Special thanks are due to all my Thai friends and roommates who have made my stay in Los Angeles more meaningful and pleasant. Last but not least I would like

to thank my family for their love, encouragement, assistance, understanding, and support. I am very grateful to them.

Contents

List Of Figures

Abstract

Advances in radio, sensor, and VLSI technology will enable small and inexpensive sensor nodes capable of wireless communication and significant computation. Large-scale networks of such sensors may require novel data dissemination paradigms which are scalable, robust, and energy-efficient. In this dissertation, we design and evaluate *directed diffusion*, one such paradigm for distributed sensing applications in wireless sensor networks. Directed diffusion incorporates attribute-based naming, reinforcement-based adaptation, data-centric routing, and application-specific processing inside the network (*e.g.,* data aggregation). By using attributes with external meaning at the lowest levels of communication, diffusion avoids multiple levels of name binding common to other approaches. Attribute-based naming in turn enables in-network processing with filters, supporting data aggregation, nested queries, and similar techniques that are critical to reducing network traffic and conserving energy. Given that, in wireless sensor networks, the communication cost is several orders of magnitude higher than the computation cost, directed diffusion can achieve significant energy savings with in-network data aggregation. We evaluate the energy efficiency of directed diffusion analytically and experimentally.

We also propose two instantiations of directed diffusion with different aggregation schemes: *opportunistic aggregation* and *greedy aggregation*. In the former approach, data is opportunistically aggregated at intermediate nodes on a low-latency tree. Our evaluation indicates that the opportunistic approach can achieve significant energy savings and can outperform idealized traditional schemes even with relatively unoptimized path selection. In the greedy approach, a greedy incremental tree is constructed to improve path sharing for more energy savings. Our result suggests that although greedy aggregation and opportunistic aggregation are roughly equivalent in low-density and medium-density networks, greedy aggregation can achieve significant energy savings in high-density networks without adversely impacting latency or robustness.

Chapter 1

Introduction

It is common to see embedded systems in several aspects of people's lives. Embedded systems control not only transportation and communication infrastructures, but also appliances in home and office environments (*e.g.,* refrigerators, VCRs, TVs, microwave ovens). Embedded systems are the first step toward ubiquitous computing [Wei91] whereby various computing elements are so seamlessly integrated into the environment that they will be invisible to common awareness.

This vision is increasingly likely to be realized with recent advances in CMOS IC, wireless communication, and MEMS [Cen] technology. Such advances have already led to dramatic reductions in size, power consumption, and circuitry cost. Various functions (*e.g.,* sensing, signal processing) can now be integrated into a single wireless node. Coordination and communication among such nodes will not only enable seamless computing but also revolutionize information technology, especially applications related to sensing and controlling physical environments. Small active devices or sensors can coordinate to perform larger sensing tasks (*i.e.,* distributed

micro-sensing tasks), which could not have been achieved with individual node capabilities. Several thousands of such devices may be deployed in hostile dynamic environments (*e.g.,* toxic terrain) or in more benign, but less accessible, environments (*e.g.,* large complex industrial plants, aircraft interiors).

In distributed micro-sensing systems, sensors can be dispersed throughout a terrain to collect information about the physical environment. Users may be interested in receiving certain collected information from a particular region or from the entire network (*e.g.,* a periodic location of an animal in region A). The users may express their *interest* simply by querying information *sinks*, which could be any sensors nearby. The specified queries need to be delivered to the sensors in the specified region, which will be *tasked* to collect information. Once such sensors detect the animal, they might coordinate with one another to triangulate the animal's location and disseminate it back to the sinks or the users. In this thesis, we propose *directed diffusion* [IGE00] for efficient and robust dissemination of query and information (Van Jacobson suggested the concept of "diffusing" attribute named data for this class of applications that later led to the design of directed diffusion).

In directed diffusion, a node requests data by sending interests for named data. Interest propagation leaves traces (or direction state) so that data, which match the interest, can be "drawn" toward that node. Intermediate nodes can process (*e.g.,* cache, aggregate, transform) or direct data and interests using application knowledge to conserve system resources (*e.g.,* energy, bandwidth).

Using the directed diffusion paradigm, the example application might be implemented as follows. The query would be transformed into an interest and *diffused* (*e.g.,* broadcasted, geographically routed) toward nodes in region A. When a node

in region A receives the interest, it activates its sensors to start collecting information about the animal. When the sensors detect the animal's location, the node disseminates the information by reversing the path of interest propagation. Intermediate nodes along the path might *aggregate* the data to reduce the total data size for energy savings.

Directed diffusion is dramatically different from IP-style communication whereby inter-node communication is layered on an end-to-end delivery service, and packets are associated with node identifiers of end-points. Unlike traditional networks, diffusion-based sensor networks are data-centric. In directed diffusion, communication primitives are expressed in terms of named data rather than node addresses. Attributes of the sensed event are used to name data in directed diffusion. As a result, data can be de-coupled from producers to achieve more robust application design by reducing reliance on individual node reachability. Even if the producer dies, its generated data cached in the other sensors may still be accessible. Furthermore, directed diffusion does not simply route data and queries, but rather disseminates them; by this we mean intermediate nodes can locally aggregate, transform, or cache data and queries for system resource savings. An important feature of directed diffusion is that interest and data propagation (and aggregation) is determined by localized interactions (message exchanges between neighbors or nodes within some vicinity).

1.1 Sensor Nodes

Many recent advances in chip integration technology will enable matchbox sized sensor nodes equipped with a battery, a power-conserving CPU (several hundred MHz), a memory (several tens of Mbytes) [KKP99], a wireless device using a diversity coding scheme [Haa00], and an energy efficient MAC (*e.g.,* TDMA). Each node will be capable of running a stripped-down version of a modern operating system (*e.g.,* Windows CE, uCLinux). Sensors (*e.g.,* seismic geophones, infrared dipoles, electret microphones for acoustic sensing) will be integrated into each node for monitoring various physical conditions (*e.g.,* temperature, pressure, humidity, motion, noise, light). Each node will possibly contain a fully functional GPS receiver and an analog-to-digital conversion system, which can produce $\leq$ 70 ksamples per second at $\leq$ 12-bit resolution (see also our testbed in Appendix A).

For power conservation reasons, some common signal processing functions may be separated from the main unit and moved to a low power ASIC so that the main processor needs to be woken up only when interesting events occur [PK00].

1.2 Sensor Networks

Given that sensor nodes in the future will be small and inexpensive, they can be densely deployed (*i.e.,* < 100 feet between two neighbors) in an unplanned fashion near the phenomena to be sensed (*e.g.,* at busy intersections or in the interior of large machinery). The advantage of such sensor networks is that, even with relatively inexpensive sensors, a high signal per noise ratio (SNR) can be obtained (given that the SNR decreases rapidly with distance). With such dense deployment, an

individual sensor node may not have to frequently perform multi-target resolution to distinguish among different targets (*e.g.,* animals, vehicles). Such multi-target resolution may require complex de-convolution algorithms and non-trivial processing capabilities.

Conversely, current sensor deployments can be classified into two approaches: a remote approach and a centralized approach. In the remote approach, large, complex sensor systems are usually deployed far away from the phenomena to be sensed. Complex signal processing algorithms are used to separate targets from environmental noise. However, even with such complex algorithms, the dramatically increased noise due to the long distance can still limit the performance of the systems.

Alternatively, centralized sensor networks are carefully deployed in the field, but each sensor node can not locally process sensed phenomena signals. Instead, time series of the signals are transmitted to some central nodes for computation and interpretation. However, transmitting such a large amount of data from the sensors, which detect the signals, to some central nodes via either long-range or hop-by-hop wireless communication may not be energy efficient. For this reason, such networks often must be wired, thereby constraining deployment.

Sensor nodes are likely battery-powered and expected to last for several days. Given that energy efficiency is a crucial requirement, short-range hop-by-hop communication is preferred over direct long-range communication due to its lower energy consumption. Furthermore, such hop-by-hop communication also provides communication diversity for communicating around obstacles [PK00]. To conserve energy, local computation is suggested for reducing data before transmission [PK00].

Such an energy efficiency requirement, coupled with computation and communication capability in future sensor nodes, justifies a different organization of a sensor network. In this organization, individual nodes would reduce the sensed phenomena signals into relatively coarse "event" descriptions. These descriptions usually contain a "codebook" value (*i.e.,* an event code) for the target, a timestamp, a signal level, and a confidence degree of estimation. Nodes can then exchange these event descriptions with their neighbors (which have also detected the target) to refine the estimation and transmit only a short description back to a user [EGHK99, IGE00, HSI+01].

With such organization, a sensor network becomes a distributed computing system. Even though it is feasible to design these sensor networks using IP and ad-hoc routing, a different set of communication primitives may lead to more efficient sensor data dissemination.

1.3 Communication for Sensor Networks

In the example application, sensor networks might practically consist of hundreds or thousands of sensor nodes. Even though the sensor nodes may be deployed in a regular fashion (*e.g.,* a 2-dimensional lattice or a linear array), often they will not be and so communication protocols can not assume structured sensor fields.

A user would be able to contact (using, possibly a long-range radio) one of the sensors in the field and to pose the following *task*: "Every I ms for the next T seconds, send me a location estimate of any four-legged animal in sub-region R of the sensor field". Generally, the network may support several task types. However, sensor networks are *task-specific*; the task types are known when the sensor network

is deployed (sensor networks may be reprogrammable and the tasks they support may change slowly over time.). Thus, directed diffusion is designed with task specificity in mind.

The task will be delivered to sensor nodes in sub-region R using mechanisms described in the next chapter. Each node in sub-region R will task its sensors to collect samples and match the samples against a locally stored library. If the sampled signals match those of a four-legged animal, the node generates F event descriptions a second. Each description contains an estimate of the animal's location, a codebook value of the animal, an intensity of the signal, and a confidence degree of the estimate. Sensors within region R may coordinate to select the best estimate, which will be delivered to the user.

This thesis focuses on the design of task and event dissemination mechanisms for wireless sensor networks, which support multiple concurrent task initiations of the specified type. However, directed diffusion is also applicable to other types of distributed sensor coordination as discussed in the next chapter. The main challenges in designing such mechanisms are

Scalability: These data dissemination systems must scale to several thousands of sensor nodes. Their overhead needs to be minimized for promoting scalability. Minimizing or eliminating manual configuration would simplify deployment.

Robustness: Sensor nodes may move, lose battery power, or temporarily fail to communicate because of environmental factors. Some nodes may be added or removed and wireless communication may not be reliable. Given that such network dynamics are common events, the data dissemination systems need to be adaptive and robust to these kinds of changes.

Energy efficiency: It has been envisioned that wireless sensor nodes would be battery-powered. The data dissemination systems must operate energy-efficiently by minimizing overhead and balancing load for long network lifetime.

1.4 Contributions

Our contribution in this thesis is in the design and evaluation of *directed diffusion*, a novel communication paradigms for large sensor networks, in the presence of various network dynamics.

Directed diffusion supports data-centric routing and application-specific processing inside the network (*e.g.,* data aggregation). Given that the amount of energy for transmitting 1Kb over a distance of 100 meters could be used for executing 3 million instructions on a general-purposed unoptimized processor with 100MIPS/W power, the communication cost is several orders of magnitude higher than the computation cost in wireless sensor networks [PK00]. Therefore, directed diffusion can achieve significant energy savings with in-network data aggregation. We design two diffusion instantiations with different aggregation schemes and evaluate the energy efficiency of directed diffusion analytically and experimentally (over our operational testbed and the *ns-2* simulator [BEF+00]).

We analyze the data delivery cost for directed diffusion and two idealized schemes: *omniscient multicast* and *flooding*. This analysis serves to sanity check the intuition behind directed diffusion and highlights some of the differences between diffusion and the other approaches. Our results suggest that, as the number of sources and sinks

increases, the cost savings due to in-network processing (*e.g.,* duplicate suppression) of diffusion become more evident.

To verify and complement our analytic evaluation, we implement our animal tracking instance of diffusion in the *ns-2* simulator. Particularly, we compare the performance of diffusion against those idealized schemes and study the sensitivity of directed diffusion performance to the choice of parameters. This packet-level simulation is also used for exploring the impact of network dynamics and the influence of the radio MAC layer on diffusion performance.

Additionally, we validate such results with an actual implementation of our tracking application. In particular, we examine in-network aggregation in our ISI testbed of 14 PC/104 sensor nodes. In one experiment, data aggregation reduces network traffic by up to 42%.

Furthermore, we propose two instantiations of directed diffusion with different aggregation schemes: *opportunistic aggregation* and *greedy aggregation*. In the opportunistic approach, data is opportunistically aggregated at intermediate nodes on a low-latency tree. Our evaluation indicates that the opportunistic approach can achieve significant energy savings and can outperform idealized traditional data dissemination schemes (*i.e.,* omniscient multicast and flooding) even with relatively unoptimized path selection.

In the greedy approach, a greedy incremental tree is constructed to improve path sharing for more energy savings. We evaluate the performance of this greedy approach by comparing it to the opportunistic approach. Our result suggests that although greedy aggregation and opportunistic aggregation are roughly equivalent in low-density and medium-density networks, greedy aggregation can achieve significant

energy savings in high-density networks (the number of neighbors > 40). In one experiment we find that greedy aggregation can achieve up to 45% energy savings over opportunistic aggregation without adversely impacting latency or robustness.

Other contributions in this dissertation include evaluation metrics, tradeoffs, simulation platforms, test suites, applications, requirements, challenges, and insights into the design of data dissemination systems for wireless sensor networks. Specifically, our evaluation metrics are average dissipated energy, average delay, and distinct event delivery ratio. These metrics indicate the overall lifetime of sensor nodes, the temporal accuracy of the estimates, and the robustness of the system. The challenge is to design a system that is long-lived but still accurate and robust. Given that sensor networks are task-specific, we describe our design using the animal tracking application as an example. We implement directed diffusion on the *ns-2* network simulator. The code is downloadable from `http://www.isi.edu/nsnam/ns`.

1.5 Dissertation Organization

Directed diffusion is described in Chapter 2. We explain its key features (*i.e.,* data naming, interest propagation, gradient establishment, data dissemination, reinforcement, and adaptation to network dynamics) and some details of an opportunistic diffusion instantiation for an animal tracking sensor network. We specify the local rules to achieve these features, demonstrate differences between directed diffusion and traditional networking, and qualitatively argue that the paradigm provides scalability, robustness, and energy efficiency. We conduct an analytic evaluation of directed diffusion in Section 3.1. The paradigm is also quantitatively evaluated over our

operational testbed (Section 3.3) and a detailed packet-level simulator (Section 3.2). We explain the details of the greedy instantiation which we have implemented in *ns-2* and compare it to our opportunistic instantiation in Chapter 4. Directed diffusion has been informed and influenced by a variety of other research efforts (Chapter 5). This dissertation ends with conclusion remarks and future directions in Chapter 6. We summarize the ISI implementation of diffusion in Appendix A. The pseudo code of the implementation is also included (Appendix B).

Chapter 2

Directed Diffusion

Directed diffusion consists of several elements: interests, data messages, gradients, and reinforcements. An *interest* message is a query or an interrogation which specifies what a user wants. Each interest message contains a description of data interested by a user. Typically, *data* in sensor networks is the collected or processed information of a phenomenon which matches an interest or a request of a user. Such data can be an *event* which is a short description of the sensed phenomenon. In directed diffusion, data is named using attribute-value pairs. The interest is disseminated throughout the sensor networks to "draw" named data toward the user. Interest propagation establishes gradients within the network for data propagation. Specifically, a *gradient* is a direction state created inside each node which receives an interest. The gradient direction is set toward the neighboring node from which the interest is received. Events are propagated toward the interest originators along multiple gradient paths. The sensor network *reinforces* one or a small number of these paths (Figure 2.1).

In this section, we describe these elements of directed diffusion for a location tracking sensor network (see also Section 1.3). Even with this specific instantiation of diffusion, there are several possible design choices. Our initial evaluation (Chapter 3) only focuses on a subset of these choices. Different design choices result in different variants of diffusion (see also Chapter 4 for another variant). Moreover, even though we describe this diffusion variant for rate-based applications, diffusion also works for event-triggered applications.

2.1 Naming

In directed diffusion, tasks are described or named using attribute-value pairs. A simplified description of the animal tracking task in Section 1.3 might be (see also Section 2.2 for more detailed description):

```
type = four-legged animal    // detect animal's location
interval = 10 ms             // send back events every 10 ms
duration = 10 minutes        // for the next 10 minutes
rect = [-100, 100, 200, 400] // from sensor nodes within rectangle
```

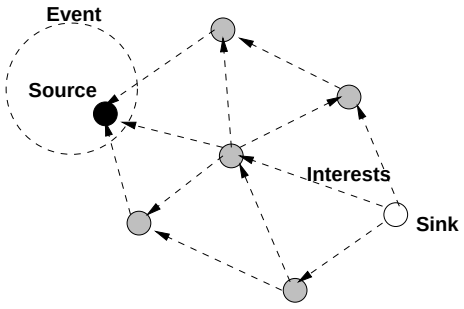
As an example of the sub-region representation, a rectangle is selected and based on some coordinate system (in practice, possibly based on GPS coordinate system).

Since the task description specifies an interest for data (matching the attributes), such a task description is also referred as an *interest*. The data is also named using the similar naming scheme. For example, a sensor that detects an animal might generate the following data (see Section 2.3 for more explanation).

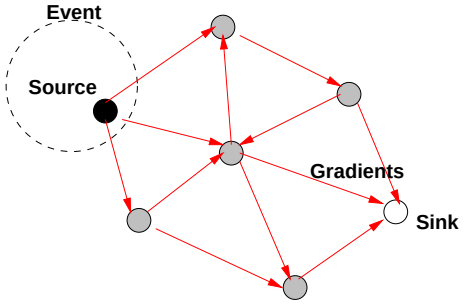
```
type = four-legged animal  // detect animal's location
instance = elephant        // instance of this type
location = [125, 220]      // estimated location
intensity = 0.6            // signal amplitude measure
confidence = 0.85          // confidence in the estimate
timestamp = 01:20:40       // event generate time
```

Given a set of supported tasks, selecting a naming scheme is the first step in the design of directed diffusion. For the attribute-value based naming scheme, each attribute is associated with a value range. For example, the range of the `type` attribute is the set of codebook values representing mobile objects (*e.g.,* vehicles, animals, humans). The attribute value can be any subset of its range. The `type` attribute value in the example is the codebook value representing four-legged animals.

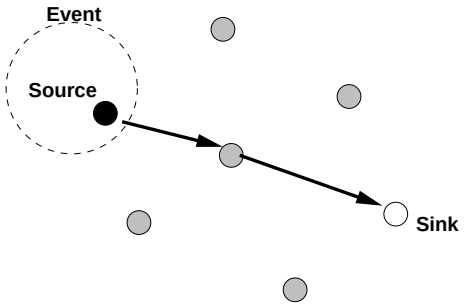
There are other choices for an arrangement of attribute-value pairs (*e.g.,* hierarchical) and other naming schemes (*e.g.,* intentional naming [AWSBL99]). To some extent, the choice of naming scheme and arrangement can affect the expressivity of tasks and may impact diffusion performance. For example, a hierarchical arrangement of attribute-value pairs could narrow down the search space during name resolution, and simplify name specifiers for better understanding. However, in this thesis, the primary objective is to gain an initial understanding of the directed diffusion paradigm. The effect of naming schemes and arrangements is a subject of future research. Even though the SCADDS group has done some of that research [HSI+01] (see also Appendix A), much more work is still needed.



(a) Interest propagation



(b) Initial gradients set up



(c) Data delivery along reinforced path

Figure 2.1: A simplified schematic for directed diffusion.

2.2 Interests and Gradients

The named task description (*i.e.,* the interest) is usually injected into the network at some (possibly arbitrary) node (the *sink*). The sink node creates a task state which will be purged after the time indicated by the `duration` attribute.

For each active task, the sink periodically *broadcasts* an interest to all its neighbors (more efficient methods to send the interest will be discussed later). This initial interest also contains the attributes described in Section 2.1. The `interval` attribute specifies an event data rate (Hence, the data rate is 100 events per second in that example). However, unlike that example, the `interval` attribute in this initial interest is much larger. As *exploratory*, the initial interest is intended to determine if there are any sensor nodes that detect the four-legged animal. Hence, the initial exploratory interest specifies a low data rate (*e.g.,* 1 event per second). This is not the only choice, but represents a performance tradeoff. Since the location of the sources is not precisely known, interests must necessarily be diffused over a broader section of the sensor network than that covered by the potential sources. As a result, if the sink had chosen a higher initial data rate, a higher energy consumption might have resulted from the wider dissemination of sensor data. However, with a higher initial data rate, the time to achieve high fidelity tracking is reduced. The desired higher data rate can be achieved by reinforcement as described in Section 2.4. Thus, the initial interest is described as follows (For event-triggered applications, the initial interest does not contain the `interval` attribute because of no explicit data rate).

```
type = four-legged animal
interval = 1 s
```

```
rect = [-100, 200, 200, 400]
timestamp = 01:20:40          // hh:mm:ss
expiresAt = 01:30:40
```

Before we describe how interests are processed, we emphasize that the interest is soft state [Jac90, SEFJ97, WTZ99] that will be periodically *refreshed* by the sink. Periodic interests from sinks are necessary because the interests are not reliably transmitted. To periodically refresh the interest, the sink simply re-sends the same interest with a monotonically increasing `timestamp` attribute. The refresh rate is a protocol design parameter that trades overhead for increased robustness to lost interests.

Every node maintains an interest cache. Each item in the cache corresponds to a distinct interest. Two interests are *distinct*, in our example, if their `type` attribute differs, or their `rect` attributes are (possibly partially) disjoint. Otherwise, these two interests *match* each other. Interests *do not contain information about the sink* but just about the immediately previous hop. Thus, interest state scales with the number of distinct active interests. Our definition of distinct interests also allows *interest aggregation*. Two interests I_1 and I_2 (with identical types and completely overlapping `rect` attributes but different `interval` attributes) can (in some situations) be represented with a single interest entry. Other interest aggregation is a subject of future research.

An interest entry in the cache consists of several fields. A timestamp field specifies the timestamp of the last received matching interest. The interest entry also contains several gradient fields (up to one per neighbor). Each gradient contains a data rate field (requested by the corresponding neighbor and derived from the `interval`

attribute of the interest) and a duration field (derived from the `timestamp` and `expiresAt` attributes of the interest). The duration field indicates the approximate lifetime of the gradient and the interest. Gradients are used for data propagation as described in Section 2.3. (For event-triggered applications, each gradient contains a *gradient type* instead of a data rate. There are two gradient types: an exploratory gradient and a data gradient. *Exploratory gradients* are intended for path setup and repair whereas *data gradients* are for sending real data. The default gradient type is exploratory.)

When a node receives an interest, it checks if the interest exists in the cache. If no matching interest exists (*i.e.,* the interest is distinct), the node creates an interest entry and determines each field of the interest entry from the received interest. This entry contains a single gradient toward the neighbor from which the interest was received, with the specified event data rate. In our example, a neighbor of the sink will create an interest entry with a gradient of 1 event per second toward the sink. Thus, it is necessary to distinguish individual neighbors. Any locally unique neighbor identifier (*e.g.,* an IEEE 802.11 MAC address [Com97], a Bluetooth cluster address [Haa00], a random, ephemeral transaction identifier [EE01a]) may be applicable. If there is the matching interest entry, but no gradient for the sender of the interest, the node adds a gradient toward that neighbor and updates the timestamp and duration fields appropriately. Finally, if there are both an entry and a gradient, the node simply updates the timestamp and duration fields.

The expired gradient will be removed from its interest entry, but not all gradients will expire at the same time. For example, if two sinks send indistinct interests with different expiration times, some node in the network may have an interest entry

with different gradient expiration times. When all gradients in an interest entry have expired, the interest entry is removed from a cache.

After receiving an interest, a node may decide to re-send the interest to some subset of its neighbors. To its neighbors, this interest *appears to originate from the sending node*, even though a distant sink might be the actual originator. With such completely *local interaction*, interests are diffused throughout the network. However, not all received interests are re-sent. By using the interest cache, a node may suppress a received interest if it recently re-sent a matching interest.

Generally, there are several possible choices for neighbors to re-send the interest. The simplest alternative is to rebroadcast the interest to all neighbors, which is equivalent to flooding the interest. This alternative is reasonable in the absence of information about the sensor nodes that can satisfy the interest. Directed diffusion with this alternative is evaluated in Chapter 3. In our example sensor network, it may also be possible to geographically route the interest or to limit the scope of interest diffusion, using some of the techniques described in the literature [KV98, KK00, CHMK00, YGE01], for energy savings. Finally, in an immobile sensor network, a node might use cached data to direct interests (Section 2.3). For example, if a node previously heard from a neighbor **C** some data that contain a location estimate within the specified rectangle of the received interest, it can direct the interest to **C**, rather than broadcast the interest. The previously heard data do not need to match the other current interest attributes for achieving this heuristic.

Given that interests are flooded, all nodes establish gradients (Figure 2.2). Unlike the simplified description (Figure 2.1(b)), every pair of neighboring nodes establishes a gradient toward each other, as a crucial consequence of local interactions. An

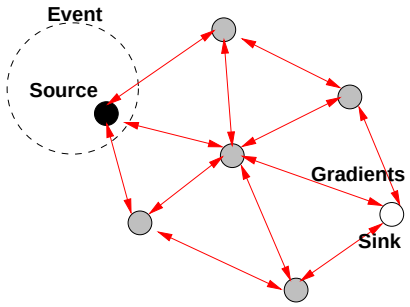


Figure 2.2: Gradient establishment.

interest does not contain information about a sink. Therefore, when a node receives an interest, it is impossible for the node to determine whether the interest is delivered back to the node because there is a loop, the interest is delivered using another path, or the identical interest is newly generated from another sink. Such bi-directional gradients can cause a node to receive one copy of low data rate events from each of its neighbors. However, this technique can enable fast recovery from failed paths or reinforcement of empirically better paths (Section 2.4), and does not incur persistent loops (Section 2.3).

Generally, a gradient is composed of a *value* and a direction in which to send events. In our sensor network, the gradient value is the data rate. The directed diffusion paradigm provides the designer the freedom to attach different semantics to gradient values (*e.g.,* event reporting rate gradients in our sensor network, probabilistically forwarding gradients in the other sensor networks for load balancing, exploratory and data gradients for event-triggered applications).

The interest propagation rules and gradient semantics described in this section are for a particular task type. Generally, a sensor network may support many different task types. Interest propagation rules and gradient semantics may be different for different task types. For example, a task type of the form "Count the number of distinct four-legged animals in rectangle R detected over the next T seconds" may not be appropriate for event data rate gradients. However, some elements of interest propagation are similar (*e.g.,* the interest entry, the interest re-distribution rule). We intend to cull these similarities into a *diffusion substrate* at each node so that sensor network designers can use a library of interest propagation techniques (or rules for data and reinforcement processing discussed in the subsequent sections) for different task types.

2.3 Data Propagation

After a node in the specified region receives an interest, the node tasks its local sensors to collect samples (to save power, sensors are off until tasked). In this thesis, we do not discuss the target recognition algorithms in detail. Briefly, these algorithms simply match sampled waveforms against a library of pre-sampled, stored waveforms (based on the observation that a four-legged animal's acoustic or seismic footprint differs from, for example, a human being's). Given that the sampled waveform may match the stored waveform to varying extents, the algorithms usually associate a degree of confidence with the match. Additionally, the intensity of the sampled waveform may roughly indicate distance of the signal origin, though possibly not direction.

A sensor node that detects a target searches its interest cache for a matching interest entry. In this example, an entry will match a target if its `rect` encompasses the target location estimate and its `type` matches the target type. Upon finding the entry, the sensor node computes the highest requested event rate among all its outgoing gradients. The node then tasks its sensor subsystem to generate event samples at this highest event rate. In this example, the initial data rate is 1 event per second until reinforcement is applied as described in Section 2.4. As a source, the node unicasts to each neighbor that corresponds to each of its outgoing gradients, an event every second of the form:

```
type = four-legged animal // type of animal seen
instance = elephant       // instance of this type
location = [125, 220]     // location estimate
intensity = 0.6           // signal amplitude measure
confidence = 0.85         // confidence in the estimate
timestamp = 01:20:40      // local time when event was generated
```

For event-triggered applications, there is no explicit data rate. A source generates data once certain conditions are satisfied. Data can be classified into two types: exploratory and non-exploratory. An event is marked as exploratory if the source has no data gradient or the source has generated no exploratory event within a window of time (or events). Exploratory events are sent along all outgoing gradients whereas non-exploratory events are sent along only data gradients.

A node that receives an event from its neighbors also searches its interest cache for a matching interest entry. If no match exists, the event is dropped. If a match exists, the node searches the *data cache* (associated with the matching interest entry)

for a matching data entry to detect and to prevent a data loop. If a received event matches a data entry, the event is dropped. Otherwise, the received event is added to the data cache and resent to the node's neighbors.

A node can also examine its data cache to determine the data rate of received events (In our simulations in Section 3, as a simplification, we include the data rate in the event descriptions). To re-send a received event, a node needs to examine the matching interest entry's gradient list. If gradients are at a data rate that is greater than or equal to the rate of received events, the node may simply send the received event to the corresponding neighbors. However, if gradients are at a lower data rate than others (due to selective reinforcement as described in Section 2.4), the node may *downconvert* to the appropriate gradient. For instance, a node may have been receiving data at 100 events per second, but one of its gradients (*e.g.,* established by a second sink originating an indistinct task with a larger interval) is at 50 events per second. In this example, the node may only send every alternate event toward the corresponding neighbor. Alternatively, it might interpolate two successive events in an application-specific manner (*e.g.,* it might choose the sample with the higher confidence estimate).

Loop prevention and downconversion demonstrate the benefit of embedding application semantics in all nodes. Even though this design is not pertinent to traditional networks, it is feasible with application-specific sensor networks and capable of improving network performance significantly (Chapter 3).

2.4 Reinforcement for Path Establishment and Pruning

In our example, the sink initially and repeatedly diffuses an interest for low-rate events (1 event per second). We call these *exploratory* events, since they are intended for path setup and repair. We call the gradients set up for exploratory events *exploratory* gradients. When sources detect a matching target, they send exploratory events (possibly along multiple paths) toward the sink. After the sink receives these exploratory events, it reinforces at least one particular neighbor to "draw down" real *data* (*i.e.,* events at a higher data rate that allow high quality tracking of targets). We call the gradients set up for receiving high quality tracking events *data* gradients.

2.4.1 Path Establishment Using Positive Reinforcement

One example of *data driven* local rules for reinforcement is to reinforce any neighbor from which a node receives a distinct exploratory event. To reinforce this neighbor, the sink re-sends the original interest but with a smaller `interval` (*i.e.,* higher data rate):

```
type = four-legged animal
interval = 10ms
rect = [-100, 200, 200, 400]
timestamp = 01:22:35
expireAt = 01:30:40
```

For event-triggered applications, there is no `interval` attribute for distinguishing between an interest and a reinforcement. Therefore, we implement a positive reinforcement message instead of using the original interest message. This reinforcement message is similar to the original interest message except the message type.

When the neighboring node receives this reinforcement, it notices that it already has a gradient toward this reinforcing node but at lower event rate than the rate specified in this interest. If this new event rate is also higher than that of any existing gradient (*i.e.,* if the "outflow" increases), the node must also reinforce at least one neighbor. Again, the node uses its data cache and local reinforcement rule. For example, this node might select the neighbor from which it first received the latest exploratory event matching the interest. Alternatively, it might select all neighbors from which distinct exploratory events were recently received (We do not need to reinforce neighbors that are already sending traffic at the higher data rate. This alternative is evaluated in Chapter 3). Through this sequence of local interactions, at least one data path is established from source to sink.

Intuitively, the described local rule selects empirically low-delay paths (Figure 2.3), and periodically reacts to changes in path quality. Whenever one path delivers an exploratory event faster then others, the sink attempts to use this path to draw down data. Therefore, the rate of the exploratory event is a tradeoff parameter between reactivity and energy efficiency. However, because receiving one new exploratory event is sufficient to trigger reinforcement, this could be wasteful of resources. More sophisticated local rules are possible. For example, a node reinforces the neighbor that has consistently sent events before others or has sent the most events. These local rules trade off reactivity for increased stability. However,

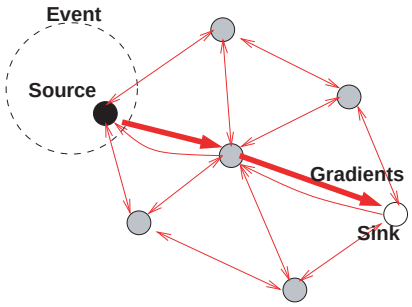


Figure 2.3: Gradients after the sink reinforces the empirically lowest delay path.

this tradeoff is a subject of future research because significant experimentation is required for complete understanding.

2.4.2 Path Establishment for Multiple Sources and Sinks

Even though we have described reinforcement using a single-source scenario, the described local rules must and do work with multiple sources (Figure 2.4). Initially, all gradients are exploratory. In this topology, data from both sources reaches the sink via neighbor **C** and **D**. If one of the neighbors (*e.g.,* **C**) consistently delivers faster, only the path through that neighbor will be reinforced. As a result, events from both sources can be aggregated to save energy. However, if the sink receives **B**'s events earlier via **D**, but **A**'s events earlier via **C**, the sink will attempt to draw down data from both neighbors (not shown). Therefore, the energy savings gained by data aggregation will not be possible in this circumstance. Reinforcement rules that encourage more data aggregation are described in Chapter 4. (Note that in directed diffusion, the sink would not be able to associate a source with an event.

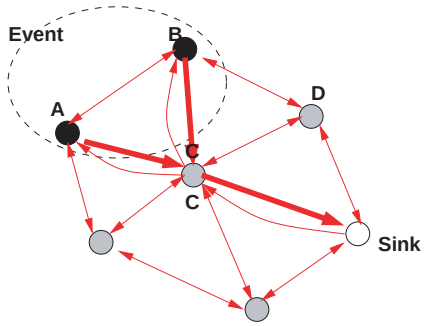


Figure 2.4: Gradients after reinforcement, multiple sources.

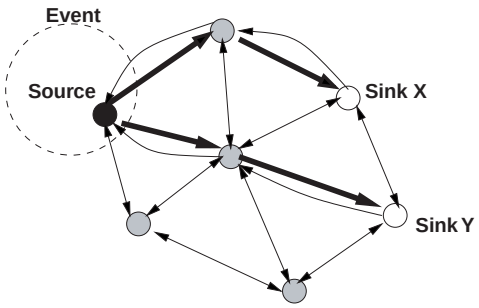


Figure 2.5: Gradients after reinforcement, multiple sinks.

Thus, the phrase "**A**'s events" is somewhat misleading. What we really mean is that data generated by **A** that is distinguishable in content from data generated by **B**.)

Similarly, if two sinks express identical interest, our interest propagation, gradient establishment, and reinforcement rules still work correctly. Without loss of generality, sink **Y** has already reinforced a high quality path to the source (Figure 2.5). Other nodes continue to receive exploratory events. When a user tasks the network at sink **X** with an identical interest, **X** can examine its data cache and immediately reinforce without waiting for data.

2.4.3 Local Repair of Failed Paths

Although only situations where sinks trigger reinforcement have been described so far, intermediate nodes on a previously reinforced path can also apply the reinforcement rules. Such reinforcement by intermediate nodes is very useful to enable local recovery of failed or degraded paths caused by several factors (*e.g.,* node energy depletion, obstacle). For example, the link quality (between the source and node **C**) degrades and events are frequently corrupted. When **C** detects this degradation (either by noticing that the event reporting rate from its upstream neighbor is now lower or by realizing that other neighbors have been transmitting previously unseen location estimates), **C** can apply the reinforcement rules to discover a new path (Figure 2.6). However, it is possible that all nodes downstream of the lossy link may initiate reinforcement procedures. As a result, several data paths are established, and resources are wasted (although the number of these paths will be eventually reduced). A possible mechanism to avoid implosion of reinforcements is that **C** may interpolate location estimates from the received events so that downstream nodes still perceive high quality tracking. Other approaches are also possible, but they are subjects of future work.

2.4.4 Path Pruning and Negative Reinforcement

The described algorithm for positive reinforcement can result in more than one reinforced path. For example (Figure 2.7), if the sink reinforces neighbor **A**, but then receives a new exploratory event from neighbor **B**, it will reinforce **B** (This path may or may not be completely disjoint from the path through neighbor **A**). If

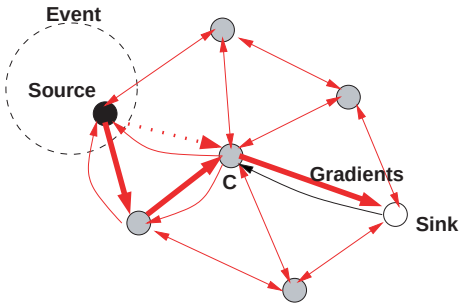


Figure 2.6: Gradients after local repair caused by connection quality degradation.

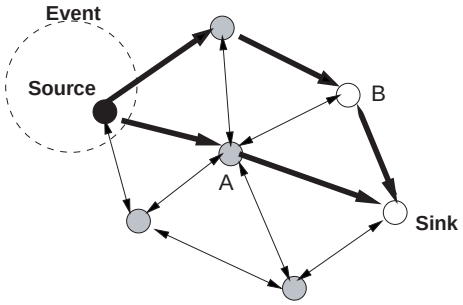


Figure 2.7: Gradients after several rounds of reinforcement, multiple paths.

the path through **B** is consistently better (*i.e.,* **B** sends events before **A** does), the sink needs a mechanism to degrade or to *negatively reinforce* the path through **A**.

One mechanism for negative reinforcement is soft state, *i.e.,* to time out all data gradients unless they are explicitly reinforced. With this approach, the sink would periodically reinforce **B** and cease reinforcing **A**. The path through **A** would eventually be degraded to being exploratory gradients. Another approach (evaluated in Chapter 3) is to explicitly degrade the path through **A** by sending a negative reinforcement message to **A**. In this rate-based diffusion, the negative reinforcement is the interest with the lower data rate. (For event-triggered applications, we implement a negative reinforcement message instead of using the original interest

message with the lower data rate. This negative reinforcement message is similar to the original interest message except the message type.)

When **A** receives this negative reinforcement, it degrades its gradient toward the sink. Moreover, if now all its gradients are exploratory, **A** negatively reinforces its neighbors that have been sending data to it (as opposed to exploratory events). This sequence of local interactions ensures the path through **A** is degraded rapidly, but at the cost of increased overhead.

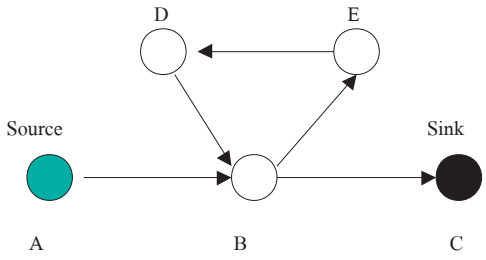
One plausible local rule for explicit negative reinforcement is to negatively reinforce the neighbors that have sent events but none of them is new (*i.e.,* other neighbors have consistently sent events before this neighbor) within a window of N events or time T. The local rule, evaluated in Chapter 3, is based on a time window of T, selected to be 2 seconds in our simulations. However, such a conservative rule may be energy inefficient. For example, even if only one event in ten, sent by neighbor **A**, was new, the sink will not negatively reinforce **A**. Other alternatives include negatively reinforcing the neighbors that have sent relatively few non-duplicate events, compared to the neighbor that have sent the most non-duplicate events. However, significant experimentation is required before deciding which local rule is more energy efficient.

2.4.5 Loop Truncation Using Negative Reinforcement

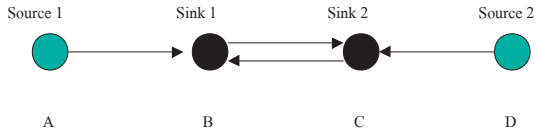
In addition to suppressing high-delay or lossy paths, our local rule for negative reinforcement can also be used for loop truncation because the looping paths never deliver events first (Figure 2.8(a)). Given that only neighbors that sent the exploratory event first are reinforced, one may expect that the looping paths would

never be reinforced (particularly for single-source-single-sink scenarios). However, the reinforced paths in a given round of exploratory events may differ from those in the previous rounds. Although no looping path is reinforced, a union of reinforced paths from multiple rounds may contain loops. Although the looping message will be immediately suppressed using a message cache, in general, we would still benefit from truncating the looping paths for resource savings. However, such loop truncation is not always appropriate, specifically for some shared data gradient maps with multiple sources and sinks. For example (Figure 2.8(b)), if both sources send distinguishable events, the gradient **B-C** and **C-B** should not be pruned because each of them is necessary for delivering events for a particular source-sink pair. Although such gradients may deliver some looping events, they also consistently deliver new events. With our conservative rule for negative reinforcement, those gradients will not be negatively reinforced.

Furthermore, even without loops, it is still reasonable to keep our negative reinforcement rule conservative so that useful paths will not be pruned. For example (Figure 2.4), both sources may consistently send distinguishable events but they may also send identical events once in a while. Although originating from different sources, the identical events are considered duplicates for diffusion. According to data centricity of diffusion, events are independent from their sources (*i.e.,* A node would not be able to associate a source with an event). The current instantiation of diffusion maintains only the data paths along which useful (new) data are consistently sent, regardless of the sources. Thus, generally, we do not guarantee that there will be at least one data path from every source to every sink. To guarantee such paths, a source id or a random unique id can be used as a data attribute. However,



(a) A truncatable loop



(b) An untruncatable loop

Figure 2.8: Negative reinforcement for loop truncation.

such id is not necessary for diffusion in general. The path from one of the sources will be pruned if the negative reinforcement rule is too aggressive against duplicates. Conversely, given our conservative rule, no source will be negatively reinforced.

2.5 Discussion

The directed diffusion paradigm does not limit the designer to only a particular usage whereby interest propagation causes gradient establishment for drawing data. Other usages are also possible, including one when nodes may propagate data in the absence of interests and implicitly trigger gradient establishment. Such usage enables sensor nodes to spontaneously propagate an important event or to warn

Diffusion element	Design Choices
Interest Propagation	• Flooding • Constrained or directional flooding based on location • Directional propagation based on previously cached data
Data Propagation	• Reinforcement to single path delivery • Multipath delivery with selective quality along different paths • Multipath delivery with probabilistic forwarding
Data caching and aggregation	• For robust data delivery in the face of node failure • For coordinated sensing and data reduction • For directing interests
Reinforcement	• Rules for deciding when to reinforce • Rules for how many neighbors to reinforce • Negative reinforcement mechanisms and rules

Figure 2.9: Partial Design Space for Diffusion

other sensor nodes of impending activity. Moreover, other design choices for each element of diffusion are also possible (see Figure 2.9).

Our description points out several key features of diffusion, and how it differs from traditional networking. First, directed diffusion is application-specific and data-centric; data is named using attribute-value pairs and de-coupled from data producers. All communication in diffusion-based networks is neighbor-to-neighbor, rather than end-to-end communication in traditional networks. In other words, every node is an "end" in a sensor network. By this, we mean there are no routers in sensor networks. Specifically, each sensor node can interpret data and interest messages. This design choice is justified by the task-specificity of sensor networks which are not general-purpose communication networks. Second, sensor nodes do not need globally unique identifiers or addresses. However, nodes do need to distinguish between neighbors. Finally, in an IP-based sensor network, for example, data gathering or

processing might be performed by a collection of specialized servers, which may, in general, be far away from the sensed phenomena. In our sensor network, given that every node can cache, aggregate, and more generally, process messages, it is possible to perform coordinated sensing close to the sensed phenomena.

However, directed diffusion is definitely related to traditional networking techniques, especially ad hoc on-demand routing protocols (see also Section 5.3). In a sense, directed diffusion is an application-level on-demand routing technique because routes or gradients are established only when needed. However, directed diffusion is different from ad hoc on-demand routing protocols in several ways.

- Directed diffusion does not attempt to find one loop-free path between source and sink before data transmission commences. Instead, constrained or directional flooding is used to establish multiple paths, and data messages are *initially* sent redundantly along these paths.

- Soon thereafter, the number of paths is reduced by reinforcement mechanisms, based on empirically observed path performance.

- A message cache is used to perform loop avoidance.

This peculiar design alternative is based on our intention to explore path establishment algorithms using strictly local (neighbor-to-neighbor) communication. The intuition behind this alternative is the ant colony [CD97, DMC96, SDC97, SHBR96] that is extremely scalable and robust using only local communication for path establishment (see also Section 5.2). However, using strictly local communication implies that, as far as a node knows, the received message from a neighbor has originated

from that neighbor (The location information in a data message might reveal otherwise, but that information still doesn't contain topology metrics). This design can be energy efficient in highly dynamic networks because topology changes need not be propagated across the network. Although the resulting communication paths may be sub-optimal, the energy inefficiency due to path sub-optimality can be countered by carefully designed in-network aggregation techniques. Thus, the design trades some energy efficiency for increased robustness and scalability.

Our particular instantiation of location tracking application captures many (if not all) essential features of remote surveillance sensor networks. Although several details of such tracking networks have been discussed, much experimentation and evaluation of various techniques is still necessary for complete understanding of the robustness, scalability, and performance implications of directed diffusion in general. The next chapter takes an initial step in this direction.

Chapter 3

Evaluating Directed Diffusion

To demonstrate the benefits of our paradigm over traditional schemes, we evaluate directed diffusion analytically in Section 3.1 and quantitatively via simulations in Section 3.2.

3.1 Analytic Evaluation

In this section, we present an analytic evaluation of the data delivery cost for directed diffusion and two idealized schemes: *omniscient multicast* and *flooding*. This analysis serves to sanity check the intuition behind directed diffusion, and highlights some of the differences between diffusion and the other approaches.

For analytic tractability, we analyze these three schemes in a very simple, idealized setting. We assume a square grid consisting of N nodes. In this grid, node transmission ranges are such that each node can communicate with exactly eight neighboring nodes on the grid. Figure 3.1 shows links between pairs of nodes that can communicate with each other. All n sources are placed along nodes on the left edge of the grid, whereas all m sinks are placed along the right edge. The first source

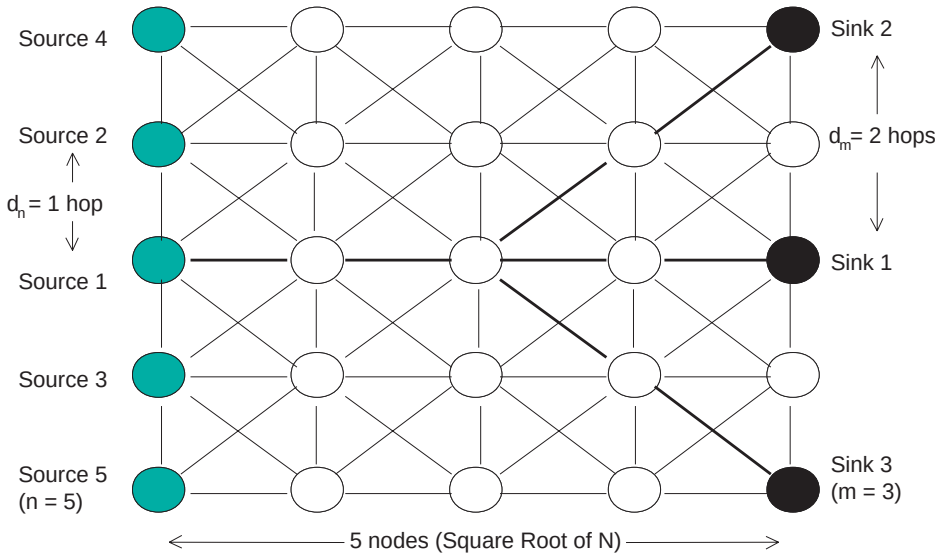


Figure 3.1: An example of our square grid topology

is at the center of the left border. The i^{th} source is $d_n\lfloor\frac{i}{2}\rfloor$ hops above (if i is even) or below (if i is odd) the first source. This placement scheme is also used for sinks except that the distance between 2 adjacent sinks is d_m hops rather than d_n hops. Given that sources and sinks are vertically placed only, $\sqrt{N}$ can not be less than $max(nd_n, md_m)$.

3.1.1 Flooding

In the *flooding* scheme, sources flood all events to every node in the network. Flooding is a contrary case for directed diffusion; if the latter does not perform better than flooding does, it cannot be considered viable for sensor networks.

In this analytic evaluation, our measure of performance is the total cost of transmission and reception of one event from each source to all the sinks. We define cost

as one unit for message transmission, and one unit for message reception. These assumptions are clearly idealized in two ways. Transmission and reception costs may not be identical, and there might be other metrics of interest. However, both costs are generally in the same order of magnitude. These assumptions will not asymptotically effect our results. We consider more realistic measures with simulation in Section 3.2.

By this measure, the cost of flooding, denoted by $C_f(N, n, m, d_n, d_m)$, or simply C_f is given by:

$$C_f = C_{tx,f} + C_{rx,f} \tag{3.1}$$

where

$$C_{tx,f} = nN \tag{3.2}$$

$$C_{rx,f} = 2n((\sqrt{N}-1)\sqrt{N} + (\sqrt{N}-1)\sqrt{N} + 2(\sqrt{N}-1)^2) \tag{3.3}$$

Thus,

$$\begin{aligned} C_f &= nN + 2n((\sqrt{N}-1)\sqrt{N} + (\sqrt{N}-1)\sqrt{N} + 2(\sqrt{N}-1)^2) \\ &= nN + 4n(\sqrt{N}-1)(2\sqrt{N}-1) \end{aligned} \tag{3.4}$$

The transmission cost $C_{tx,f}$ for flooding n events (one event from each source) is nN because each node sends only one MAC broadcast per event. Conversely, each node can receive the same event from all neighbors. Thus, the reception cost $C_{rx,f}$ for those n events is determined by $2n$ times the number of links in the network.

The data delivery cost for flooding is dominated by $O(nN)$ which is asymptotically higher than the cost of other schemes (see Section 3.1.2 and 3.1.3).

3.1.2 Omniscient Multicast

In the *omniscient multicast* scheme, each source transmits its events along a shortest-path multicast tree to all sinks. In our analysis, as well as in the simulations described in Section 3.2, we *do not* account for the cost of tree construction. Omniscient multicast instead indicates the best possible performance achievable in an IP-based sensor network without considering overhead. We use this scheme to give the reader some intuition for how the choice of in-network processing mechanism effects performance.

For omniscient multicast, the data delivery cost is determined by twice the number of links on its source-specific shortest-path trees. However, even in this simple grid topology, there are several shortest paths for each source-sink pair. We choose the shortest path using the following simple, deterministic rule. From a sink to a source, a diagonal link is always the next hop as long as it leads to a shortest path. Otherwise, a horizontal link is selected. This path-selection rule is repeated until the source is reached. Thus, no shortest path includes vertical links. For example, if we denote a shortest path tree rooted at source j by T_j, then the number of links on T_1 has two components: the number of horizontal links $L_H(T_1)$ and diagonal links $L_D(T_1)$. Specifically,

$$L_H(T_1) = \sqrt{N} - 1 \tag{3.5}$$

$$L_D(T_1) = d_m \lfloor \frac{m}{2} \rfloor (\lfloor \frac{m}{2} \rfloor + 1) - d_m \lfloor \frac{m}{2} \rfloor ((m-1) \mod 2) \tag{3.6}$$

Other choices could result in a different cost, since the number of shared links on the tree could be different.

The cost of omniscient multicast C_o is the sum of the costs of n trees, one rooted at each source. If we denote by $C(T_j)$ the cost to transmit an event from source j, it turns out that we can express this cost in terms of T_1 as:

$$C(T_j) = C(T_1) + C(T_j - T_1) - C(T_1 - T_j) \tag{3.7}$$

$C(T_j - T_k)$ is interpreted as the cost of transmission and reception along the tree formed by removing, from T_j, those links that are common to T_j and T_k. Furthermore, for ease of exposition, $C(T_j)$ can be expressed as the sum of two costs: the cost of transmission and reception along the horizontal links $H(T_j)$, and the analogous cost along the diagonal links $D(T_j)$. Specifically,

$$H(T_j) = 2L_H(T_j) \tag{3.8}$$

$$D(T_j) = 2L_D(T_j) \tag{3.9}$$

We can then write C_o as follows:

$$C_o = \sum_{j=1}^{n} \Big\{ D(T_1) + H(T_j) + D(T_j - T_1) - D(T_1 - T_j) \Big\} \tag{3.10}$$

where

$$H(T_j) = 2\Bigg\{\sqrt{N} - 1 - \Bigg(\lfloor\frac{j}{2}\rfloor d_n \\ - min(\lfloor\lfloor\frac{j}{2}\rfloor\frac{d_n}{d_m}\rfloor, \lfloor\frac{m-(j \mod 2)}{2}\rfloor)d_m\Bigg)\Bigg\} \tag{3.11}$$

$$D(T_j - T_1) = 2\Bigg\{\lceil\frac{m+(j \mod 2)}{2}\rceil\lfloor\frac{j}{2}\rfloor d_n \\ + \sum_{l=1}^{min(\lfloor\lfloor\frac{j}{2}\rfloor\frac{d_n}{d_m}\rfloor, \lfloor\frac{m-(j \mod 2)}{2}\rfloor)} (d_n\lfloor\frac{j}{2}\rfloor - ld_m)\Bigg\} \tag{3.12}$$

$$D(T_1 - T_j) = 2\Bigg\{\sum_{l=1}^{\lfloor\frac{m-(j \mod 2)}{2}\rfloor} min(d_n\lfloor\frac{j}{2}\rfloor, ld_m)\Bigg\} \tag{3.13}$$

Asymptotically, the data delivery cost of omniscient multicast C_o is dominated by $O(n\sqrt{N})$ for $m \ll \sqrt{N}$.

3.1.3 Directed Diffusion

The analysis of diffusion proceeds along the same lines as that of omniscient multicast. To simplify the analysis, we assume that the tree that diffusion's localized algorithms construct is the "union" of the shortest path tree rooted at each source. This assumption is approximately valid when the network operates at low load levels. Furthermore, of the many available shortest paths, diffusion chooses one according to the following rule: from a sink to a source, a diagonal link is always the next hop

as long as it is along the shortest path to a source; otherwise, a horizontal link is selected. This rule is the same one we used for omniscient multicast.

Despite using the same path-selection scheme, the cost of diffusion C_d differs from that of omniscient multicast, primarily because of application-level data processing. Specifically, if all sources send identical target location estimates, then, given that diffusion can perform application-level duplicate suppression, the data delivery cost of diffusion is twice the number of links in the union of all shortest path trees rooted at the source. Hence,

$$\begin{aligned} C_d = C(UT_{1\to n}) = \; & C(T_1) + \sum_{j=2}^{n} \Big\{ H(T_j - UT_{1\to(j-1)}) \\ & + D(T_j - UT_{1\to(j-1)}) \Big\} \end{aligned} \tag{3.14}$$

where

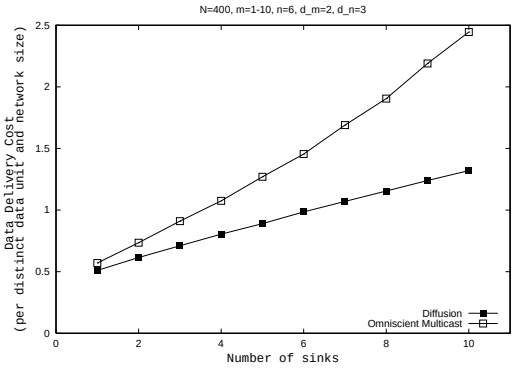
$$H(T_j - UT_{1\to(j-1)}) = H(T_j) \tag{3.15}$$

$$\begin{aligned} D(T_j - UT_{1\to(j-1)}) = 2 \Bigg\{ & \lceil \frac{m + (j \mod 2)}{2} \rceil d_n \\ & + \sum_{l=1}^{min(\lfloor \lfloor \frac{j}{2} \rfloor \frac{d_n}{d_m} \rfloor, \lfloor \frac{m-(j \mod 2)}{2} \rfloor)} min(d_n, d_n \lfloor \frac{j}{2} \rfloor - l d_m) \Bigg\} \end{aligned} \tag{3.16}$$

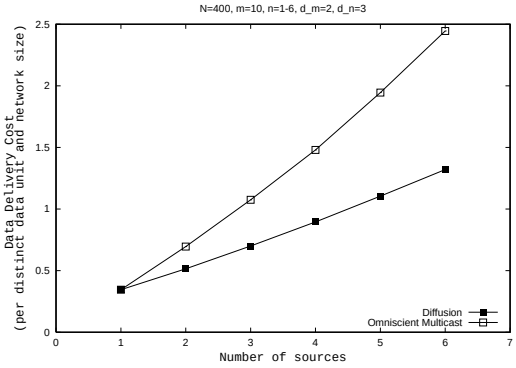
Similar to C_o, the dominant term of C_d is $O(n\sqrt{N})$ for $m \ll \sqrt{N}$.

3.1.4 Comparison

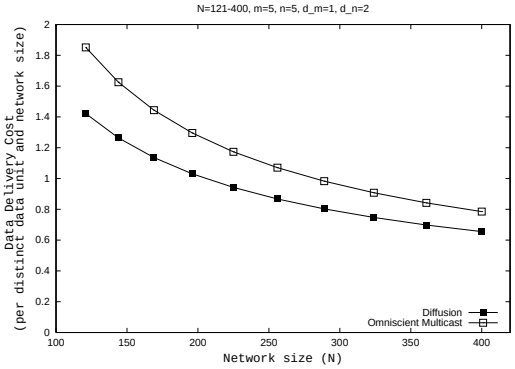
The data delivery cost of flooding C_f is several orders of magnitude higher than that of omniscient multicast C_o. However, C_o is still higher than the diffusion cost C_d



(a) Impact of the number of sinks



(b) Impact of the number of sources



(c) Impact of network size

Figure 3.2: Impact of various parameters.

because $D(T_1) - D(T_1 - T_j) \geq 0$ and $D(T_j - T_1) \geq D(T_j - UT_{1\rightarrow(j-1)})$. To validate this reasoning, the data delivery cost (normalized by network size) for directed diffusion and omniscient multicast is plotted using various parameters (*i.e.,* N, m, and n). As the number of sources and sinks increases (Figure 3.2(a) and 3.2(b)), the cost saving due to in-network processing (*e.g.,* duplicate suppression) of diffusion becomes more evident (given that C_d increases at a lower rate than C_o). This result has been recently confirmed by researchers [KEW02].

Of particular interest is the plot of cost versus network size (Figure 3.2(c)). Since diffusion can suppress application-level duplicates, one would expect that C_o is merely nC_d. The main reason this does not hold is that our analysis somewhat conservatively estimates diffusion costs. In practice, diffusion would have negatively reinforced several of the links that our analysis includes.

3.2 Simulation

In this section, we report on some results from a simulation of our location tracking sensor network. We use packet-level simulation to explore, in some detail, the implications of some of our design choices. This section describes our methodology, compares the performance of diffusion against some idealized schemes, then explores impact of network dynamics on simulation.

3.2.1 Goals, Metrics, and Methodology

We implemented our animal tracking instance of diffusion in the *ns-2* [BEF+00] simulator (the current *ns-2* release with diffusion supports is downloadable from

`http://www.isi.edu/nsnam/ns`). Our goals in conducting this evaluation study were four-fold: First, verify and complement our analytic evaluation. Second, understand the impact of dynamics—such as node failures—on diffusion. Third, explore the influence of the radio MAC layer on diffusion performance. Finally, study the sensitivity of directed diffusion performance to the choice of parameters.

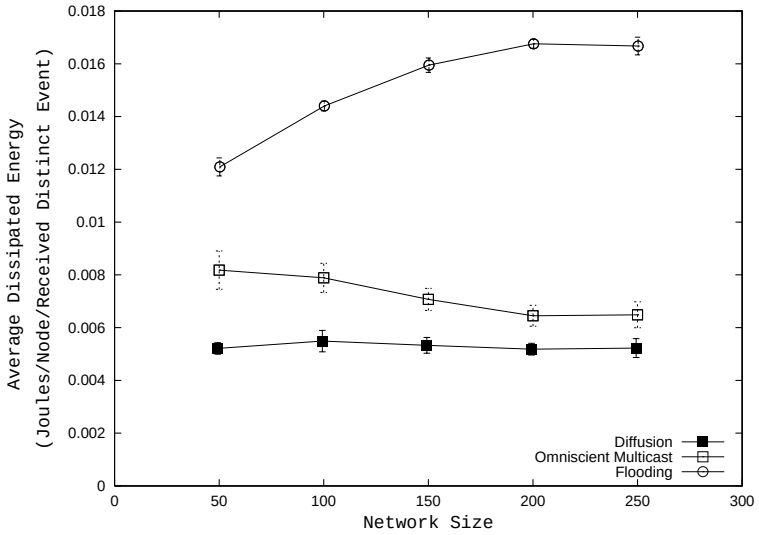
We choose three metrics to analyze the performance of directed diffusion and to compare it to other schemes: average dissipated energy, average delay, and distinct-event delivery ratio. **Average dissipated energy** measures the ratio of total dissipated energy *per node* in the network to the number of *distinct* events seen by sinks. This metric computes the average work done by a node in delivering useful tracking information to the sinks. The metric also indicates the overall lifetime of sensor nodes. **Average delay** measures the average one-way latency observed between transmitting an event and receiving it at each sink. This metric defines the temporal accuracy of the location estimates delivered by the sensor network. **Distinct-event delivery ratio** is the ratio of the number of distinct events received to the number originally sent. A similar metric was used in earlier work to compare ad-hoc routing schemes [BMJ+98]. We study these metrics as a function of sensor network size.

In all our experiments, we operate the sensor network in a regime far from overload. Thus, our sensor nodes do not experience congestion. We do this to simplify our understanding of the results. There exist plausible approaches (such as in-network data rate downconversion or aggressive data quality reduction through aggregation) for dealing with congestion in diffusion-based sensor networks. However, such approaches are beyond the scope of this dissertation.

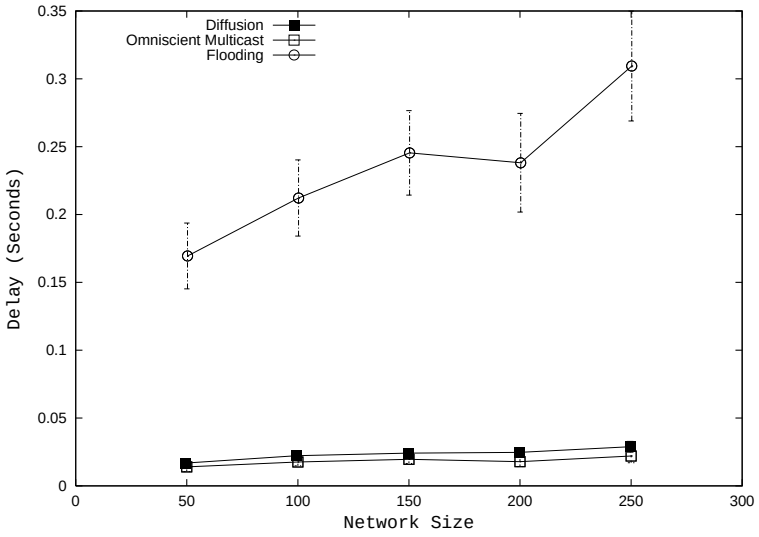
To completely specify our experimental methodology, we need to describe the sensor network generation procedure, our choice of radio parameters, and our workload. The following paragraphs do this.

In order to study the performance of diffusion as a function of network size, we generate a variety of sensor fields of different sizes. In each of our experiments, we study five different sensor fields, ranging from 50 to 250 nodes in increments of 50 nodes. Our 50 node sensor field generated by randomly placing the nodes in a 160m by 160m square. Each node has a radio range of 40m. Other sizes are generated by scaling the square and keeping the radio range constant in order to approximately *keep the average density of sensor nodes constant*. We do this because the macroscopic connectivity of a sensor field is a function of the average density. If we had kept the sensor field area constant but increased network size, we might have observed performance effects not only due to the larger number of nodes but also due to increased connectivity. Our methodology factors out the latter, allowing us to study the impact of network size alone on some of our mechanisms.

The *ns-2* simulator implements a 1.6 Mbps 802.11 MAC layer. Our simulations use a modified 802.11 MAC layer. To more closely mimic realistic sensor network radios [Kai99], we altered the *ns-2* radio energy model such that the idle time power dissipation was about 35mW, or nearly 10% of its receive power dissipation (395mW), and about 5% of its transmit power dissipation (660mW). This is not a completely satisfactory choice of MAC layer, since there are compelling energy efficiency reasons for selecting a TDMA-style MAC for sensor networks rather than one based on channel acquisition using RTS/CTS [PK00]. Briefly, these reasons have to do with energy consumed by the radio during idle intervals; with a TDMA-style



(a) Average dissipated energy



(b) Average delay

Figure 3.3: Directed diffusion compared to flooding and omniscient multicast.

MAC, it is possible to put the radio in standby mode during such intervals. By contrast, an 802.11 radio consumes as much power when it is idle as when it receives transmissions. In Section 3.2.5, we analyze the impact of a MAC energy model in which listening for transmissions dissipates as much energy as receiving them.

Finally, in most of our simulations, we use a fixed workload which consists of five sources and five sinks. All sources are randomly selected from nodes in a 70m by 70m square within the sensor field. Sinks are uniformly scattered across the sensor field. Each source generates two events per second. The rate for exploratory events was chosen to be one event in 50 seconds. Events were modeled as 64 byte packets, interests as 36 byte packets. Interests were periodically generated every 5 seconds, and the interest duration was 15 seconds. We chose the window for negative reinforcement to be 2 seconds. These parameter choices were informed both by the particular sensor network under consideration (small event descriptions, sources within a geographic region) and by our stated desire to explore a regime of the sensor network in a non-congested regime (see above). Data points in each graph represent the mean of ten scenarios with 95% confidence intervals.

3.2.2 Comparative Evaluation

Our first experiment compares diffusion to omniscient multicast and flooding scheme for data dissemination in networks. Figure 3.3(a) shows the average dissipated energy per packet as a function of network size. Omniscient multicast dissipates a little less than a half as much energy per packet per node than flooding. It achieves such energy efficiency by delivering events along a single path from each source to every sink. Directed diffusion has noticeably better energy efficiency than omniscient

multicast. For some sensor fields, its dissipated energy is only 60% that of omniscient multicast. As with omniscient multicast, it also achieves significant energy savings by reducing the number of paths over which redundant data is delivered. In addition, diffusion benefits significantly from *in-network aggregation*. In our experiments, the sources deliver identical location estimates, and intermediate nodes *suppress* duplicate location estimates. This corresponds to the situation where there is, for example, a single four-legged animal within the specified sub-region.

Why then, given that there are five sources, is diffusion (with negative reinforcement) not nearly five times more energy efficient than omniscient multicast? First, both schemes expend comparable—and non-negligible—energy listening for transmissions. Second, our choice of reinforcement and negative reinforcement results in directed diffusion frequently drawing down high quality data along multiple paths, thereby expending additional energy. Specifically, our reinforcement rule that reinforces a neighbor who sends a new (*i.e.,* previously unseen) event is very aggressive. Conversely, our negative reinforcement rule, which negatively reinforces neighbors who only consistently send duplicate (*i.e.,* previously seen) events, is very conservative.

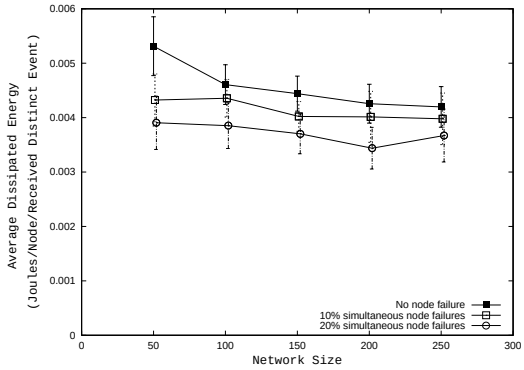
Figure 3.3(b) plots the average delay observed as a function of network size. Directed diffusion has a delay comparable to omniscient multicast. This is encouraging. To a first approximation, in an uncongested sensor network and in the absence of obstructions, the shortest path is also the lowest delay path. Thus, our reinforcement rules seem to be finding the low delay paths. However, the delay experienced by flooding is almost an order of magnitude higher than other schemes. This is

an artifact of the MAC layer: to avoid broadcast collisions, a randomly chosen delay is imposed on all MAC broadcasts. Flooding uses MAC broadcasts exclusively. Diffusion only uses such broadcasts to propagate the initial interests. On a sensor radio that employs a TDMA MAC-layer, we might expect flooding to exhibit a delay comparable to the other schemes.

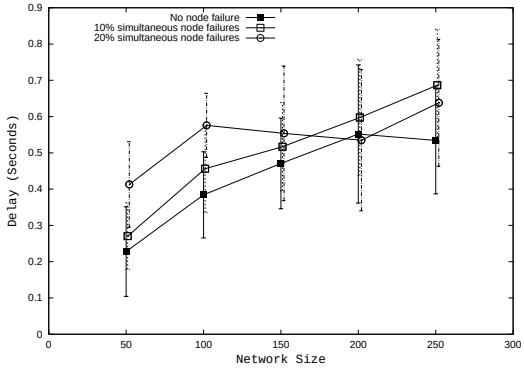
In summary, then, directed diffusion exhibits better energy dissipation than omniscient multicast and has good latency properties. Finally, all three schemes incurred an event delivery ratio of nearly one (not shown), since this experiment ignored network dynamics and was congestion-free.

3.2.3 Impact of Dynamics

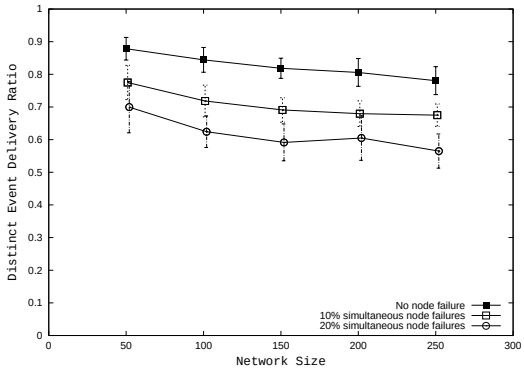
To study the impact of dynamics on directed diffusion, we simulated node failures as follows. For each sensor field, repeatedly turned off a fixed fraction (10 or 20%) of nodes for 30 seconds. These nodes were uniformly chosen from the sensor field, with the additional constraint that an equal fraction of nodes on the sources to sinks shortest path trees was also turned off for the same duration. The intent was to create node failures in the paths diffusion is most likely to use, and to create random failures elsewhere in the network. Furthermore, unlike the previous experiment, each source sends different location estimates (corresponding to the situation in which each source "sees" different animals). We did this because the impact of dynamics is less evident when diffusion suppresses identical location estimates from other sources. We could also have studied the impact of dynamics on other protocols, but, because omniscient multicast is an idealized scheme that doesn't factor in the cost of route recomputation, it is not entirely clear that such a comparison is meaningful.



(a) Average dissipated energy



(b) Average delay



(c) Event Delivery Ratio

Figure 3.4: Impact of node failures on directed diffusion.

Our dynamics experiment imposes fairly adverse conditions for a data dissemination protocol. At any instant, 10 or 20 percent of the nodes in the network are unusable. Furthermore, we do not permit any "settling time" between node failures. Even so, diffusion is able to maintain reasonable, if not stellar, event delivery (Figure 3.4(c)) while incurring less than 20% additional average delay (Figure 3.4(b)). Moreover, the average dissipated energy actually *improves*, in some cases, in the presence of node failures. This is a bit counter-intuitive, since one would expect that directed diffusion would expend energy to find alternative paths. As it turns out, however, our negative reinforcement rules are conservative enough that several reinforced paths (high-quality paths) are kept alive in normal operation. Thus, at the levels of dynamics we simulate, diffusion doesn't need to do extra work. The lower energy dissipation results from the failure of some high-quality paths.

We take these results to indicate that the mechanisms in diffusion are relatively stable at the levels of dynamics we have explored. By this we mean that diffusion does not, under dynamics, incur remarkably higher energy dissipation or event delivery delays.

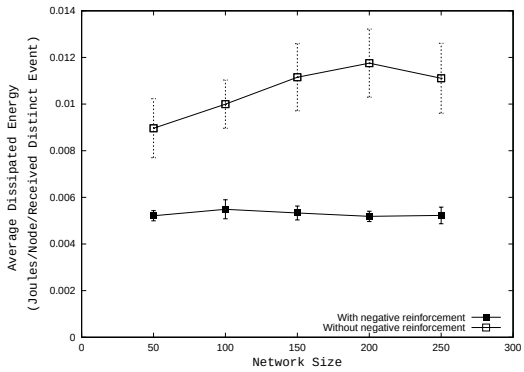
3.2.4 Impact of Aggregation and Negative Reinforcement

To explain what contributes to directed diffusion's energy efficiency, we now describe two separate experiments. In both of these experiments, we do not simulate node failures. First, we compute the energy efficiency of diffusion with and without aggregation. Recall from Section 3.2.2 that in our simulations, we implement a simple aggregation strategy, in which a node suppresses identical data sent by different sources. As Figure 3.5(b) shows, diffusion expends nearly 5 times as much energy,

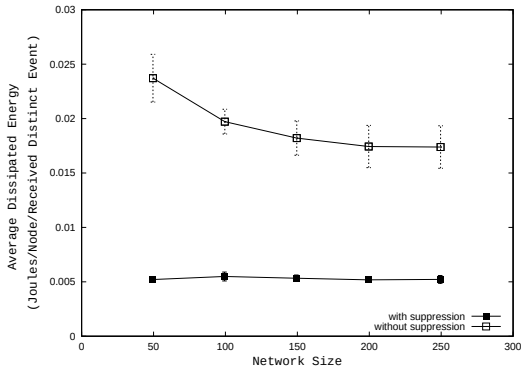
in smaller sensor fields, as when it can suppress duplicates. In larger sensor fields, the ratio is 3. Our conservative negative reinforcement rule accounts for the difference in the performance of diffusion without suppression as a function of network size. With the same number of sources and sinks, the larger network has longer alternate paths. These alternate paths are pruned by negative reinforcement because they consistently deliver events with higher latency. As a result, the larger network expends less energy without suppression. We believe that suppression also exhibits the same behavior, but the energy difference is relatively small.

The second mechanism whose benefits we quantify is negative reinforcement. This mechanism prunes off higher latency paths, and can contribute significantly to energy savings. In this experiment, we selectively turn off negative reinforcement and compare the performance of directed diffusion with and without reinforcement. Intuitively, one would expect negative reinforcement to contribute significantly to energy savings. Indeed, as Figure 3.5(a) shows, diffusion without negative reinforcement expends nearly twice as much energy as when negative reinforcement is employed. This suggests that even our conservative negative reinforcement rules prune off paths which deliver consistently higher latency.

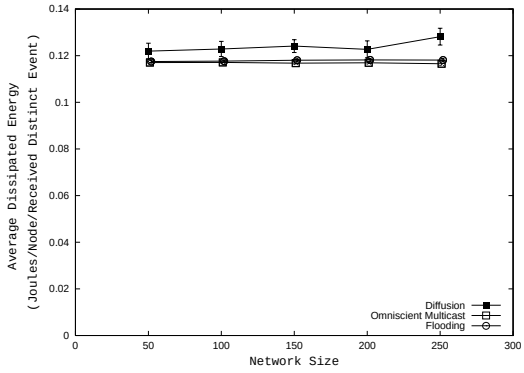
In the absence of negative reinforcement or suppression, diffusion's delay increases by factors of three to eight (the graphs are not included for lack of space). This is an artifact of the 802.11 MAC layer. In diffusion, data traffic is transmitted using MAC unicast. As more paths are used (in the absence of negative reinforcement), or more copies of data are sent (without suppression), MAC-layer channel contention increases, resulting in backoffs and subsequent delays.



(a) Negative reinforcement



(b) Duplicate suppression



(c) High idle radio power

Figure 3.5: Impact of various factors on directed diffusion.

3.2.5 Sensitivity Analysis

Finally, we evaluate the sensitivity of our comparisons (Section 3.2.2) to other factors: our choice of energy model, number of sinks, and size of source region.

In our comparisons, we selected radio power dissipation parameters to more closely mimic realistic sensor radios [Kai99]. We re-ran the comparisons of Section 3.2.2, but with power dissipation comparable to the AT&T Wavelan: 1.6W transmission, 1.2W reception and 1.15W idle [SK97]. In this case, as Figure 3.5(c) shows, the distinction between the schemes disappears. In this regime, we are better off flooding all events. This is because idle time energy utilization completely dominates the performance of all schemes. This is the reason why sensor radios try very hard to minimize listening for transmissions.

In our comparisons, we used 5 sinks. How sensitive is our comparison to this choice? We re-ran our simulations of Section 3.2.2 with 25% of the nodes as sinks. As Figure 3.6 shows, diffusion's energy efficiency is more marked than with fewer sinks. Intuitively, this is because, with a higher fraction of the sensor nodes as sinks, fewer non-sink nodes are explored by diffusion's reinforcement rules. In this situation, diffusion is more energy efficient because it aggressively suppresses duplicates. However, unlike our earlier comparisons, flooding dissipates only twice as much energy as diffusion. Furthermore, omniscient multicast has energy efficiency comparable to flooding. Both these observations are attributable to using MAC-layer unicast for neighbor-to-neighbor transmissions in diffusion and omniscient multicast. With more sinks, and consequently increased branching at each node, each node transmits the same packet multiple times, once to each neighbor. Diffusion is less impacted

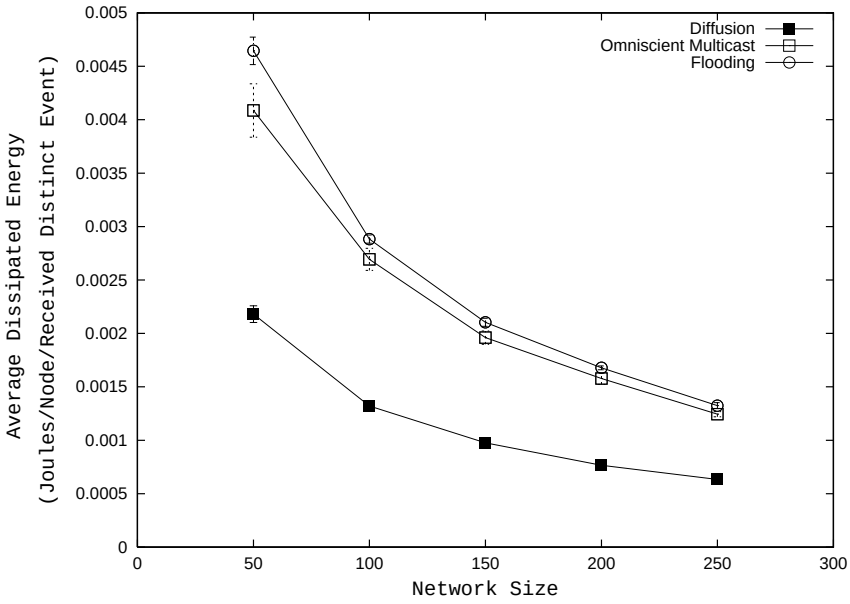


Figure 3.6: Impact of more sinks on energy efficiency.

by this because it performs in-network suppression of identical data. Finally, with more sinks and larger sensor fields, the event delivery ratio of diffusion falls to 80% (not shown). This can be attributed to suppression. Especially when the paths from sources to a sink overlap significantly, diffusion will suppress duplicates close to the sources and event losses nearer the sinks cannot be recovered.

We also conducted an experiment where the sources in our comparisons of Section 3.2.2 were uniformly distributed over the entire sensor field. For our choice of parameters, the results were not sensitive to this change.

3.3 Testbed Experimentation

In Section 3.1 and 3.2, our results suggest that application-specific processing inside the network is essential for wireless sensor networks where energy resources are limited. Examples of such processing include application-specific duplicate suppression, data aggregation, and data fusion. The approaches described in this dissertation are useful if they can be efficiently implemented and improve the energy-efficiency of distributed systems such as sensor networks. In this section, we discuss application techniques for sensor networks in details and measure the benefits of in-network aggregation over our operational testbed. We also discuss how our diffusion architecture enables other types of in-network processing, such as nested queries.

3.3.1 Application Techniques for Sensor Networks

In this section, we consider application techniques which illustrate how topology-independent low-level naming and in-network processing can be used to build efficient applications for sensor networks. The first approach we examine is application-specific data aggregation, an example of how in-network processing can reduce data traffic to conserve energy. Finally, we describe how our diffusion architecture enables other types of in-network processing, such as nested queries whereby one sensor cues another.

3.3.1.1 In-network data aggregation

An anticipated sensor application is to query a field of sensors and then take some action when one or more of the sensors is activated. For example, a surveillance

system could notify a biologist if an animal enters a region. Coverage of deployed sensors will overlap to ensure robust coverage, so one event will likely trigger multiple sensors. All sensors will report detection to the user, but communication and energy costs can be reduced if this data is aggregated as it returns to the user. Data can be aggregated to a binary value (there was a detection), an area (there was a detection in quadrant 2), or with some application-specific aggregation (seismic and infrared sensors indicate 80% chance of detection).

Although details of aggregation can be application-specific, the common systems problem is the design of mechanisms for establishing data dissemination paths to the sensors within the region, and for aggregating responses. Consider how one might implement this kind of data fusion in a traditional network with topologically-assigned low-level node names. First, in order to determine which sensors are present in a given region, a binding service must exist which, given a geographical region, lists the node identifiers of sensors within that region. Once these sensors are tasked, an election algorithm must dynamically elect one or more network nodes to aggregate the data and return the result to the querier.

Instead, our architecture allows us to realize this using *opportunistic* data aggregation. Sensor selection and tasking is achieved by naming nodes using geographic attributes. As data is sent from the sensors to the querier, intermediate sensors in the return path identify and cache relevant data. This is achieved by running application-specific code. These intermediate nodes can then suppress duplicate data by simply not propagating it, or they may slightly delay and aggregate data from multiple sources. We are also experimenting with influencing the dynamic selection of aggregation points to minimize overall data movement (Chapter 4).

Opportunistic data aggregation benefits from several aspects of our approach. Diffusion provides the *filter* software architecture to inject application-specific code into the network. (A complete description of filters is outside the scope of this thesis; for details see [HSI+01, CHG+01] and Appendix A.) Attribute naming and matching allow this application-specific code to remain inactive until triggered by relevant data. A common attribute set means that the application-specific code incurs no network costs to interact with directory or mapping services.

In Section 3.2, we analyzed the performance of diffusion with and without aggregation through simulation [IGE00]. In Section 3.3.2.1 we evaluate our implementation of this over real sensor nodes and validate our initial results with laboratory tests.

3.3.1.2 Nested queries

Real-world events often occur in response to some environmental change. For example, a person entering a room is often correlated with changes in light or motion, or a flower's opening with the presence or absence of sunlight. Multi-modal sensor networks can use these correlations by triggering a secondary sensor based on the status of another, in effect nesting one query inside another. Reducing the duty cycle of some sensors can reduce overall energy consumption (if the secondary sensor consumes more energy than the initial sensor, for example as an accelerometer triggering a GPS receiver) and network traffic (for example, a triggered imager generates much less traffic than a constant video stream). Alternatively, in-network processing might choose the best application of a sparse resource (for example, a motion sensor triggering a steerable camera).

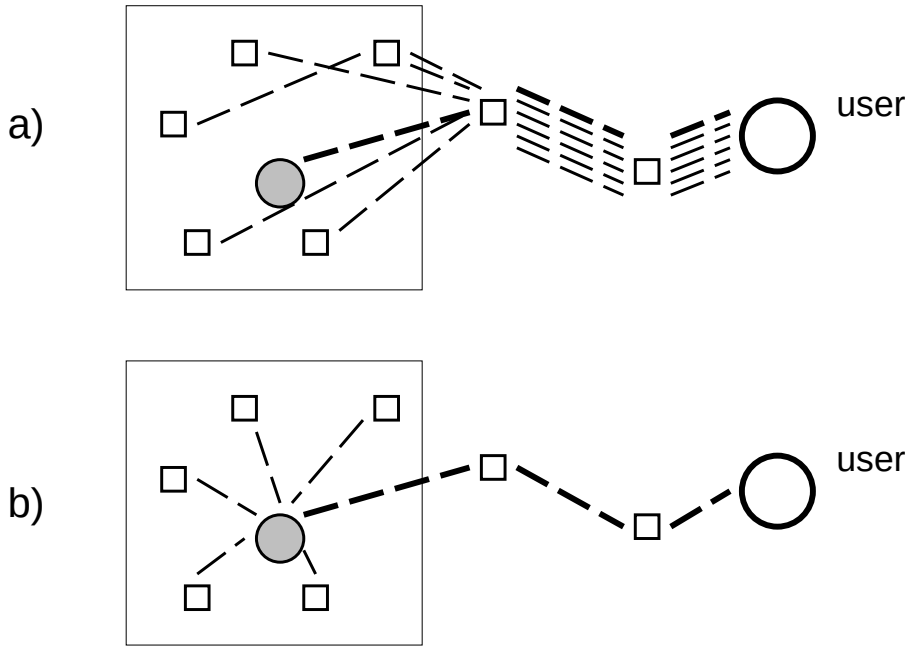


Figure 3.7: Two approaches to implementing nested queries.

Figure 3.7 shows two approaches for a user to cause one sensor to trigger another in a network. In both cases, we assume sensors know their locations and not all nodes can communicate directly. Part (a) shows a direct way to implement this: the user queries the initial sensors (small squares), when a sensor is triggered, the user queries the triggered sensor (the small gray circle). The alternative shown in part (b) is a nested, two-level approach where the user queries the triggered sensor which then sub-tasks the initial sensors. This nested query approach grew out of discussions with Philippe Bonnet and embedded database query optimization in his COUGAR database [BGMS99].

The advantage of a nested query is that data from the initial sensors can be interpreted directly by the triggered sensor, rather than passing through the user. In monitoring applications the initial and triggered sensors would often be quite close

to each other (to cover the same physical area), while the user would be relatively distant. A nested query localizes data traffic near the triggering event rather than sending it to the distant user, thus reducing network traffic and latency. Since energy-conserving networks are typically low-bandwidth and may be higher-latency, reduction in latency can be substantial, and reductions in aggregate bandwidth to the user can mean the difference between an overloaded and operational network. The challenges for nested queries are how to robustly match the initial and triggered sensors and how to select a good triggered sensor if only one is desired.

Implementation of direct queries is straightforward with attribute-addressed sensors. The user subscribes to data for initial sensors and, when something is detected, the user requests the status of the triggered sensor (either by subscribing or asking for recent data). Direct queries illustrate the utility of predefined attributes identifying sensor types. Diffusion may also make use of geography to optimize routing.

Nested queries can be implemented by enabling code at each triggered sensor that watches for a nested query. This code then sub-tasks the relevant initial sensors and activates its local triggered sensor on demand. If multiple triggered sensors are acceptable but there is a reasonable definition of which one is best (perhaps, the most central one), it can be selected through an election algorithm. One such algorithm would have triggered sensors nominate themselves after a random delay as the "best", informing their peers of their location and election (this approach is inspired by SRM repair timers [FJ95]). Better peers can then dispute the claim. Use of location as an external frame of reference defines a best node and allows timers to be weighted by distance to minimize the number of disputed claims.

In Section 3.3.2.2, we report the nested-query experiments by the SCADDS group over an operational testbed.

3.3.2 Performance Evaluation

In Section 3.2, we have observed that in-network processing is important to the performance of directed diffusion [IGE00]. In this section, we validate these results with an actual implementation of a simple surveillance application using attribute-based names and application-specific code.

3.3.2.1 Aggregation benefits

We examined in-network aggregation in our testbed of 14 PC/104 sensor nodes distributed on two floors of ISI (Figure 3.8). These sensors are connected by Radiometrix RPC modems (off-the-shelf, 418 MHz, packet-based radios that provide about 13kb/s throughput) with 10dB attenuators on the antennas to allow multi-hop communications in our relatively confined space.

The exact topology varies depending on the level of RF activity, and the network is typically 5 hops across.

To evaluate the effect of aggregation we placed a sink on one side of the topology ("D" at node 28) and then placed data sources on the other side ("S" at nodes 25, 16, 22, and 13), typically 4 hops apart. All sources generate events representing the detection of some object at the rate of one event every 6 seconds. For experiment repeatability events are artificially generated, rather than taken from a physical sensor and signal processing. Each event generates a 112 bytes message and is given sequence numbers that are synchronized at experiment start. An operational

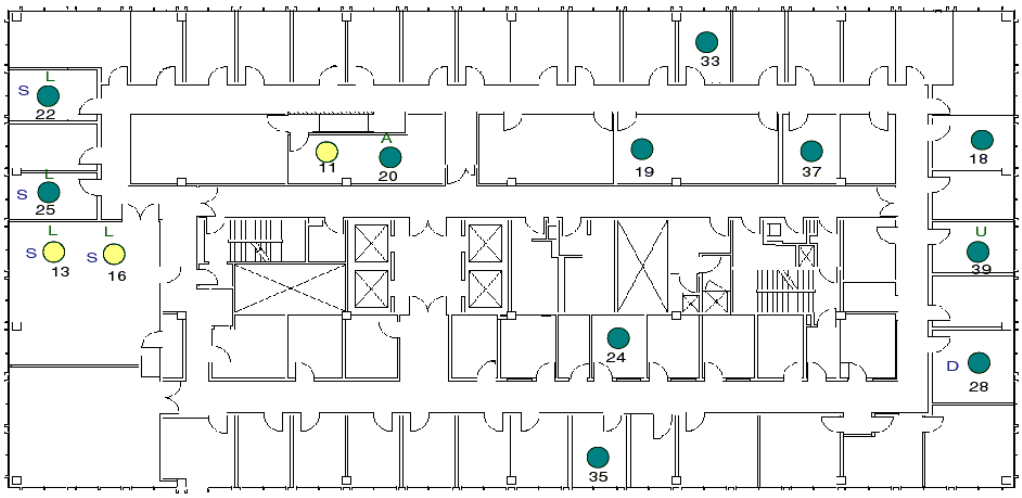


Figure 3.8: Node positions in our sensor testbed.

sensor network would use timestamps instead of sequence numbers. Both require synchronization, but time can be synchronized globally with GPS or NTP. We use sequence numbers because at the time of this experiment we had not synchronized our clocks. Experimentally, other than synchronization overhead, sequence numbers and timestamps are equivalent. All nodes were configured with application-specific aggregation code that passes the first unique event and suppresses subsequent events with identical sequence numbers. Although this scenario abstracts some details of a complete sensor network (for example, real signal processing may have different sensing delays), we believe it captures the essence of the networking component of multi-sensor aggregation.

We would like to compare the energy expended per received event. Unfortunately, we cannot measure that directly for two reasons. First, we do not have hardware to directly measure energy consumption in a running system. Second, we have previously observed that choice of MAC protocol can completely dominate energy

measurements. In low power radios, MAC protocols that do not sleep periodically are dominated by the amount of time spent listening, regardless of choice of protocol. Thus energy-conscious protocols like PAMAS [SR98] or TDMA are necessary for long-lived sensor networks. The SCADDS group is currently experimenting with power-aware MAC approaches [YHE02].

Although we currently cannot measure energy consumption on an appropriate MAC, we can estimate the effectiveness of reducing traffic for MACs with different duty cycles. A simple model of energy consumption is:

$$p_a = dp_\ell t_\ell + p_r t_r + p_s t_s$$

where p and t define the relative power and time spent listening, receiving, and sending and d is defined as the required listen duty cycle (the fraction of time the radio must be listening to receive all traffic destined to it). We found our sensor network contained pockets of severe congestion, but in the aggregate, radios listen:receive:send times were about 1:3:40. Relative energy consumption of listen:receive:send has been measured at ratios from 1:1.05:1.4 to 1:2:2.5 [SK97, XHE01]. For simplicity, assume energy consumption ratios of 1:2:2. With these parameters, energy usage for nodes with a duty cycle of 1 are completely dominated by energy spent listening. At duty cycle of 22% half of the energy is spent listening. Duty cycles of 10% begin to be dominated by send cost. Duty cycle for most radios today is 100%, but TDMA radios such as in WINSng nodes [PK00] may have duty cycles of 10–15% for non-base-stations. This analysis illustrates the importance of energy-conserving MAC protocols.

Since we cannot directly measure energy per event, Figure 3.9 measures bytes sent from diffusion in all nodes in the system normalized to the number of distinct events received. Each point in this graph represents the mean of five 30-minute experiments with 95% confidence intervals. Performance with one source is basically identical with and without suppression (this form of aggregation). As expected, suppression requires less data per event with multiple sources than experiments without suppression. With suppression the amount of traffic is roughly constant regardless of the number of sources. This application-specific data aggregation shows the benefit of in-network processing. It also shows that diffusion is useful for point-to-multipoint communication, since traffic represents both data and control traffic. Comparing traffic with and without suppression shows that suppression is able to reduce traffic by up to 42% for four sources. The network exhibits very high loss rates at that level of traffic. Our current MAC is quite unsophisticated, performing only simple carrier detection and lacking RTS/CTS or ARQ. Since all messages are broken into several 27-byte fragments, loss of a single fragment results in loss of the whole message, and hidden terminals are endemic to our multi-hop topology, this MAC performs particularly poorly at high load. The SCADDS group is currently working on a better MAC protocol.

We can confirm these results with a simple traffic model. We approximate all messages as 127B long and add together interest messages (sent every 60s and flooded from each node), reinforcement messages (sent on the reinforced path between the sink and each source), simple data messages (9 out of every 10 data messages, sent only on the reinforced path, and either aggregated or not), and exploratory data messages (1 out of every 10 data messages, sent from each source and flooded in turn

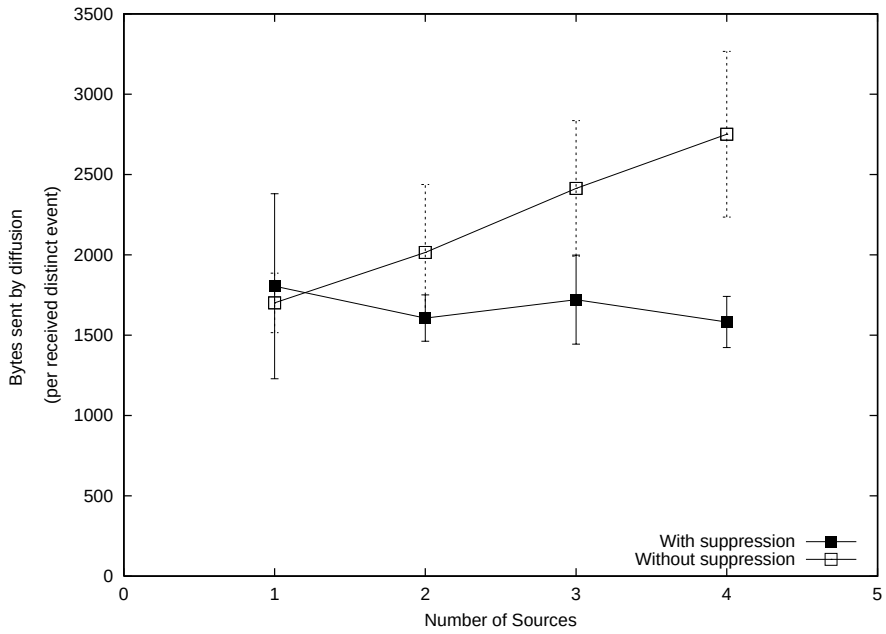


Figure 3.9: Bytes sent from all diffusion modules.

from each node, again possibly aggregated). If data messages are not aggregated, each source incurs the cost of the full path, while if data messages are aggregated after the first hop each incurs one hop cost to the aggregation point and then one message will travel on to the sink. Summing the message cost and normalizing per event we expect aggregation to provide a flat 990B/event independent of the number of sources, and we expect bytes sent per event to increase from 990 to 3289B/event without aggregation as the number of sources rise from 1 to 4.

The shape of this prediction matches our experimental results, but in absolute terms it underpredicts the B/event of aggregation and overpredicts the 4-source/no-aggregation case. We believe these differences are due to MAC-layer collisions in the experiment that tend to drive bytes-per-event to the middle. Only 55–80% of events generated in the experiment were delivered to the sink, so bytes-per-event in

less congested portions of the experiment (with one source or aggregation) is high because traffic is normalized over fewer events. On the other hand, with four sources and no aggregation, we believe collisions happen very near the data sources and so the aggregate amount of data sent is lower that predicted. In addition, we sometimes observe longer paths in experiment than we expected.

These experimental measurements of aggregation are also useful to validate our simulation experiments that consider a wider range of scenarios (Section 3.2). Our simulation studies have shown that aggregation can reduce energy consumption by a factor of 3–5× in a large network (50–250 nodes) with five active sources and five sinks (Figure 3.5(b)). Although care must be used in comparing energy to bytes sent, a 3–5-fold energy savings with five sources is much greater than the 42% (or 1.7-fold) traffic savings we observe with four sources. The primary reason for this difference is differences in ratio of exploratory to data messages in these systems. Exploratory messages (low-data rate messages) are used to select good gradients and so are flooded to all nodes. Data messages (high-rate messages) are sent only on reinforced gradients forming a path between the sources and sinks. In simulation the ratio of exploratory to data messages sent from a source was about 1:100 (exploratory messages were sent every 50s, data every 0.5s, messages were modeled as 64B packets). In our testbed this ratio was about 1:10 (exploratory messages every 60s, data every 6s, with messages of roughly the same size). Increasing this ratio in experiment was not possible given our small radio bandwidth (13kb/s rather than 1.6Mb/s in simulation) while keeping reasonable experimental running times. This large difference in ratios is consistent with the large difference in energy or traffic savings.

A potential disadvantage of data aggregation is increased latency. The effect of aggregation on latency is strongly dependent on the specific, application-determined aggregation algorithm. The algorithm used in these experiments does not affect latency at all, since we forward unique events immediately upon reception and then suppress any additional duplicates (incurring only the additional negligible cost of searching for duplicates). Other aggregation algorithms, such as those that delay transmitting a sensor reading with the hope of aggregating readings from other sensors, can add some latency. Understanding aggregation and sensor fusion algorithms is an important area of future work.

3.3.2.2 Nested query benefits

In Section 3.3.1.2 we suggested that nested queries could reduce network costs and latency, and we argued that nested queries could be implemented using attributes and application-specific code. To validate our claim about the potential performance benefits of this implementation, the SCADDS group measured the performance of an application that uses nested queries against one that does not.

The application is similar to that described in Section 3.3.1.2 and Figure 3.7: a user requests acoustic data correlated with (triggered by) light sensors. The SCADDS group reuses our PC/104 testbed (Figure 3.8) placing the user "U" at node 39, the audio sensor "A" at node 20, and light sensors "L" at nodes 16, 25, 22, and 13. It is one hop from the light sensors to the audio sensor, and two hops from there to the user node. To provide a reproducible experiment, the SCADDS group simulates light data to change automatically every minute on the minute. Light sensors report their state every 2s (no special attempt is made to synchronize or

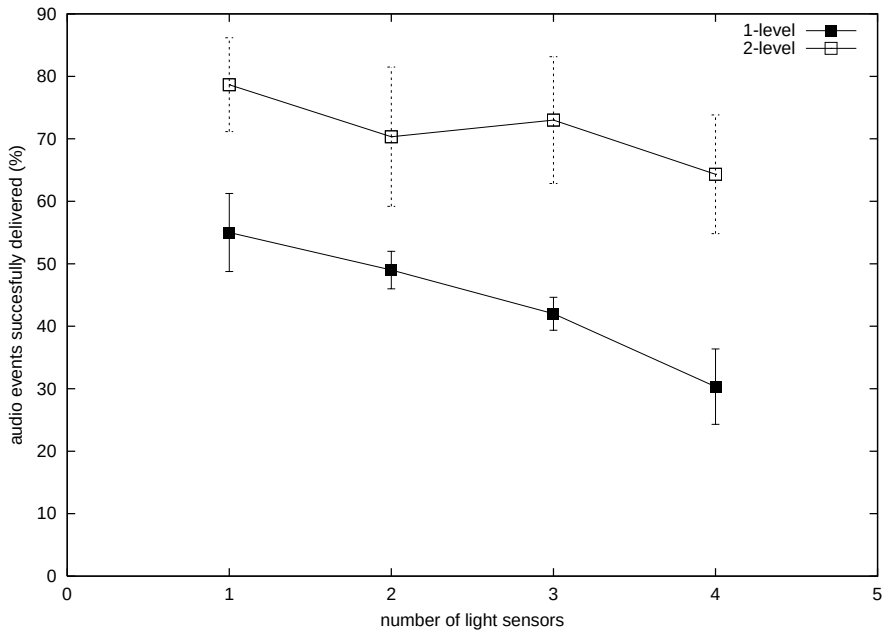


Figure 3.10: Percentage of audio events successfully delivered to the user.

unsynchronize sensors). Audio sensors generate simulated audio data each time any light sensor changes state. Light and audio data messages are about 100 bytes long.

Figure 3.10 shows the percentage of light change events that successfully result in audio data delivered to the user. (Data points represent the mean of three 20-minute experiments and show 95% confidence intervals.) The total number of possible events are the number of times all light sources change state and a successful event is audio data delivered to the user. These delivery rates do not reflect per-hop message delivery rates (which are much higher), but rather the cumulative effect of sending best-effort data across three or five hops for nested or flat queries, respectively.

This system is very congested, and as described above (Section 3.3.2.1), our primitive MAC protocol exaggerates the impact of congestion. Missing events translate into increased detection latency. Although a sensor network could afford to miss a

few events (since they would be retransmitted in the next time the sensor is measured), these loss rates are unacceptably high for an operational system.

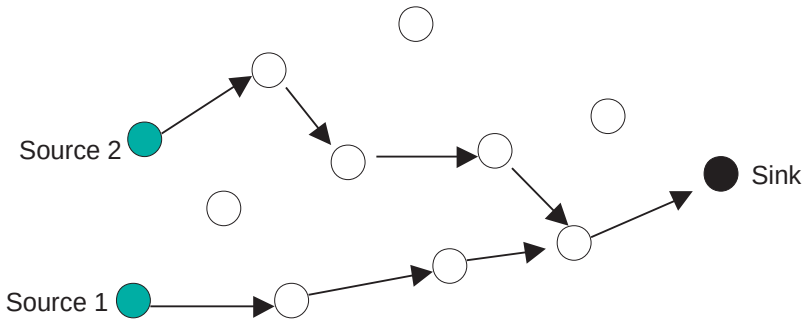
However, this experiment sharply contrasts the bandwidth requirements of nested and flat queries. Even with one sensor the flat query shows significantly greater loss than the nested query because both light and audio data must travel to the user. Both flat and nested queries suffer greater loss when more sensors are present, but the one-level query falls off further. Comparing the delivery rates of nested queries with one-level queries shows that localizing the data to the sensors is very important to parsimonious use of bandwidth. In an uncongested network we expect that nested queries would allow operation with a lower level of data traffic than one-level queries and so would allow a lower radio duty cycle and a longer network lifetime.

Chapter 4

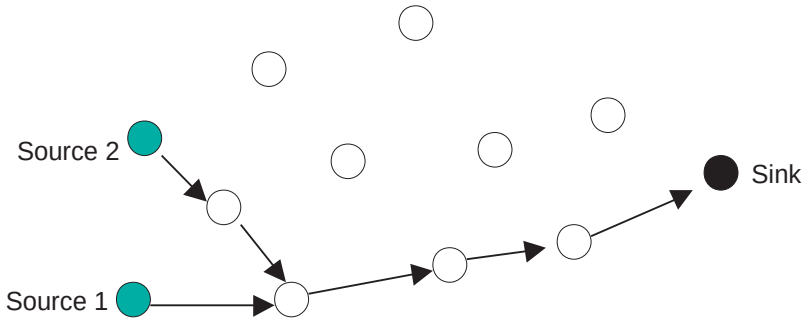
Greedy Aggregation

The instantiation of directed diffusion described in Chapter 2 establishes low-latency paths between sources (sensor nodes that detect phenomena) and sinks (user nodes) using only localized algorithms. Paths from different sources to a sink form an *aggregation tree* rooted at the sink. Data from different sources is *opportunistically* aggregated. Whenever similar data happens to meet at a branching node in the tree, the copies of similar data are replaced by a single message. Energy-wise, opportunistic aggregation on a low-latency tree is not optimal because data may not be aggregated (or reduced) near the sources. (Figure 4.1). To save more energy, one could design another instantiation of diffusion which favors path selection to increase early sharing of paths and reduce energy consumption.

Sometimes, there are drawbacks for early path sharing: higher latency and less robustness. Some paths will be a bit longer to trade off for early aggregation. Given that an aggregate generally contains more information than a non-aggregated message, a loss of an aggregate adversely impacts the quality of the result more than a loss of a non-aggregated message. Conversely, there are practical constraints that



(a) Late Aggregation



(b) Early Aggregation

Figure 4.1: An Example of Late Aggregation and Early Aggregation.

favor early path sharing. For example, early aggregation may reduce overall traffic which is preferable, given the limited bandwidth (Section 4.3).

Nevertheless, it is not obvious that such path optimization will practically lead to significant energy savings. Specifically, the additional cost of establishing the optimal aggregation tree could be high. Assuming perfect aggregation (*i.e.,* the data size of an aggregate is equal to the data size of an individual event), the cost of the tree is the number of links on the tree. Therefore, finding the optimal aggregation tree is computationally infeasible because it is equivalent to finding the Steiner tree that is known to be NP-hard [GP68, Win87]. Thus, we do not expect to get an optimal tree. Moreover, given imperfect aggregation and a sub-optimal aggregation tree, energy savings might not be much better than those of opportunistic aggregation.

One promising heuristic is a greedy incremental tree (GIT) [TM80]. To construct a greedy incremental tree, a shortest path is established for only the first source to the sink whereas each of the other sources is incrementally connected at the closest point on the existing tree. In this chapter, we propose a new instantiation of directed diffusion that uses a GIT-like algorithm (Section 4.2) to improve path sharing. We have implemented this *greedy-tree* approach [IEGH02] in *ns-2* and we compare it to our prior *opportunistic* approach (Section 4.3). Our result suggests that although the two are roughly equivalent in low-density networks, greedy aggregation can achieve significant energy savings in high-density networks. Recent work has compared the greedy incremental tree with the shortest path tree (SPT) using *abstract simulations* [KEW02]. Based on the event-radius model and the random sources model, their results indicate that the transmission savings by the GIT over the SPT do not exceed 20%. However, the energy savings of our greedy aggregation

can definitely be *much higher than 20%*, given our source placement schemes and high-density networks (Section 4.3.4).

4.1 Data Aggregation and Directed Diffusion

Once sensors detect events of interest, they disseminate the events to users. Intermediate nodes may aggregate several events into a single event to reduce transmissions and total data size for system resource savings. The total size reduction mainly depends on data characteristics, event representations, and applications. Only data aggregation that leads to total size reduction is considered and justified in this dissertation. Furthermore, data aggregation will also reduce the number of transmissions. As a result, the total per-transmission overhead (*e.g.,* packet headers, MAC control packets) will be reduced and the energy savings will be even more evident.

Similar to data compression, data aggregation can be classified into two approaches (*i.e.,* lossless and lossy). With *lossless aggregation*, all detailed information is preserved. However, according to information theory, the total size reduction is upper bounded by entropy (*i.e.,* a measure of how much information is encoded in a message). The basic principle of data reduction is to eliminate redundant information. Given that events may be highly correlated, there will be a significant amount of redundancy among them. Unlike lossless aggregation, *lossy aggregation* may discard some detailed information or degrade data quality for more energy savings.

Examples of lossless aggregation are timestamp aggregation and packing aggregation. Timestamp aggregation can be used in a remote surveillance application where an event consists of several attributes including a timestamp. Distinct events may

actually be temporally correlated (within seconds of one another). The redundant information (*e.g.,* the hour and minute field in the timestamp) may be minimized (*i.e.,* not repeated) using an efficient representation for aggregated events. For packing aggregation, several non-aggregated messages are packed into one aggregate without compression. The only savings are the total per-transmission overhead, such as packet headers.

An example of lossy aggregation is outline aggregation used in eScan [ZGE02]. The goal of eScan is to depict the remaining energy levels of sensor nodes. Leveraging the spatial locality of energy usage, topologically adjacent nodes can sometimes be approximately represented by a bounding polygon. This approach trades off some inaccuracy in node energy representation for reduced energy usage.

Directed diffusion was designed with application-level data processing in mind. Unsurprisingly, directed diffusion can take advantage of data aggregation due to in-network data processing capability. Nevertheless, some directed diffusion mechanisms can be adjusted to achieve even more benefit out of data aggregation. We expect that, with the proper interactions between data aggregation rules and reinforcement rules, diffusion will establish energy efficient paths rather than low delay paths. In particular, if aggregated data are distinguishable from non-aggregated data, it will be possible to design reinforcement rules that favor aggregated data paths over non-aggregated data paths. Those rules will encourage path sharing and achieve even more energy saving due to more data aggregation (see Section 4.2).

Energy savings of data aggregation depend on reduction in total data size. If the total data size is rarely reduced after aggregation, a shortest path will be more energy efficient than an aggregated data path. Moreover, without a reasonable

reduction in total data size, aggregated data paths introduce traffic concentration (and possibly also congestion) which adversely impacts network lifetime. In this scenario, path optimization is not worth performing. Conversely, if the total data size is dramatically reduced after aggregation, it will be reasonable to trade off delay for energy efficiency by favoring longer but aggregated data paths over shorter but non-aggregated data paths. A *gradient map* (*i.e.,* data gradients from sources to sinks) similar to a greedy tree would be preferred. Specifically, aggregation points need to be carefully selected (using reinforcement) so that additional dissipated energy due to longer paths does not exceed energy savings due to total data size reduction.

4.2 Greedy Aggregation

Greedy aggregation is a novel diffusion approach to construct a greedy incremental tree for data aggregation. This approach differs from the previous diffusion approach (*i.e.,* opportunistic aggregation on a lowest latency tree, see Chapter 2) in path establishment and maintenance. To construct a greedy incremental tree, a shortest path is established for only the first source to the sink whereas each of the other sources is incrementally connected at the closest point on the existing tree.

4.2.1 Path Establishment

Directed diffusion establishes paths using *path reinforcement*; a node in the network may locally decide (based possibly on perceived traffic characteristics) to draw data from one or more neighbors in preference to other neighbors. In directed diffusion,

exploratory samples have initially and repeatedly been flooded throughout the network. The sink then has some empirical information about which of its neighbors can provide it with the highest quality data (lowest loss or lowest delay). To this preferred neighbor, it sends out a reinforcement. That neighbor then locally determines its most preferred neighbor in the direction of the source, and so on. In our prior opportunistic approach, the most preferred neighbor is a neighbor which has delivered samples with the lowest delay.

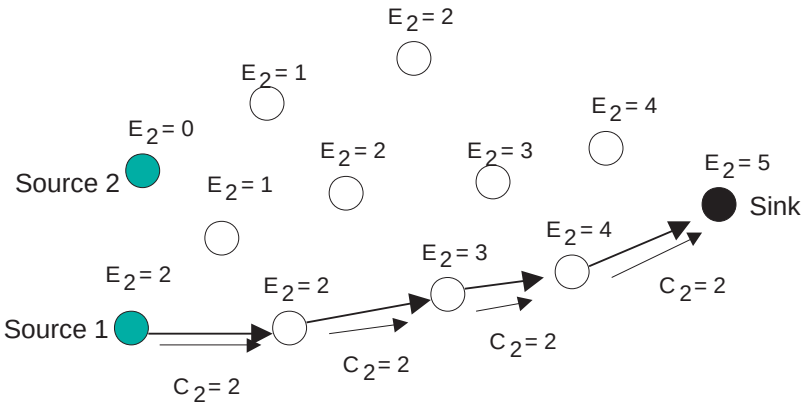
For opportunistic aggregation, the sink initially diffuses an interest for an *exploratory* event (*i.e.,* a low event-rate notification) intended for path establishment and repair. We call the gradients set up for sending exploratory events *exploratory gradients*. Once a source detects a matching target, it sends exploratory events, possibly along multiple paths, toward the sink. After the sink receives these exploratory events, it *reinforces* one particular neighbor in order to "draw down" real data (*i.e.,* events at a higher data rate that allow high quality tracking of targets). We call the gradients set up for sending high quality tracking events *data gradients*. The local rule for selecting an *empirically low delay path* is to reinforce any neighbor from which a node receives a previously unseen exploratory event.

For our greedy aggregation, each event contains an additional attribute E, an energy cost for delivering this event from the source to the current node. Once a source detects phenomena, it sends exploratory events with $E = 1$ (energy unit for one-hop message transmission) to each neighbor for whom it has a gradient. After a node receives these previously unseen exploratory events, it adds the cost of that transmission to E before resending. Since our radios have fixed transmission power, we measure energy as equivalent to hops, but direct measures of variable energy

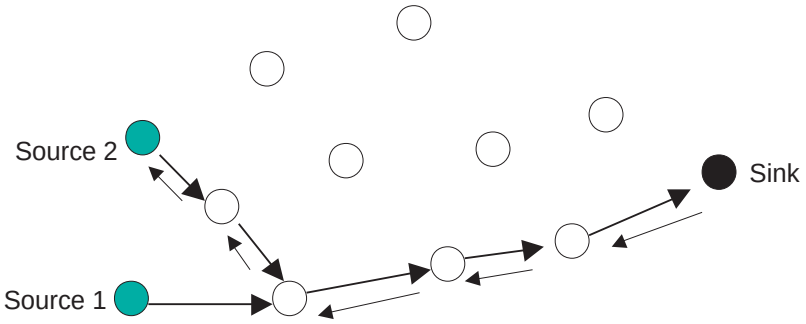
could also be used. Although this energy attribute is useful for selecting a lowest-energy path, it is not sufficient for constructing a greedy incremental tree because there is no path-sharing information.

To provide such information, each source on the existing tree (*i.e.,* a source with data gradients) also generates an *incremental cost message*, once it receives a previously unseen exploratory message generated by other sources. The incremental cost message contains the random message id of the corresponding exploratory event and the incremental energy cost C required for delivering that exploratory event to the existing tree. This incremental cost message is only sent along the existing tree using data gradients. Unlike the energy cost E, the incremental energy cost C can only be decreased (in order to find the closest point on the existing tree). Once an on-tree node receives a previously unseen incremental cost message, it searches for the corresponding exploratory event in its message cache. The node updates C of the incremental cost message to the minimum value between the current C and the energy cost E' of the exploratory event retrieved from the cache, before sending the incremental cost message to its outgoing data gradients.

Once a sink receives a previously unseen exploratory event, it does not reinforce a neighbor immediately, because a lowest-delay path is not necessarily an energy-efficient path. Instead, a reinforcement timer of T_p is set up. After the timer expires, the sink reinforces any neighboring node that sent the exploratory event or the incremental cost message (of the corresponding exploratory event) at the lowest energy cost. If the energy cost of an exploratory event and the incremental cost message are equivalent, the sink reinforces the neighboring node that sent the exploratory event. Other ties are decided in favor of the lowest delay. When the



(a) An Incremental Cost Message



(b) Reinforcement

Figure 4.2: An Example of Path Establishment.

neighboring node receives the reinforcement message (associated with the random message id of the exploratory message), the node sets a data gradient toward the sink (or the node that sends the reinforcement) and reinforces an upstream neighboring node immediately using the above local rule without setting up a timer.

As a result, a greedy incremental tree can be constructed using the described local rule (see Figure 4.2 for example). Unlike the exploratory event, the incremental cost message contains the minimum energy cost of delivering data from a new source

to the existing tree (not to the sink). The path from the first source to the sink is a lowest energy path because of no existing tree or no incremental cost message generated yet. Each subsequent source is incrementally connected to the aggregation tree at the closest point (using the energy cost information provided by the incremental cost message). Hence, one might wonder if this algorithm requires unsynchronized sources so that each source can be incrementally connected to the tree. We do not make that assumption because sources can be synchronized if they are triggered by the same phenomena. Although we have described the algorithm using examples in which different sources start at different times, the algorithm in fact works when sources start transmitting events in near simultaneity. In that scenario, the algorithm initially constructs a lowest-energy-path tree (*i.e.,* each source is connected to the sink using the lowest-energy path), but this problem is not persistent. Since exploratory events are sent periodically, in a subsequent round of exploratory events, the greedy incremental tree will be constructed and the lowest-energy-path tree will be pruned off using the negative reinforcement mechanism in Section 4.2.3.

4.2.2 Data Aggregation and Set Covering Problem

To enable data aggregation, intermediate nodes will process or delay received data for a period of time T_a before sending them. This delay is crucial for data aggregation because multiple data will not be received at the same time. The delay should be selected based on the application or other system factors. For example, in a TDMA MAC, one might match the aggregation time to a multiple of the TDMA frame duration, or as a fraction of the periodicity of sensor data generation. Thus, unsent data can be periodically aggregated and appropriately delivered to neighbors using

data gradients. However, intermediate nodes do not necessarily delay received data for the same period of time. An intermediate node that receives a sufficient amount of data for aggregation does not need to delay the received data any further. In addition, an intermediate node that is not an aggregation point does not need to delay the data at all.

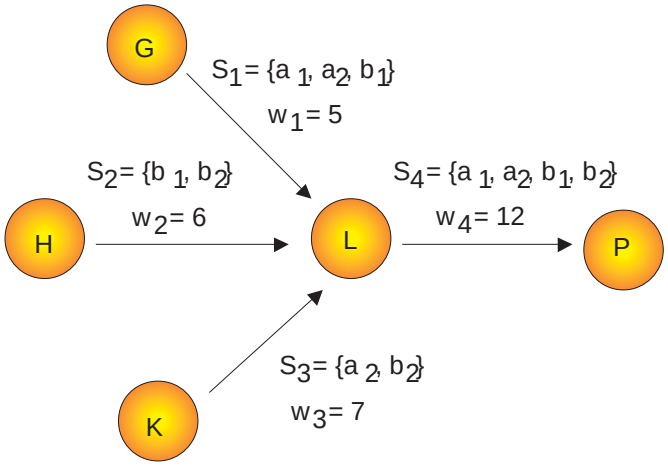
Similar to exploratory events, messages and aggregates also contain the energy cost attribute. After aggregating multiple messages into an aggregate, the final step before sending the aggregate is to compute the associated energy cost for that aggregate. This energy information will be used for empirical adaptation to energy-efficient paths. Specifically, the negative reinforcement rule uses this information for path pruning (Section 4.2.3). However, different neighbors might report aggregates of different subsets of data items, with varying costs. The challenge is to find the set of incoming aggregates which cover the data items at the smallest cost. Given incoming aggregates, it is not trivial to calculate the minimum energy cost of the outgoing aggregate because it is a *weighted set-covering problem* [BSK96, CLR90] (*i.e.,* a generalized version of an NP-hard set-covering problem).

An instance of the set-covering problem consists of a finite set $X = \{x_1, x_2, ..., x_m\}$ and a family F of subsets S_i of X, $F = \{S_1, S_2, ..., S_n\}$, such that every element of X belongs to at least one subset in F: $X = \cup_{i=1}^{n} S_i$. The regular set-covering problem is to determine a minimum-size subset $C \subseteq F$ whose members cover all of X, *i.e.,* $X = \cup_{S_i \in C} S_i$. For the weighted set-covering problem, each set S_i in the family F is associated with a weight w_i and the weight of a cover C is $\sum_{S_i \in C} w_i$. The problem is to find a minimum-weight cover (The regular set-covering problem is a special case of the weighted set-covering problem with $w_i = 1$ for all i).

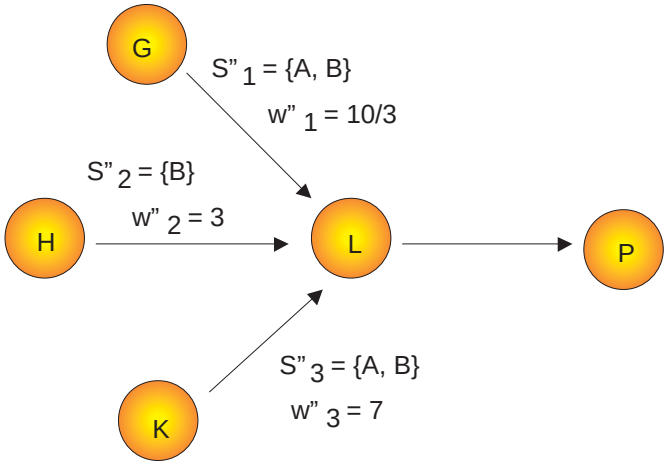
Our problem can be directly mapped into the weighted set-covering problem as follows. An incoming aggregate is a subset S_i whereas the outgoing aggregate is X. Each incoming aggregate is associated with the energy cost w_i. Therefore, the energy cost of the outgoing aggregate is the minimum weight of the cover plus 1.

Approximate algorithms for this problem include greedy heuristics [BSK96, CLR90], probabilistic methods [Sen93], genetic-algorithms-based heuristics [LHRP90], neural-networks-based techniques [GW97], and Lagrangian heuristics [Bea90]. We chose the greedy heuristic because of its high-quality solutions (The worst ratio between the cost of a greedy solution and the optimal solution is $1 + \ln d$ where d is the maximum size of any set S_i [CLR90, GW97]). The heuristic of the greedy set-covering algorithm is to greedily select the next subset (among the remaining subsets) for covering uncovered elements at the lowest cost ratio until all elements are covered. The cost ratio r_i of S_i is $w_i/|S_i'|$ where $S_i' \subset S_i$ is the set of uncovered elements in S_i. However, there might exist a subset S_k in C where all elements in S_k are covered by the union of other subsets in C. The final step of the greedy heuristic is to remove such redundant subsets from C.

For example (Figure 4.3(a)), node **L** receives incoming aggregates $S_1 = \{a_1, a_2, b_1\}$, $S_2 = \{b_1, b_2\}$, and $S_3 = \{a_2, b_2\}$. Suppose that $w_1 = 5, w_2 = 6$, and $w_3 = 7$ are the associated energy costs. Therefore, $r_1 = 5/3, r_2 = 6/2$, and $r_3 = 7/2$ are the initial cost ratios. Given that r_1 is the minimum cost ratio, S_1 is the first selected subset and the remaining uncovered element is b_2. At the second step, $r_2 = 6/1$ and $r_3 = 7/1$ because $S_2' = \{b_2\}$ and $S_3' = \{b_2\}$. Thus, S_2 is selected as the final subset of the cover. **L** then sends an outgoing aggregate $S_4 = S_1 \cup S_2 = \{a_1, a_2, b_1, b_2\}$ to **P** with associated energy cost $w_4 = w_1 + w_2 + 1 = 12$.



(a) Data Aggregation



(b) Negative Reinforcement

Figure 4.3: The Use of Weighted Set Covering Problems.

4.2.3 Path Pruning

The algorithm described in Section 4.2.1 can result in more than one path being reinforced due to synchronized sources and network dynamics. For energy efficiency, we need to prune unnecessary or inefficient paths. Our pruning mechanism is to *negatively reinforce* neighbors on those paths. To find the inefficient paths, our pruning rule might also need to compute the set cover. Given that different neighbors might report aggregates of different subsets of data items, with varying costs, the challenge is to find the set of neighbors who cover the data items at the smallest cost. A simple pruning rule is to negatively reinforce neighbors that are not in the set cover.

Specifically, one plausible choice for a pruning rule is to negatively reinforce neighbors from which no energy-efficient aggregates have been received within a window of N events or time T_n. The local rule we evaluate in Section 4.3 is based on a time window of T_n, chosen to be 2 seconds in our simulations (or 4 times of the aggregation delay T_a). An incoming aggregate is considered energy-efficient if it is selected as a subset in the set cover C. For example (Figure 4.3(a)), node **L** will negatively reinforce node **K** because S_3 is not in C. However, given that events a_i and b_i are generated by sources A and B, respectively, this direct approach is a bit conservative and energy inefficient (**G** will likely send b_2 in its next aggregate. Therefore, **H** should also be negatively reinforced).

Our more energy-efficient rule is to compute the set cover C of sources instead of events. To do this, each event in an aggregate is replaced by its source. To preserve the initial cost ratio, we maintain the event proportions by scaling the new

associated energy cost w_i'' of the transformed aggregate S_i''. Thus, $w_i'' = w_i * |S_i''|/|S_i|$. For example, the subsets of events in Figure 4.3(a) can be transformed to the subsets of sources in Figure 4.3(b). After the transformation, $S_1'' = \{A, B\}, S_2'' = \{B\}$, and $S_3'' = \{A, B\}$ whereas $w_1'' = 5 * 2/3, w_2'' = 6 * 1/2$, and $w_3'' = 7 * 2/2$. The cost ratios are still $r_1 = 5/3, r_2 = 3$, and $r3 = 7/2$. Using the greedy heuristic, S_1'' is selected as the only subset in C. Therefore, **L** negatively reinforces **H** and **K**.

When **H** receives this negative reinforcement, it degrades its gradient toward **L** (from a data gradient to an exploratory gradient). Furthermore, if all its gradients are now exploratory, **H** negatively reinforces those neighbors that have been sending data to it (as opposed to exploratory events). This sequence of local interactions ensures that the path through **H** is degraded rapidly.

4.3 Performance Evaluation

In this section, we report on some results from a simulation. We use a packet-level simulation to explore (in some detail) the implications of some of our design choices. This section describes our methodology, compares the performance of greedy aggregation against the opportunistic aggregation, then explores impact of network dynamics and other factors on simulation.

4.3.1 Methodology

We implemented our greedy aggregation in the *ns-2* simulator [BEF+00]. Our goals in conducting this evaluation study were four-fold: First, to verify the viability of greedy aggregation as an alternative instantiation of directed diffusion. Second, to

understand the impact of network dynamics (*e.g.,* node failures) on greedy aggregation and opportunistic aggregation. Third, to explore the influence of the source placement on the performance of both approaches. Finally, to study the sensitivity of greedy aggregation to the parameter choices.

We select three metrics to analyze the performance of greedy aggregation and to compare it to opportunistic aggregation: average dissipated energy, average delay, and distinct-event delivery ratio. These metrics were used in Section 3.2 to compare diffusion with other idealized schemes. We study these metrics as a function of sensor network density.

To completely specify our experimental methodology, we need to describe the sensor network generation procedure (*e.g.,* our choice of ratio parameters, our workload). In order to study the performance of greedy aggregation as a function of network density, we generate a variety of sensor fields in a 200m by 200m square. In most of our experiments, we study seven different sensor quantities, ranging from 50 to 350 nodes in increments of 50 nodes. Each sensor field is generated by uniformly randomly placing the nodes in the square. Each node has a radio range of 40m. Thus, the node density (expressed in terms of how many other nodes each node can hear on average) ranges from 6 to 43 neighbors. For each network size, our results are averaged over ten different generated fields. We evaluate greedy aggregation over a modified 802.11 MAC layer which was also used in Section 3.2.

Finally, in most of our simulations, we use a fixed workload which consists of five sources and one sink. The sources are randomly selected from nodes in a 80m by 80m square at the bottom left corner of the sensor field. The sink is randomly selected from nodes in a 36m by 36m square at the top right corner of the field. This

source placement scheme differs from the event-radius model [KEW02] for the low-level data aggregation because sources may not be triggered by the same phenomena and may not be within one hop from one another. Similar to the random source placement model [KEW02], our source placement scheme is intended for the high-level data aggregation except that all sources in our placement scheme are placed in a subregion of the sensor field far from the sink. Our greedy aggregation targets high-level data (*e.g.,* abstract event representation) because we expect that low-level data (*e.g.,* seismic signals) will be locally processed. Therefore, only the random source placement scheme and our source placement scheme are used in this evaluation, not the other models of [KEW02].

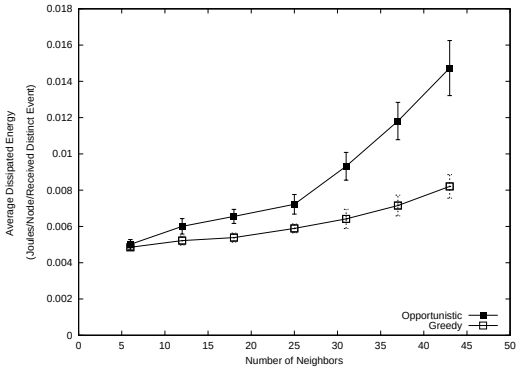
Each source generates two events per second. Thus, the aggregation delay T_a is set to 0.5 seconds. The rate for exploratory events was chosen to be one event in 50 seconds. Events were modeled as 64 byte packets and other messages were 36 byte packets. The size of aggregates depends on the aggregation function and the number of data items in the aggregates (Section 4.3.4). For perfect aggregation, the aggregate size is 64 bytes. Interests were periodically generated every 5 seconds and the gradient timeout was 15 seconds. We chose the window T_n for negative reinforcement to be 2 seconds and the timer T_p for positive reinforcement to be 1 second.

4.3.2 Comparative Evaluation

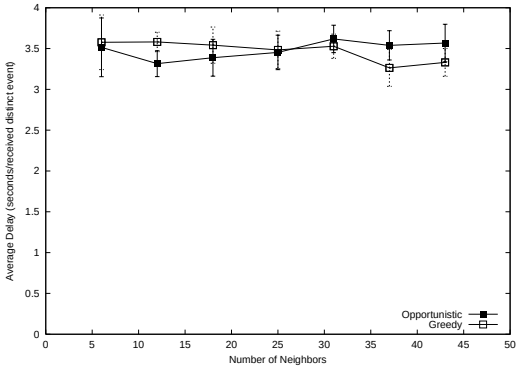
Our first experiment compares greedy aggregation to the opportunistic aggregation for data dissemination in sensor networks. Figure 4.4(a) shows the average dissipated energy per packet as a function of the network density. Assuming perfect

aggregation, greedy aggregation has noticeably better energy efficiency than the opportunistic approach for very high-density networks. For dense sensor fields (40 or more neighbors per node), its dissipated energy is only 55% that of opportunistic aggregation. The reason for this benefit is that, given a high-density network, there exist several shortest paths from a source to a sink. The available path diversity reduces the probability of path sharing among different sources for opportunistic aggregation. The dissipated energy for both approaches increases with network size due to some diffusion overhead (*e.g.,* flooding of interests and exploratory events).

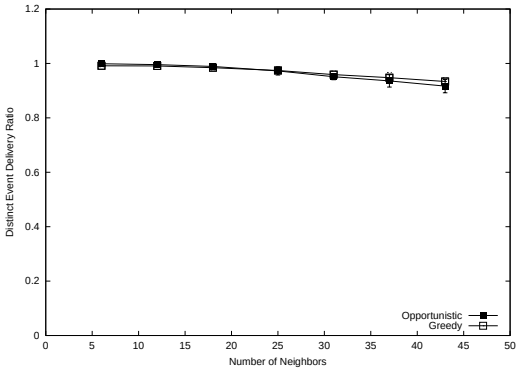
Figure 4.4(b) plots the average delay observed as a function of the network density. Greedy aggregation has a delay comparable to opportunistic aggregation. This is a bit surprising because we expected that greedy aggregation would trade off delay for energy efficiency. In low-density networks (fewer than 15 neighbors per node), greedy aggregation has a bit longer delay (up to 10% longer than that of opportunistic aggregation) because there are not many shortest paths. Thus, a shared path to a new source is unlikely to be a shortest path. Conversely, in high-density networks, the greedy approach has a bit lower delay because a shared path to a new source can still be close to a shortest path. Unlike greedy aggregation, the opportunistic approach does not encourage path sharing. Given random jitter delay and MAC fairness, the lowest-delay paths for sources are unlikely to be shared, particularly in high-density networks. Therefore, in high-density networks, opportunistic aggregation is less energy-efficient and the overall traffic reduction is less. As a result, the delay of opportunistic aggregation is a bit longer for the high-density networks, given a constant bandwidth.



(a) Average dissipated energy



(b) Average delay



(c) Distinct-event delivery ratio

Figure 4.4: Greedy aggregation compared to opportunistic aggregation.

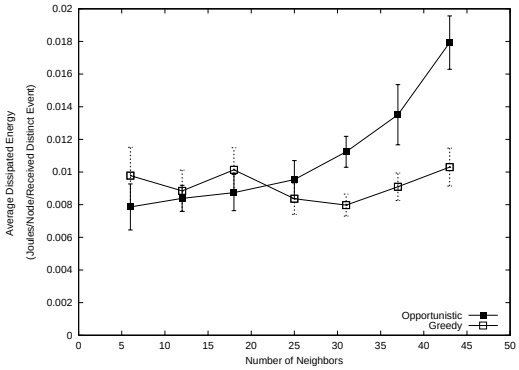
The result in Figure 4.4(c) indicates that, for these uncongested networks without network dynamics, both approaches achieve the similar distinct-event delivery ratio as expected.

In general, opportunistic aggregation performs sufficiently well unless the network is highly dense. However, such highly dense networks might not be common in practice unless the sensing range is significantly shorter than the transmission range.

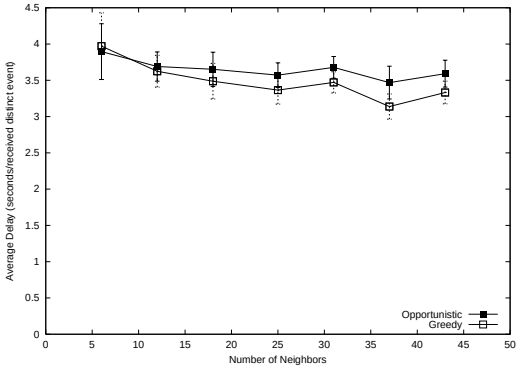
4.3.3 Impact of Network Dynamics

To study the impact of network dynamics on greedy aggregation and opportunistic aggregation, we simulated node failures as follows. Similar to Section 3.2.3, for each sensor field, we repeatedly turned off 20% of nodes for 30 seconds. These nodes were uniformly chosen from the sensor field. Our dynamics experiment imposes fairly adverse conditions for a data dissemination protocol. At any instant, 20% of the nodes in the network are unusable. Furthermore, we do not permit any "settling time" between node failures.

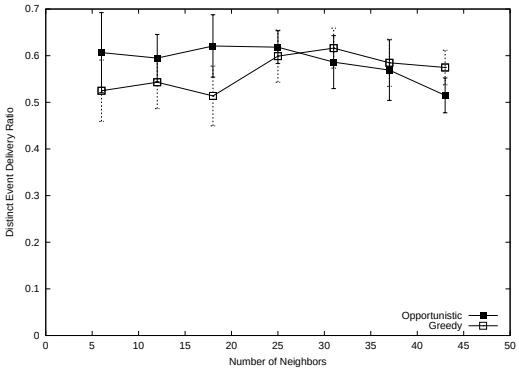
As expected, node failures adversely effect the event delivery ratio of greedy aggregation more than that of opportunistic aggregation in the low-density networks (Figure 4.5(c)). With increases in network density, the size of the opportunistic tree increases due to less path sharing whereas the size of the greedy tree approximately remains the same. It is likely that there are more node failures on the opportunistic tree than the greedy tree. Therefore, the event delivery ratio of greedy aggregation is higher than that of opportunistic aggregation for high-density networks. This lower event delivery ratio of opportunistic aggregation also results in its higher dissipated energy per event received (Figure 4.5(a)).



(a) Average dissipated energy



(b) Average delay



(c) Distinct-event delivery ratio

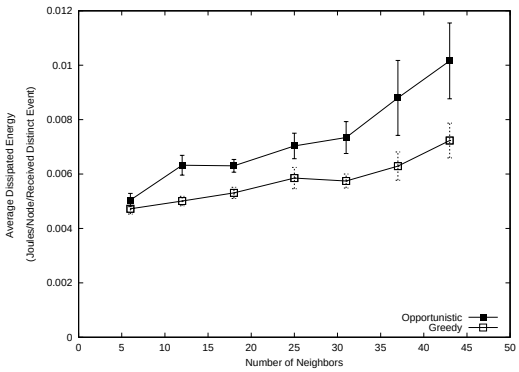
Figure 4.5: Impact of node failures.

4.3.4 Sensitivity Analysis

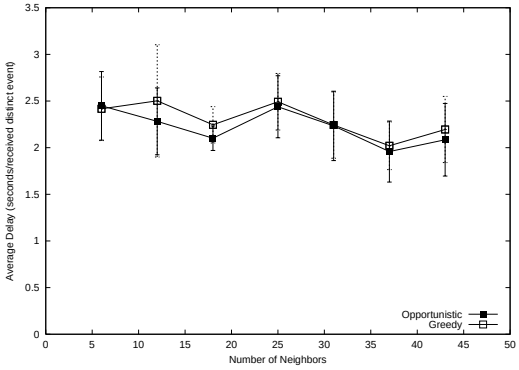
We also study the sensitivity of our greedy aggregation to several factors: number of sources, number of sinks, source placement schemes, and aggregation functions. These factors can adversely impact the energy savings of our approach, particularly when sources are near to one another and data is not perfectly aggregatable.

In our previous comparisons, we placed 5 sources in the bottom left corner of the sensor field. How sensitive is our comparison to this source placement scheme ? In this experiment, we randomly placed 5 sources in the sensor field and re-ran the simulations of Section 4.3.2. Our results in Figure 4.6(a) indicate that the energy savings of greedy aggregation are reduced to 30%. In these scenarios, sources are not necessarily closer to one another than to a sink. Even using the greedy incremental tree, *aggregation points* (*i.e.,* branching nodes) can be very far from sources.

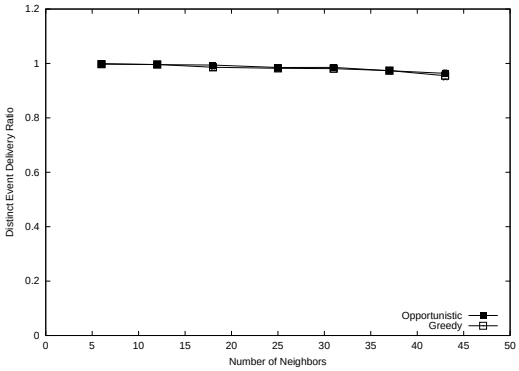
We also conducted an experiment to evaluate the sensitivity of our comparisons to the number of sinks (from 1 to 5 sinks) in the sensor field of 350 nodes. Similar to experiments in Section 4.3.2, 5 sources are randomly selected from nodes in a 80m by 80m square at the bottom left corner of the sensor field. The first sink is placed in a 36m by 36m square at the top right corner whereas the other sinks are uniformly scattered across the entire sensor field (a 200m by 200m square). With more sinks, the energy efficiency of greedy aggregation is comparable to that of opportunistic aggregation (Figure 4.7(a)). This impact of the random sink placement is similar to that of the random source placement. However, the event delivery ratio of greedy aggregation is higher than that of opportunistic aggregation because early aggregation of greedy aggregation reduces the overall traffic in general (Figure 4.7(c)).



(a) Average dissipated energy

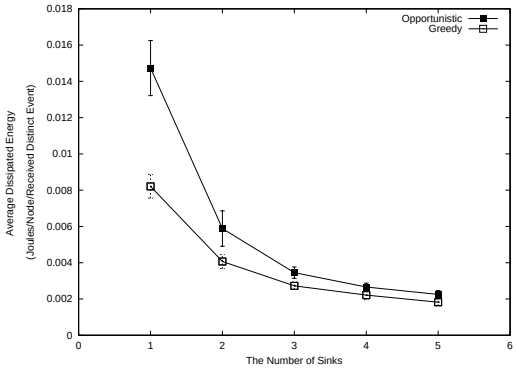


(b) Average delay

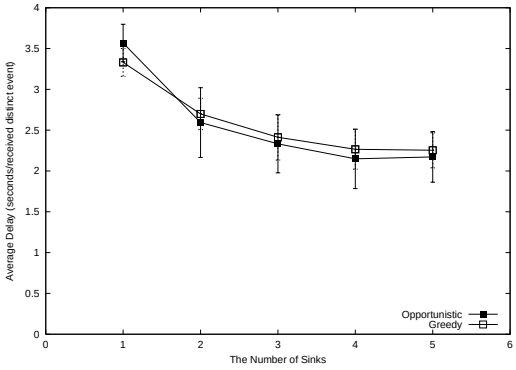


(c) Distinct-event delivery ratio

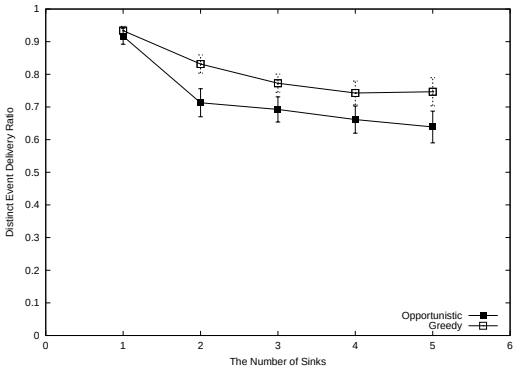
Figure 4.6: Impact of the random source placement.



(a) Average dissipated energy



(b) Average delay

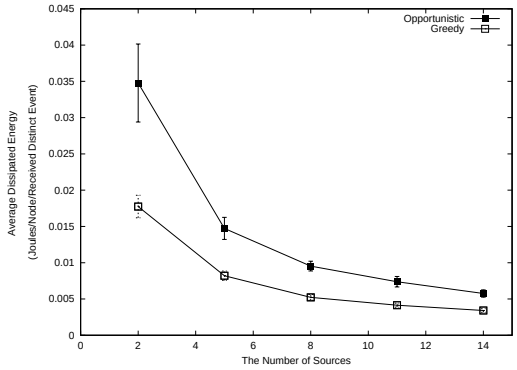


(c) Distinct-event delivery ratio

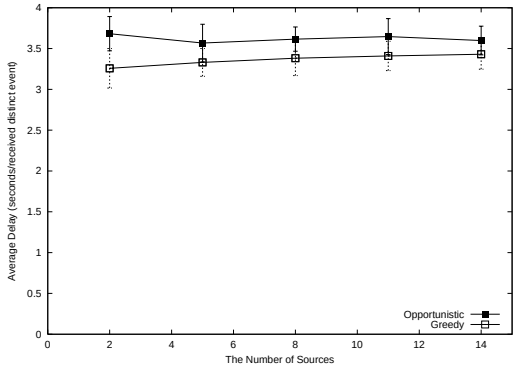
Figure 4.7: Impact of the number of sinks.

To study the sensitivity of our comparisons to the number of sources, we re-ran our simulations in the sensor field of 350 nodes with 2, 5, 8, 11, and 14 sources. As the number of sources increases, the energy efficiency of greedy aggregation converges to that of the opportunistic aggregation (Figure 4.8(a)). Given the high number of sources in the fixed area, our source placement scheme is approaching the event-radius model whereby sources are very close to one another. In these scenarios, paths from several sources are likely to be merged early even without path optimization.

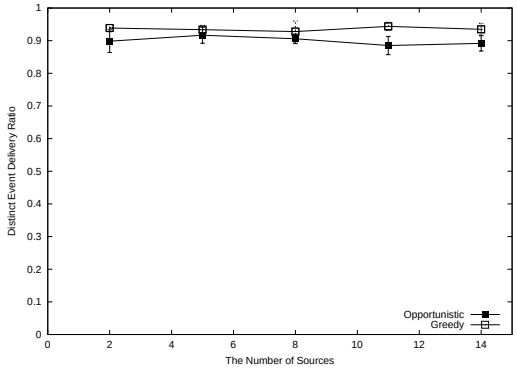
So far, we assumed perfect aggregation for all our previous experiments. Of particular interest is the impact of other aggregation functions on the energy savings of greedy aggregation. In this experiment, we used linear aggregation for such sensitivity study. Given linear aggregation, the size of aggregate (denoted by $z(S_i)$) is the linear function of data items in the aggregate S_i. Specifically, $z(S_i) = d_i * |x| + h$ where d_i is the number of data items in the aggregate, each data item size $|x|$ is 28 bytes, and the header size h is 36 bytes for our experiment. Linear aggregation is information lossless but does not save as much as lossy aggregation because the only savings are the packet headers. Given that this aggregation function depends on the number of data items, we re-ran the previous experiment with linear aggregation. As expected, the adverse impact of the inefficient aggregation function becomes more evident with the increased number of sources or data items (Figure 4.9(a)). In one experiment (11 sources), we found that greedy aggregation can achieve 36% energy savings using the linear aggregation function whereas it can achieve 43% energy savings using the perfect aggregation function.



(a) Average dissipated energy

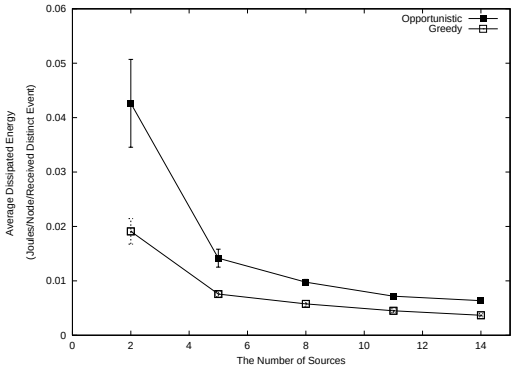


(b) Average delay

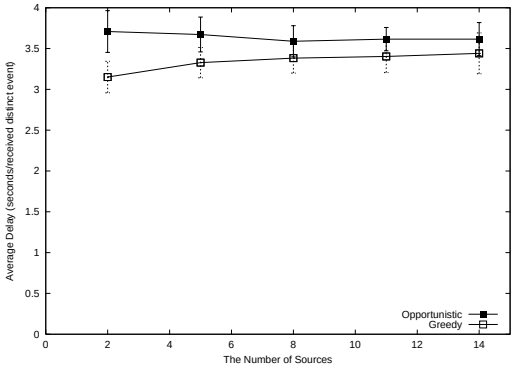


(c) Distinct-event delivery ratio

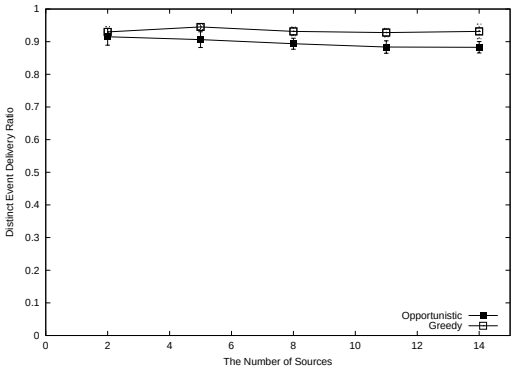
Figure 4.8: Impact of the number of sources.



(a) Average dissipated energy



(b) Average delay



(c) Distinct-event delivery ratio

Figure 4.9: Impact of linear aggregation.

Chapter 5

Related Work

Our work has been informed and influenced by a variety of other research efforts, which are summarized in this section.

5.1 Networked Embedded Systems

Distributed sensor networks are specific instances of ubiquitous computing, envisioned by Weiser [Wei91] as the third wave in computing next to mainframe and PC computing era. In ubiquitous computing, various computing elements are so seamlessly integrated into the environment that they will be invisible to common awareness.

To step toward ubiquitous computing, several projects have demonstrated the feasibility of networked embedded systems and cooperative sensor applications, but did not approach the issues of scalable node coordination. Their focus was on issues of designing and building small wireless devices. Wireless Integrated Network Sensors (WINS) [PK00, Kai99, pro98] has focused on hardware technologies and

signal processing techniques in the context of a military situational awareness problem. The WINS project has made significant progress in identifying feasible radio designs for low-power environmental sensing. Their work has been instrumental in clarifying the trade-off between computation and communication and the need for in-network processing. Our focus on in-network processing is motivated by their work. Their project has also focused on low-level network synchronization necessary for network self-assembly. Our directed diffusion primitives can provide inter-node communication once network self-assembly is complete.

Other research groups have also focused on designing and building similar networked sensors that are much smaller (*e.g.,* motes [HSW+00], smart dust [KKP99]). Motes are a square inch in size whereas smart dust is only a few cubic millimeters in volume. This smart dust can be integrated into the physical environment and programmed using amorphous computing [AAC+00].

The SCADDS group developed a tiered system architecture with both larger and smaller nodes [CEE+01]. Less resource-constrained nodes form the highest tier and act as gateways to the second tier. The second tier is composed of motes connected to low-power sensors running micro-diffusion. Most of the network "intelligence" is programmed into the first tier. Second-tier nodes are controlled and programmed from these more capable nodes. The SCADDS work also includes algorithms for low-power time synchronization [EE01b, EGE02] and localization [BHE00, GE01] that are necessary for coordinated monitoring. In addition, the SCADDS group has proposed several adaptive-fidelity techniques for self-configuring a multi-hop wireless sensor network [XHE01, CE02]. These techniques exploit the redundancy provided by highly-dense sensor networks to prolong the system lifetime. They focus on

turning the radio off as much as possible while maintaining application fidelity. The group has also designed an energy-efficient MAC for wireless sensor networks using a similar technique [YHE02]. The SCADDS work complements diffusion, especially by reducing idle-energy consumption.

Piconet [BCE+97] is a low-rate (about 40 Kb/s), low-range (5m) ad hoc radio network. Their work has presented fundamental advances in energy-conserving network communications for networks of devices. Its applications include home and office information discovery systems. Their work focuses on static hierarchies of networked devices, concentrators, and hosts. While similar to the SCADDS tiered architecture with full and micro-diffusion, they do not consider attribute-named data or *dynamic* in-network processing. Active badge [WHFG92, WH92] and the ORL location system [HH94, WJH97] focus on location tracking applications. Stajano and Jones [SJ98] present a system whereby users can query and control their basic personalized computing resources and services from their cell-phones (*e.g.,* checking emails or stock quotes). Due to the requirement for base stations, their work is centralized, but diffusion is completely distributed.

Jini is an example of a resource discovery system built over Internet protocols [Wal99]. It provides a directory service and uses Java to distribute processing to user nodes, making it well suited to a local-area network with high bandwidth and multicast. By contrast, we distribute the directory across the network and allow application-specific processing at intermediate system nodes, addressing problems of resource-constrained, multi-hop wireless networks. The Ninja Service Discovery Service [CZH+99] locates XML-named objects through a network of collaborating servers but again targets high bandwidth local-area resources.

In addition, recent work has demonstrated some of the advantages of diffusion-like application-specificity in the context of sensor networks [KRB99]. Specifically, SPIN embeds application semantics in flooding to achieve energy efficiency. Directed diffusion applies application-specificity in the context of more sophisticated distributed sensing algorithms under a modified wireless contention MAC on *ns-2*. Data in SPIN is identified by application-specific metadata that appears to assume individual sensors are addressable. We instead use attributes to name data alone; globally unique identifiers are not necessary. Additionally, for SPIN to be useful, the actual data must be significantly larger than the meta data whereas diffusion is energy efficient regardless of the data size. Moreover, SPIN does not consider application-specific in-network processing.

LEACH analyzes the performance of cluster-based routing mechanism with in-network data compression [HCB00]. They emphasize how intermediate-range communication via cluster-heads and how compression can reduce energy consumption. Their in-network compression is one example of the kind of in-network processing that we would like to support. LEACH can achieve energy savings by processing application-level data at its cluster heads whereas diffusion can process such data anywhere in the network. Additionally, they do not specify how flows and opportunities for aggregation would be activated whereas our work focuses on the naming mechanisms that allow such activity.

Declarative Routing from MIT's Lincoln Labs is closest to our work [CHMK00]. The publish/subscribe-oriented API [CHG$^+$01] we use was defined in collaboration with them. They have also developed an independent implementation of the API. Their work and ours name data rather than end-nodes. Differences are in how

routes and transmission are optimized, both by applications and the core system. The primary difference is that declarative routing does not include mechanisms to enable in-network data processing.

Of particular interest are IDSQ (Information-Driven Sensor Querying) and CADR (Constrained Anisotropic Diffusion Routing) [CHZ02, ZSR02]. They use information gain and cost function for data routing and for selecting which sensors to query. Unlike diffusion, their work assumes that each sensor node has a globally unique identifier and each querying node knows sensor characteristics (*e.g.,* a sensor position) of every node in the network. Such global knowledge is expensive in general. Similar to LEACH, they do not specify mechanisms to activate flows and opportunities for in-network processing whereas our work focuses on the naming mechanisms that allow such activity. Moreover, given our flexible framework, one could design a diffusion instantiation which also uses an information utility measure for guiding data dissemination.

Furthermore, a recent routing protocol for sensor networks approached the multipath delivery issues by using multiple overlapping spanning trees [SGAP00]. Their work is however based on topographically-addressed sensor nodes. Diffusion also explores some design choices of multi-path delivery but the primary difference in our work is the use of attribute-based naming for structure and data diffusion for communication. Ganesan et al. [GGSE01] have proposed a braided multipath scheme for energy-efficient recovery from isolated and patterned failures. Based on our diffusion framework, their work constructs multiple paths between two nodes using localized algorithms. Recently, some work has also focused on mobile-user support

for directed diffusion [PCMN02]. The Smart Messages (SM) project developed a cooperative computing model for distributed embedded systems [BIK+02]. Their work borrows heavily from active networks, active messages, and mobile agents, particularly their implementation solutions. The project has demonstrated the model flexibility by implementing diffusion using the SM architecture.

Recent work has identified data-centric storage [REG+02] and SCOUT [KAE00] as companion methods of directed diffusion. In these companion methods, data is stored at a sensor node determined by the data name (regardless of the data originator). Subsequent queries for that data can then be sent directly to that particular node. However, the data-centric storage requires strongly geographic routing whereas diffusion do not require any routing mechanism.

The COUGAR device database system proposes distributing database queries across a sensor network as opposed to moving all data to a central site [BGMS99]. Sensor data is represented as an Abstract Data Type attribute, the public interface to which corresponds to specific signal processing functions supported by a sensor type. They then perform joins or aggregation in the network as specified by a centrally computed query plan. Their work is common with ours in its emphasis on in-network processing. The primary difference between their work and ours is how placement of in-network processing is determined. We emphasize the use of filters to enable either ad-hoc or sensor-specific placement of in-network processing whereas COUGAR centrally translates the query and assigns processing to the distributed system, incurring overhead to centrally collect network information for query optimization.

DataSpace describes an attribute based naming mechanism for querying physical objects that produce and store local data [IG00]. The DataSpace is divided into smaller administrative and logical *datacubes*, which are logically grouped into *dataflocks*. Dataflocks are addressed at the network level through IPv6 multicast addresses that correspond to their geographic coordinates, and their values for certain attributes that serve as network indices. Query results may involve aggregation of more specific queries addressed to *sub-datacubes*. At a high-level their naming approach is similar to ours, but instead of mapping attributes and geometry to a very large number of multicast groups we route directly on attributes themselves without this indirection. In addition, they do not explore in-network processing.

5.2 Biological Systems

Biological systems can offer useful insights and inspirations in the design of directed diffusion. Of particular interest are the reaction-diffusion models for morphogenesis and the models based on behavior of ant colonies.

In 1952, Turing proposed the reaction-diffusion model [Tur52] to provide a scientific explanation for the patterns of pigmentation in animals. The model demonstrates how local interactions of two chemicals, Morphogens (*i.e.,* activator and inhibitor), can model nature's global behaviors (*i.e.,* pigmentation patterns). Directed diffusion is analogous to this work in the sense that local interactions of sensors lead to global objectives (*e.g.,* energy-efficient paths between sources and sinks).

Extensive models of ant colony metaphors also inspire directed diffusion. These models are based on the fact that the almost blind ants seem to be able to find

shortest paths to destinations using only the pheromone trails deposited by other ants. The models include AntNet [CD97], an adaptive routing algorithm, whereby each node periodically launches a forward ant to build a path to a random selected destination. Each ant collects information about the delay of the path components and the load status of the network. After the forward ant reaches the destination, a backward ant is launched to back-propagate this information to modify the routing tables of visited nodes.

Schoonderwoerd et al. [SHBR96] proposed an ant-based control for load balancing in telecommunications networks whereby the routing tables are replaced by pheromone tables (*i.e.,* tables of probabilities). Based on Schoonderwoerd's work, Devika et al. [SDC97] investigated ant routing algorithms for dynamic networks.

Ant System [DMC96], a general-purpose heuristic algorithm, was applied to optimization problems, including the traveling salesman, the quadratic assignment, and the job-shop scheduling.

However, in the bi-directional trail laying principle of ant colony that is used to find the shortest path, there are two problems: the blocking problem and the shortcut problem [Sut90]. The blocking problem occurs when an established route fails. Ants may spend a relatively long time finding a new route due to the strong pheromone of the previous route. The shortcut problem occurs when a new, shorter route suddenly exists. The new route will not easily be found, because the pheromone of the old route is so strong that all the ants continue going to the old route.

Although inspired by the principle, directed diffusion does not suffer from those two classical problems because the model does not completely rely on the route history. Multiple exploratory paths are kept active and periodically observed for

adaptation to network dynamics. Directed diffusion is also different in many ways. Earlier work has not focused on energy-efficient wireless communication. Nor do they attempt to leverage application-specificity, caching-capability, and aggregation-functionality.

5.3 Ad hoc Networks

Directed diffusion borrows heavily from ad hoc networks, especially their unicast routing protocols. Ad hoc networks are autonomous networks of mobile wireless hosts that can communicate with one another in the absence of a fixed infrastructure. Most proposed ad hoc routing protocols can be classified as proactive or reactive. Proactive routing protocols (*e.g.,* DSDV [PB94]) continuously compute routes to all nodes so that a route is already available when needed. On the other hand, reactive or on-demand routing protocols (*e.g.,* DSR [JM96], AODV [Per97], TORA [PC97]) compute only the needed route on the fly. Directed diffusion is similar to on-demand routing protocols in the sense that gradients are established only when needed. It is possibly closest to TORA in its attempt to localize repair of node failures and its de-emphasis of optimal routes.

In a sense, ad hoc networks using on-demand routing protocols can be considered self-organizing networks. The network structure is organized in response to a need to communicate, thus, easily adapting to the traffic pattern, node mobility, node failure, and network congestion.

The continuous route computation of proactive protocols may not be bandwidth-efficient, but the route recovery via a flooding technique used by several reactive

protocols may not either. The hybrid schemes (*e.g.,* ZRP [HP98], CBRP [JLT98]) may overcome these disadvantages.

Many of the techniques developed for improving ad hoc routing performance can be directly applied to directed diffusion. These include techniques that reduce the impact of the broadcast storm problem [NTCS99] (*i.e.,* the significant redundancy, contention, and collision, caused by flooding). LAR [KV98] localizes queries to a limited geographic region by using the Global Positioning System (GPS). Castaneda et al. [CD99] proposed query localization techniques based on prior routing histories. Their diffusion analogue is GEAR [YGE01].

Internet ad hoc routing can also be used in sensor networks. However, since ad hoc routing recreates IP-style addressing, it would require some kind of directory service to locate sensors, unlike our approach where they are named by attributes. In addition, ad hoc routing does not support in-network processing and so would not benefit from energy savings due to aggregation.

5.4 Active Networks

Recent work in active networks [TSS+97] and active services [AMK98] has examined ways to provide in-network processing for the Internet. Sample applications include information transcoding, network monitoring, and caching. This work is built over an Internet-like infrastructure, often augmented with an extended run-time environment, and assumes nodes are individually addressable. We instead build directly over hop-by-hop communications primitives and identify data instead of nodes. Our

work differs from active services in that we assume that communications costs between nodes vary greatly while currently proposed active services assume roughly equivalent distances between all service-providing nodes. We differ from active networks primarily in the target domain: we target sensor networks where bandwidth is limited, energy is expensive, and compute power is comparatively plentiful and inexpensive. Instead, active networks typically considers Internet-like domains where bandwidth is plentiful, the ratio of compute power to bandwidth is much lower, and energy is not an issue. All of these approaches distribute application-specific code throughout the network, raising questions about code safety and portability. These problems are not central to some sensor networks (such as those that are devoted to a single application), but more complex networks would benefit from active-networks-style execution environments to support in-place upgradability.

An application of active services is Gathercast [BS00]. The goal of Gathercast is to losslessly aggregate small packets (with the same destination) into a larger packet for reducing the overhead of routing table lookups (not for reducing the total data size). Their work opportunistically aggregates small packets without altering any routing mechanisms or paths. Diffusion incorporates data-centric routing and application-specific processing inside the network. Our work can optimize the aggregation tree for more energy savings.

Recent work on peer-to-peer file sharing systems such as Freenet [CSWH00] explores application-specific, hop-by-hop processing. Unlike active networks and our work, these approaches emphasize protocols designed for a particular application. In addition, our work runs directly over hop-by-hop communication rather than over a virtual network layered over the Internet.

5.5 Multicast Routing

In sensor networks, it is typical that multiple users may be interested in the same information. Directed diffusion is also influenced by the design of multicast routing protocols. One ultimate goal of a multicast protocol and directed diffusion is to distribute data to a subset of nodes by using the minimum number of edges. However, the computation of such tree (*i.e.,* a Steiner tree [GP68, Win87]) is an NP-complete problem [Kar72] even if all edge weights are equal. Hence, no multicast protocol uses the Steiner tree as its distribution tree.

Current multicast protocols on Internet can be classified into three approaches based on their distribution tree (*i.e.,* a shortest-path tree, a shared tree, and a greedy tree). Multicast protocols based on a shortest-path tree include DVMRP [Dee88], MOSPF [Moy94], and PIM-DM [DEF+97]. Even though a shortest-path tree may not be optimal, it is resilient to change. The inefficiency of the tree is independent of the order of group membership dynamics [DL93]. Such protocols inspire directed diffusion to some extent. The initial interest dissemination and gradient establishment of directed diffusion is similar to data-driven shortest-path tree construction of DVMRP and PIM-DM. The difference is that those protocols rely on underlying unicast routing to aid tree construction, but directed diffusion can not. Directed diffusion can, however, do in-network processing of data and queries to achieve network scalability and energy efficiency, which existing multicast routing schemes can not. Moreover, unlike MOSPF, directed diffusion does not need global topology information for path establishment.

Multicast protocols based on a shared tree include CBT [BFC93] and PIM-SM [EFH+97]. The network efficiency of these protocols depends on the position of the core. However, finding the optimal position of such node is also NP-complete [Kar72]. Positive and negative reinforcements of directed diffusion are similar to joins and prunes in shared-tree construction of CBT and PIM-SM. The difference is that there may be application semantics associated with reinforcements. In addition, the gradient map is rather a shared mesh than a shared tree due to the multi-path delivery.

A greedy tree [TM80] is probably the most energy efficient tree used by existing multicast protocols. The greedy tree has been shown to outperform the shortest path tree in various studies [Wax88, Wax93]. An efficient data distribution of such tree can be achieved if the membership of the multicast group is not too dynamic (*i.e.,* for some patterns of joins and prunes, the resulting multicast tree can be very inefficient). QoSMIC [FBP98] constructs its multicast tree using the local search and offers alternate paths to enable the support of QoS requirements. Our greedy approach (an instantiation of directed diffusion) differently constructs the greedy aggregation tree using a greedy heuristic for an NP-hard weighted set-covering problem.

5.6 Reliable Multicast

Several reliable multicast schemes have greatly inspired directed diffusion, especially their Application Level Framing model (ALF) [CT90, RM98, FDTH98] and multicast loss recovery. ALF explicitly includes an application's semantics in the design of

the application's protocol. It was later combined with Light-Weight Sessions (LWS) [Jac93] to create a framework for Scalable Reliable Multicast (SRM) [FJL+97]. Particularly, ALF violates the traditional layered architecture approach by integrating the end-to-end transmission control layers (*i.e.,* transport layter to application layer in the OSI model) for flexibility. As a result, ALF allows the application to be network conscious and to adapt its behavior to the network dynamics. The idea is further developed in directed diffusion by integrating one more layer (*i.e.,* network layer to application layer in the OSI model) than those integrated by ALF. Diffusion can profit by routing data directly in application-level terms and can reduce overhead caused by multiple layers of naming.

Directed diffusion is also motivated by multicast loss recovery in solving two major problems of reliable multicast (*i.e.,* implosion and exposure). Implosion occurs when the loss of a packet triggers simultaneous requests or replies from a large number of receivers. Exposure occurs when recovery-related messages reach receivers, which have not experienced loss. Directed diffusion also needs to limit implosion of reinforcement.

Of particular interest are hierarchical or tree-based approaches which include RMTP [LP96] and LMS [PPV98] (*i.e.,* systems where the members are organized into a recovery hierarchy). RMTP avoids implosion via ACK fusion whereby ACKs are sent to the parent member (not to the source) who merges the ACKs before sending them to its own parent. Designated receivers in the hierarchy can perform local recovery to limit exposure by sending retransmissions from their data caches. Therefore, local recovery is successfully provided if the recovery hierarchy of members is closely correlated to the multicast distribution tree.

Unlike RMTP, LMS uses the multicast tree itself rather than a separate hierarchy. A retransmission request sent to the turning-point router from a non-replier link is forwarded to the replier link whereas a request from a replier link is forwarded to the parent. Thus, exactly one request can escape the loss subtree. The recipient of the escaping request responds with a directed multicast retransmission from its data caches. Due to a nature of the explicit hierarchy, RMTP and LMS suffer from concentration of responsibility.

Search Party [CM99], a randomized version of LMS, improves robustness at the expense of added latency and overhead by random routing of requests. Like LMS, Search Party is well applicable on per source trees but questionable for shared trees.

Local recovery can significantly reduce network traffic. A nearby node with the requested data on its cache can retransmit it without informing the source. Directed diffusion is similar to reliable multicast in a sense that it also takes advantage of locally cached data. The in-network-processing feature of directed diffusion is also similar to router assist for local recovery in LMS and Search Party (They require minimal router functionality for specialized forwarding certain kinds of data). The idea is further developed in directed diffusion by embedding application knowledge in network nodes.

5.7 Internet Web Caching

Directed diffusion is also inspired by Internet web caching which can efficiently reduce the receivers' latency and save network bandwidth (A nearby node with the requested web page on its cache can directly reply the web request, and improve responsiveness

of web accesses). Generally, web caches coordinate to improve the performance by using a caching hierarchy [CDN+96]. Even though hierarchical caching unquestionably helps scaling the World Wide Web better, higher-level caches tend to become bottlenecks. An alternative to avoid such problem is distributed caching. Given that some load on origin web servers is moved to web caches, load-balancing techniques may be useful. Heddaya et al. [HM97] proposed WebWave, a load-balanced distributed caching system based on Cybenko's dynamic load balancing model [Cyb89].

However, Internet web caching architectures require manual configuration, considered impractical in the context of sensor networks. Unlike Internet web caching, directed diffusion is designed with data centricity in mind. Node-independent data naming in diffusion reduces the need for manual configuration and improves robustness. Locally cached data are still accessible even if the sources are unreachable.

Given that different users surf the web at different times, web caching can also be considered a method for asynchronous multicast delivery. Zhang et al. [MNR+98] proposed adaptive web caching whereby cache servers self-organize themselves into overlapping multicast groups. The mesh of overlapping groups forms an implicit hierarchy to diffuse popular web pages toward the demand. Directed diffusion is similar to adaptive web caching in a sense that queried information in sensor networks also needs to be efficiently *diffused* to the interested users. Our work differs by also providing other types of in-network processing (in addition to caching) and by operating directly over a MAC (instead of the Internet infrastructure).

5.8 Attribute-based naming systems

There has been a large amount of work on attribute-based naming, both for general purpose use over Internet-style networks, for special domains (such as the web), and as an internal structuring mechanism for services.

Research and industry have developed numerous attribute-based naming systems layered on top of general-purpose networks. Univers and yellow-pages naming at the University of Arizona [BDP93, Pet87] were designed to provide service discovery for groups of computers (for example, print to an unloaded postscript-capable printer). Like our work, they include attributes and operators, but they build over standard Internet protocols for communications. Commercial attribute-based naming systems such as X.500 [CCI88] and LDAP [YHK95] also operate over Internet or Internet-like routing and provide a primarily hierarchical organization (although some variants relax this [Neu89]). Dependence on IP-level addressing and routing adds substantial overhead when applying these systems to highly resource-constrained environments such as sensor networks. (For example, some approaches to service location for smart spaces require services for IP assignment, IP-level routing, host name lookup, and service registration and lookup.) With end-to-end processing only, these systems also do not provide in-network processing.

As an alternative to providing attribute-based naming for end-user use, several systems have proposed attribute-based communications for structuring distributed systems. Linda proposed structuring distributed programs using several CPUs around an attribute-indexed common memory called a tuple space [CG85]. For the S/Net implementation this was the basic communication mechanism, but

proposed implementations assume uniform and rapid communications between all processors. Later systems such as ISIS [Bir93] and the Information Bus [OPSS93] provide a "publish and subscribe" approach where information providers publish information and clients subscribe to attribute-specified subsets of that information. These systems are designed to be robust to failure, but again assume reasonably fast, plentiful, and expensive communications between nodes. These approaches are not directly applicable to resource-constrained sensor networks. They do not use application-specific, in-network processing since all processes are reasonably close to each other; when they do use processing (such as at a wide-area gateway) it is manually configured.

More specific still is work that proposes attribute-based primitives as solutions to specific problems. SRM first suggested using named data as the fundamental data unit for reliable multicast communication, and it demonstrated this approach with a distributed whiteboard [FJ95]. Our work is inspired by these approaches, but it differs by providing a wider range of matching operators (rather than just equality), adding in-network processing to leverage CPU-communications trade-offs for sensor networks, and operating directly over low-level (hop-by-hop) communications protocols instead of the Internet multicast infrastructure.

The Intentional Naming System is an attribute-based name system operating in an overlay network over the Internet [AWSBL99]. Its use of attributes as a structuring mechanism and a method to cope with dynamically locating devices is similar to our approach in motivation and mechanism. The primary difference is that we assume that attribute-based communication (data diffusion) is the basic communications primitive (above hop-by-hop messaging), while they construct an overlay

network over an IP-based Internet. Architecturally this implies that we distribute name matching across many small communications nodes while they manage names at a few resolvers that cooperatively manage parts of the namespace. Finally, the details of matching are different in the two systems. Their work provides a sophisticated hierarchical attribute matching procedure. Our approach is much more modest by comparison (targeting smaller embedded devices) but adds comparative operators in addition to equality.

5.9 Distributed Robotics

Diffusion's distributed sensing applications are similar to distributed robotics' applications that multiple distributed entities coordinate to achieve a particular goal. However, communication models of distributed robotics are usually very simple. Some work even assumes no communication at all [Ark92]. Furthermore, the number of cooperative elements in these systems is relatively small compared to that of sensor applications.

Of particular interest are coordination protocols designed for troops of low-cost robots to explore and to acquire maps of unknown environment [LSdMS97]. Robots move almost randomly and coordinate with one another by transferring the collected information when they meet. After completion of the exploratory run, the robots deliver their partial maps to a host computer for deriving a complete map of the area. The derived map is analogous to the sensing fidelity improved by coordination of multiple sensors.

Chapter 6

Conclusions and Future Work

Advances in radio, sensor, and VLSI technology will enable small and inexpensive sensor nodes capable of wireless communication and significant computation. Large-scale networks of such sensors may require novel data dissemination paradigms which are scalable, robust, and energy-efficient [EGHK99]. In this dissertation, we design and evaluate directed diffusion, one such paradigm for distributed sensing applications in wireless sensor networks. Directed diffusion incorporates data-centric routing and application-specific processing inside the network (*e.g.,* data aggregation). Given that, in wireless sensor networks, the communication cost is several orders of magnitude higher than the computation cost [PK00], directed diffusion can achieve significant energy savings with in-network data aggregation. This benefit of data aggregation has been confirmed analytically and experimentally (Chapter 3).

There are also several lessons we can draw from our performance evaluation of diffusion. First, directed diffusion has the potential for significant energy efficiency. Even with relatively unoptimized path selection, it outperforms an idealized traditional data dissemination scheme like omniscient multicast. Second, diffusion

mechanisms are stable under the ranges of network dynamics considered in this dissertation. However, for directed diffusion to achieve its full potential, careful attention has to be paid to the design of sensor radio MAC layers.

6.1 Summary of Contributions

Our contribution was in the design and evaluation of directed diffusion. Specifically, we contributed two instantiations of directed diffusion with different aggregation schemes: opportunistic aggregation and greedy aggregation. We also analyzed and evaluated diffusion both in simulation and in testbed experimentation.

We designed diffusion with several novel features: application-specific processing inside the network, data-centric routing, reinforcement-based adaptation, and attribute-based naming. By using attributes with external meaning (such as sensor type and geographic location) at the lowest levels of communication, diffusion avoided multiple levels of name binding common to other approaches. Attribute-based naming in turn enabled in-network processing with filters, supporting data aggregation, nested queries, and similar techniques that were critical to reducing network traffic and conserving energy. These features were important in the emerging domain of wireless sensor networks where network and power resource constraints were fundamental.

We evaluated the effectiveness of these techniques by quantifying the benefits of in-network processing analytically and experimentally. Specifically, we conducted an analytic evaluation of the data delivery cost for directed diffusion. Our results suggested that, as the number of sources and sinks increased, the cost savings due

to in-network processing of diffusion became more evident. To verify and complement our analytic evaluation, we implemented diffusion in the *ns-2* simulator. We compared the performance of diffusion against idealized schemes and studied the sensitivity of directed diffusion performance to the choice of parameters. This packet-level simulation was also used for exploring the impact of network dynamics and the influence of the radio MAC layer on diffusion performance. Additionally, we validated our results with an actual implementation of our tracking application. In particular, we examined in-network aggregation in our ISI testbed of 14 PC/104 sensor nodes.

Our evaluation indicated that diffusion achieved significant energy savings and outperformed idealized schemes even with relatively unoptimized path selection (or opportunistic aggregation). We proposed another instantiation of directed diffusion that constructed a greedy incremental tree to improve path sharing for more energy savings. We evaluated the performance of this greedy approach by comparing it to the opportunistic approach. Our result suggested that although greedy aggregation and opportunistic aggregation were roughly equivalent in low-density networks, greedy aggregation achieved significant energy savings in high-density networks without adversely impacting latency or robustness. Given that the energy was the scarce resource, this path optimization technique was essential for prolonging the lifetime of the highly-dense sensor networks.

Other contributions in this dissertation included evaluation metrics, tradeoffs, simulation platforms, test suites, applications, requirements, challenges, and insights into the design of data dissemination systems for wireless sensor networks. Specifically, our evaluation metrics were average dissipated energy, average delay,

and distinct event delivery ratio. These metrics indicated the overall lifetime of sensor nodes, the temporal accuracy of the estimates, and the robustness of the system. The challenge was to design a system that was long-lived but still accurate and robust. Given that sensor networks were task-specific, we described our design using the animal tracking application as an example. We implemented directed diffusion on the *ns-2* network simulator. The code was downloadable from `http://www.isi.edu/nsnam/ns`.

6.2 Future Work

The techniques presented in this dissertation are only initial attempts and particular instantiations of directed diffusion for remote surveillance sensor networks. There are still several other kinds of sensor networks in need of similar mechanisms. However, the design space of diffusion is so large and non-trivial that, even for the specific sensor network in this dissertation, we have not completely explored the entire design space yet. In this section, we discuss some of these alternative designs and future directions

To avoid re-implementing diffusion mechanisms for every new application, a diffusion substrate provides a library of diffusion mechanisms (*e.g.,* interest propagation, gradient establishment, reinforcement) for sensor network designers. So far, we have designed two sets of such diffusion mechanisms: opportunistic aggregation for low-density networks and greedy aggregation for high-density networks. However, these two sets are not sufficient to support different sensor networks. For example, a geographic interest-propagation mechanism is a better mechanism for a network

in which each sensor node is equipped with a GPS receiver. Therefore, a future direction is to design and to support more alternative mechanisms (and new services). Possible alternative and new services include:

- *Reinforcement.* Given our aggressive reinforcement rule and conservative negative reinforcement rule described in Section 2.4, there tend to be several data paths after several rounds of exploratory events. An alternative rule for path establishment is to reinforce the neighbor that has consistently sent events before others or has sent the most events. These local rules trade off reactivity for increased stability. A more aggressive rule for path pruning is to negatively reinforce the neighbors that have sent relatively few distinct events. However, extensive simulations are required to fully understand the tradeoffs.

- *Urgent Event Propagation.* In our example sensor network, some users have to be interested in data first before paths can be established to draw data. However, directed diffusion is not limited to only such usage. In some sensor networks, important events may be spontaneously propagated without prior interests to warn other sensor nodes of impending activity.

- *Multiple-Constraint Data Dissemination.* So far, we have designed two diffusion instantiations, each of which establishes data gradients using only a certain metric (the latency for the opportunistic approach and the energy cost for the greedy approach). However, one could design novel instantiations of diffusion based on other metrics, such as remaining node energy and data quality. For example, a node may include its remaining energy in an exploratory event before sending the event to neighbors. A plausible reinforcement rule is

to favor neighbors with high remaining energy over those with low remaining energy. Such a rule reduces the probability of network partitions and prolongs the overall network lifetime.

Moreover, an objective function that is a composite of several metrics could also be used. To establish data gradients, a node measures relevant metrics of each neighbor and reinforces the neighbor with the highest or the lowest value of the composite function. In a sense, our greedy approach is an initial step in this direction. The greedy approach maintains the paths with the lowest energy cost per data item. However, each data item might not provide the equal gain of data quality. A better rule might be to maintain the paths with the lowest energy cost per data-quality unit. However, before such rule can be designed, it is necessary to understand better about application-perceived quality.

- *Quality of Service.* In a sense, such multiple-constraint functions can be used to select a path based on its quality (*e.g.,* latency, remaining energy). A plausible next step after the design of the multiple-constraint data dissemination is to provide quality of service according to the packet importance (priority). For example, a lowest-latency path might consist of nodes with low remaining energy. To avoid the energy depletion of these nodes, such path should be reserved only for high-priority packets.

- *Mobility.* Even though our experiments on network dynamics impose fairly adverse conditions for a data dissemination protocol, diffusion is relatively stable

at the levels of dynamics we have explored. However, the impact of node mobility on directed diffusion has not been investigated. Stable mechanisms to rapidly recover from path failures are important, particularly to a network of sensors which constantly move (*e.g.,* float on the water or in the air). One would expect that, at low mobility, the periodic exploratory events are sufficient to recover from failures. For high mobility, our local recovery techniques could be used. Nevertheless, significant experimentation is required before deciding if such mechanisms are sufficient. Some techniques from ad hoc networks will be useful in this context.

- *Congestion Control.* Given bandwidth limitations, a wireless sensor network can easily be overloaded (congested). As a result, a congestion control mechanism is unavoidably necessary in the sensor network. Generally, a basic principle of a congestion control technique is to send fewer packets when there is an indication of congestion. Based on this principle, one could design a plausible congestion control mechanism for the sensor network as follows.

  Each sender, by default, will transmit packets at a decreasing rate over time unless a reinforcement is received. Each reinforcement contains some extra information which might indicate congestion (*e.g.,* the percentage of packet drops, the number of empty queue slots). Upon receiving a reinforcement, the node appropriately increases or decreases the sending rate based on such information. In a sense, this reinforcement is similar to an acknowledgement used in the TCP congestion control [Jac88]. A primary difference is that,

unlike the diffusion sender, the TCP source only increases the sending rate once it receives an acknowledgment.

Furthermore, our experiences with diffusion on our testbed have suggested several areas for future work. These areas include enhancing our testbed and protocols, applying them to additional applications, and understanding how to build sensor networks.

- The SCADDS group has several planned changes to our testbed hardware. Most importantly, we plan to move to a different radio by RF Monolithics and to use a UCB Mote as the packet controller [YHE02]. The packet-level controller of our Radiometrix RPC was very helpful for rapid development, but this revised approach will give us complete control over the MAC protocol.

- We have now explored diffusion performance both in simulation and with testbed experiments. In-network aggregation shows qualitatively the same results in both evaluations (Section 3.2 and 3.3). A next step is to use the experiments to parametrize the simulations.

- In our experimentation, we were repeatedly challenged by the difficulty in understanding what was going on in a network of dozens of physically distributed nodes. Our current environment augments the radio network with a separate wired network for experimental data collection, but much more work is needed in developing analysis tools for these networks. Tools are needed to report the changing radio topology, observe collision rates and energy consumption, permit more flexible logging, and accurately synchronize node clocks. In-network monitoring tools, such as eScan [ZGE02], are needed.

- Appropriate MAC protocols for sensor networks is a continuing challenge. In spite of published work in this area [BCE+97, SGAP00] and ongoing activities, a freely available, energy aware MAC protocol remains needed [YHE02].

- A balance of control and data traffic is particularly important in bandwidth-constrained systems such as sensor networks. Several known techniques to constrain control traffic exist for soft-state protocols in wired networks [Jac90, SEFJ97, WTZ99]; these approaches need to be applied to our system.

- We have explored some applications of sensor networks, but many other applications remain. One interesting direction is to explore how collaborative signal processing interacts with in-network processing and filters [ZSR02].

- Finally, although we focus on wireless sensor networks, the techniques we develop are also relevant to wired sensor networks. Wired connections greatly reduce bandwidth constraints and eliminate power constraints, but attribute-based naming can reduce system complexity by decoupling data sources and sinks, and in-network processing may reduce latency and improve scalability. Although prior systems have separately used these abstractions for virtual information systems, a future direction is to apply them to large, wired sensor networks that are coupled with the physical world.

As we have emphasized in this section, this work only represents an initial foray into the design of diffusion mechanisms. Even though several fundamental issues of wireless sensor networks have been addressed in this dissertation, we have not explored the entire space of alternative designs and solutions. To draw a simple

analogy, we are with sensor networks where we were with the Internet about three decades ago.

Reference List

[AAC+00] Harold Abelson, Don Allen, Daniel Coore, Chris Hanson, Erik Rauch, Gerald Jay Sussman, and Ron Weiss. Amorphous Computing. *Communications of the ACM, Special Issue on Embedding the Internet*, 43(5):74–82, May 2000.

[AMK98] Elan Amir, Steven McCanne, and Randy H. Katz. An active service framework and its application to real-time multimedia transcoding. In *Proceedings of the ACM SIGCOMM*, pages 178–189, Vancouver, Canada, September 1998. ACM.

[Ark92] Ronald C. Arkin. Cooperation without communication: Multi-agent schema based robot navigation. *Journal of Robotics Systems*, 9(3):351–364, 1992.

[AWSBL99] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremy Lilley. The Design and Implementation of an Intentional Naming System. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 186–201, Charleston, SC, 1999.

[BCE+97] F. Bennett, D. Clarke, J. Evans, A. Hopper, A. Jones, and D. Leask. Piconet: Embedded Mobile Networking. *IEEE Personal Communications Magazine*, 4(5):8–15, October 1997.

[BDP93] Mic Bowman, Saumya K. Debray, and Larry L. Peterson. Reasoning about naming systems. *ACM Transactions on Programming Languages and Systems*, 15(5):795–825, November 1993.

[Bea90] J.E. Beasley. A lagrangian heuristic for set-covering problems. *Naval Research Logistics*, 37(1):151–164, 1990.

[BEF+00] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[BFC93] T. Ballardie, P. Francis, and J. Crowcroft. Core Based Trees (CBT). In *Proceedings of the ACM SIGCOMM*, pages 85–95, San Francisco, CA, September 1993.

[BGMS99] Philippe Bonnet, Johannes Gehrke, Tobias Mayr, and Praveen Seshadri. Query processing in a device database system. Technical Report TR99-1775, Cornell University, October 1999.

[BHE00] Nirupama Bulusu, John Heidemann, and Deborah Estrin. GPS-less Low-Cost Outdoor Localization for Very Small Devices. *IEEE Personal Communications Magazine, Special Issue on Smart Spaces and Environments*, 7(5):28–34, October 2000.

[BI96] Richard R. Brooks and S. Sidtharama Iyengar. Robust distributed computing and sensing algorithm. *IEEE Computer*, 29(6):53–60, June 1996.

[BIK+02] Cristian Borcea, Deepa Iyer, Porlin Kang, Akhilesh Saxena, and Liviu Iftode. Cooperative computing for distributed embedded systems. In *Proceedings of the International Conference on Distributed Computing Systems*, Vienna, Austria, July 2002. IEEE.

[Bir93] K. P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):36–53, December 1993.

[BMJ+98] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad-Hoc Network Routing Protocols. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom'98)*, pages 85–97, Dallas, TX, October 1998.

[BS00] B. Badrinath and P. Sudame. Gathercast: The design and implementation of a programmable aggregation mechanism for the internet. In *IEEE International Conference on Computer Communications and Networks (ICCCN)*, pages 206–213, Las Vegas, NV, October 2000.

[BSK96] Thomas Bäck, Martin Schütz, and Sami Khuri. A comparative study of a penalty function, a repair heuristic, and stochastic operators with the set-covering problem. In J. M. Alliot, E. Lutton, E. Ronald, M. Schoenhauer, and D. Snyers, editors, *Artificial Evolution*, pages 3–20. Springer, Berlin, 1996.

[CCI88] CCITT. The directory: Overview of concepts, models and service. Recommendation X.500, CCITT, 1988.

[CD97] Gianni Di Caro and Marco Dorigo. AntNet: A Mobile Agents Approach to Adaptive Routing. Technical Report 97-12, IRIDIA, Universite' Libre de Bruxelles, 1997.

[CD99] Robert Castaneda and Samir R. Das. Query Localization Techniques for On-demand Routing Protocols in Ad Hoc Networks. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99)*, pages 186–194, Seattle, WA, August 1999.

[CDN+96] A. Chankhunthod, P.B. Danzig, C. Neerdaels, M.F. Schwartz, and K.J. Worrell. A hierarchical internet object cache. In *Proceedings of USENIX Annual Technical Conference*, pages 153–164, San Diego, CA, January 1996.

[CE02] A. Cerpa and D. Estrin. ASCENT: Adaptive Self-Configuring Sensor Networks Topologies. In *Proceedings of the IEEE Infocom*, New York, USA, June 2002.

[CEE+01] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat Monitoring: Application driver for wireless communications technology. In *Proceedings of the ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, pages 20–41, San Jose, Costa Rica, April 2001.

[Cen] MEMS Technology Applications Center. http://mems.mcnc.org.

[CG85] Nicholas Carriero and David Gelernter. The S/Net's Linda kernel. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, pages 110–129. ACM, December 1985.

[CHG+01] Daniel A. Coffin, Daniel J. Van Hook, Ramesh Govindan, John Heidemann, and Fabio Silva. Network routing application programmer's interface (api) and walk through 8.0. Technical Report 01-741, USC/ISI, March 2001.

[CHMK00] Daniel A. Coffin, Daniel J. Van Hook, Stephen M. McGarry, and Stephen R. Kolek. Declarative ad-hoc sensor networking. In *Proceedings of the SPIE Integrated Command Environments Conference*, pages 109–120, San Diego, California, USA, July 2000. SPIE. (part of SPIE International Symposium on Optical Science and Technology).

[CHZ02] Maurice Chu, Horst Haussecker, and Feng Zhao. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. *International Journal of High Performance Computing Applications*, 2002.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1990.

[CM99] Adam M. Costello and Steven McCanne. Search party: Using randomcast for reliable multicast with local recovery. In *Proceedings of the IEEE Infocom*, pages 1256–1264, New York, March 1999.

[Com97] IEEE Computer Society LAN MAN Standards Committee. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Technical Report 802.11-1997, Institute of Electrical and Electronics Engineers, New York, NY, 1997.

[CSWH00] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage retrieval system. In *Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, Berkeley, CA, USA, July 2000.

[CT90] D. Clark and D. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of the ACM SIGCOMM*, pages 201–208, September 1990.

[Cyb89] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7(2):279–301, July 1989.

[CZH+99] Steven E. Czerwinski, Ben Y. Zhao, Todd D. Hodes, Anthony D. Joseph, and Randy H. Katz. An architecture for a secure service discovery service. In *Proceedings of the ACM/IEEE Mobicom*, pages 24–35, Seattle, WA, USA, August 1999. ACM.

[Dee88] S. Deering. Multicast Routing in Internetworks and Extended LANs. In *Proceedings of the ACM SIGCOMM*, pages 55–64, August 1988.

[DEF+97] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, A. Helmy, and L. Wei. Protocol Independent Multicast Version 2: Dense Mode Specification. Internet-Draft, May 1997. draft-ietf-idmr-pim-dm-spec-05.txt, Work in Progress.

[DL93] Matthew Doar and Ian Leslie. How bad is naive multicast routing. In *Proceedings of the IEEE Infocom*, pages 82–89, San Francisco, CA, March 1993.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics*, 26(1):29–41, 1996.

[EE01a] Jeremy Elson and Deborah Estrin. Random, ephemeral transaction identifiers in dynamic sensor networks. In *Proceedings of the International Conference on Distributed Computing Systems*, Phoenix, Arizona, USA, April 2001. IEEE.

[EE01b] Jeremy Elson and Deborah Estrin. Time synchronization for wireless sensor networks. In *Proceedings of the 2001 International Parallel and Distributed Processing Symposium (IPDPS), Workshop on Parallel and Distributed Computing Issues in Wireless and Mobile Computing*, San Francisco, California, USA, April 2001.

[EFH+97] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol Independent Multicast – Sparse Mode (PIM-SM): Protocol Specification. RFC-2117, June 1997.

[EGE02] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. Submitted for review, February 2002.

[EGHK99] Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom'99)*, pages 263–270, Seattle, Washington, August 1999.

[FBP98] M. Faloustsos, A. Banerjea, and R. Pankaj. QoSMIC: Quality of Service Multicast Internet Protocol. In *Proceedings of the ACM SIGCOMM*, pages 144–153, September 1998.

[FDTH98] Michael Fuchs, Christophe Diot, Thierry Turletti, and Markus Hofmann. A Naming Approach for ALF Design. In *HIPPARCH Workshop Program*, University College London, UK, June 1998.

[FJ95] Sally Floyd and Van Jacobson. Link-sharing and resource management models for packet networks. *ACM/IEEE Transactions on Networking*, 3(4):365–386, August 1995.

[FJL+97] S. Floyd, V. Jacobson, C. Liu, S. Mccanne, and L. Zhang. A reliable mulicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, December 1997.

[GE01] Lewis Girod and Deborah Estrin. Robust range estimation using acoustic and multimodal sensing. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Hawaii, October 2001.

[GGSE01] Deepak Ganesan, Ramesh Govindan, Scott Shenker, and Deborah Estrin. Highly resilient, energy efficient multipath routing in wireless sensor networks. In *Proceedings of the ACM MobiHoc*, pages 251–254, Long Beach, CA, October 2001. Poster.

[GP68] E. N. Gilbert and H.O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, January 1968.

[GW97] T. Grossman and A. Wool. Computational experience with approximation algorithms for the set covering problem. *Euro. J. Operational Research*, 101(1):81–92, August 1997.

[Haa00] Jaap C. Haartsen. The Bluetooth Radio System. *IEEE Personal Communications Magazine, Special Issue on Short-Range Connectivity*, 7(1):28–36, Febuary 2000.

[HCB00] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the Hawaii International Conference on System Sciences*, Maui, Hawaii, January 2000.

[HH94] A. Harter and A. Hopper. A distributed location system for the active office. *IEEE Network, Special Issue on Distributed Systems for Telecommunications*, 8(1):62–70, January 1994.

[HM97] A. Heddaya and S. Mirdad. WebWave: Globally Load Balanced Fully Distributed Caching of Hot Published Documents. In *Proceedings of 17th IEEE International Conference on Distributed Computing Systems*, pages 160–168, Baltimore, Maryland, May 1997.

[HP98] Zygmunt J. Haas and Marc R. Pearlman. The performance of query control schemes for the zone routing protocol. In *Proceedings of the ACM SIGCOMM*, pages 167–177, September 1998.

[HSI+01] John Heidemann, Fabio Silva, Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, and Deepak Ganesan. Building efficient wireless sensor networks with low-level naming. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 146–159, Banff, Canada, October 2001.

[HSW+00] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for network sensors. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 93–104, Cambridge, MA, USA, November 2000. ACM.

[IEGH02] Chalermek Intanagonwiwat, Deborah Estrin, Ramesh Govindan, and John Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *Proceedings of the International Conference on Distributed Computing Systems*, Vienna, Austria, July 2002. IEEE.

[IG00] Tomasz Imielinski and Samir Goel. DataSpace: Querying and Monitoring Deeply Networked Collections in Physical Space. *IEEE Personal Communications Magazine, Special Issue on Smart Spaces and Environments*, 7(5):4–9, October 2000.

[IGE00] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom'2000)*, pages 56–67, Boston, Massachusetts, August 2000.

[Jac88] Van Jacobson. Congestion avoidance and control. In *Proceedings of the ACM SIGCOMM*, pages 314–329, Stanford, CA, August 1988.

[Jac90] Van Jacobson. Compressing TCP/IP headers for low-speed serial links. RFC 1144, Internet Request For Comments, February 1990.

[Jac93] Van Jacobson. Lightweight sessions – a new architecture for real time applications and protocols. Annual Principal Investigators Meeting, September 1993. ftp://ftp.ee.lbl.gov/talks/vj-nws93-2.ps.Z.

[JLT98] M. Jiang, J. Li, and Y. Tay. Cluster Based Routing Protocol (CBRP) Functional Specification. Internet-Draft, August 1998. draft-ietf-manet-cbrp-spec-00.txt, Work in Progress.

[JM96] D. B. Johnson and D. A. Maltz. Dynamic Source Routing in Adhoc Wireless Networks. In T. Imielinksi and H. Korth, editors, *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996.

[KAE00] Satish Kumar, Cengiz Alaettinoglu, and Deborah Estrin. SCalable Object-tracking through Unattended Techniques (SCOUT) . In *Proceedings of the 8th International Conference on Network Protocols(ICNP)*, Osaka, Japan, November 2000.

[Kai99] William J. Kaiser. WINS NG 1.0 Transceiver Power Dissipation Specifications. Sensoria Corp., 1999.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[KEW02] Bhaskar Krishnamachari, Deborah Estrin, and Stephen Wicker. Impact of data aggregation in wireless sensor networks. In *Proceedings of the International Workshop on Distributed Event-Based Sytems*, Vienna, Austria, July 2002.

[KK00] B. Karp and H.T. Kung. Greedy perimeter stateless routing for wireless networks. In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom'2000)*, pages 243–254, Boston, Massachusetts, August 2000.

[KKP99] J. Kahn, R. Katz, and K. Pister. Next century challenges: Mobile networking for smart dust. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom'99)*, pages 271–278, Seattle, Washington, August 1999.

[KRB99] Joanna Kulik, Wendi Rabiner, and Hari Balakrishnan. Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99)*, pages 174–185, Seattle, WA, August 1999.

[KV98] Yong-Bae Ko and Nitin H. Vaidya. Location-Aided Routing (LAR) in Mobile Ad Hoc Networks. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom'98)*, pages 66–75, Dallas, TX, October 1998.

[LHRP90] G.E. Liepins, M.R. Hilliard, J. Richardson, and M. Palmer. Genetic algorithms applications to set covering and traveling salesman problems. In Donald E. Brown and Chelsea C. White III, editors, *Operations Research and Artificial Intelligence: The Integration of Problem-Solving Strategies*, pages 29–57. Kluwer Academic Publishers, 1990.

[LP96] J. C. Lin and S. Paul. RMTP: A Reliable Multicast Transport Protocol. In *Proceedings of the IEEE Infocom*, pages 1414–1424, San Francisco, CA, March 1996.

[LSdMS97] M. Lopez-Sanchez, R. Lopez de Mantaras, and C. Sierra. Incremental map generation by low cost robots based on possibility/necessity grids. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence*, pages 351–357, Providence, RI, August 1997.

[MNR+98] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson. Adaptive Web Caching: Towards a New Global Caching Architecture. *Computer Networks and ISDN Systems*, 30(22-23):2169–2177, November 1998.

[Moy94] J. Moy. Multicast Extensions to OSPF. RFC-1584, March 1994.

[Neu89] Gerald W. Neufeld. Descriptive names in X.500. In *Proceedings of the 1989 Symposium on Communications Architectures and Protocols*, pages 64–71. ACM, September 1989.

[NTCS99] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu. The Broadcast Storm Problem in a Mobile Ad Hoc Network. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99)*, pages 151–162, Seattle, WA, 1999.

[OPSS93] Brian Oki, Manfred Pfluegl, Alex Siegel, and Dale Skeen. The information bus—an architecture for extensible distributed systems. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pages 58–68, Asheville, North Carolina, USC, December 1993. ACM.

[PB94] Charles E. Perkins and Pravin Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. In *Proceedings of the ACM SIGCOMM*, pages 234–244, August 1994.

[PC97] V. D. Park and M. S. Corson. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In *Proceedings of INFOCOM 97*, pages 1405–1413, April 1997.

[PCMN02] Rahul Pupala, Ankur Choksi, Richard P. Martin, and Badri Nath. Mobility support for diffusion-based ad-hoc sensor networks. Technical Report DCS-463, Rutgers University, Piscataway, NJ, April 2002.

[Per97] Charles Perkins. Ad-Hoc On Demand Distance Vector Routing (AODV). Internet-Draft, November 1997. draft-ietf-manet-aodv-00.txt, Work in Progress.

[Pet87] Larry L. Peterson. A yellow-pages service for a local-area network. *Proceedings of the ACM SIGCOMM '87*, pages 235–242, August 1987.

[PK00] Gregory J. Pottie and William J. Kaiser. Wireless Integrated Network Sensors. *Communications of the ACM, Special Issue on Embedding the Internet*, 43(5):51–58, May 2000.

[PPV98] C. Papadopoulos, G. Parulkar, and G. Verghese. An Error Control Scheme for Large-scale Multicast Applications. In *Proceedings of the IEEE Infocom*, pages 1188–1196, San Francisco, March 1998.

[pro98] The WINS project. http://www.janet.ucla.edu/WINS/, 1998.

[REG+02] Sylvia Ratnasamy, Deborah Estrin, Ramesh Govindan, Brad Karp, Scott Shenker, Li Yin, and Fang Yu. Data-centric storage in sensornets. Submitted for review, February 2002.

[RM98] Suchitra Raman and Steven McCanne. Scalable data naming for application level framing in reliable multicast. In *Proceedings of ACM Multimedia*, pages 391–400, Bristol, UK, September 1998.

[SDC97] Devika Subramanian, Peter Druschel, and Johnny Chen. Ants and Reinforcement Learning: A Case Study in Routing in Dynamic Networks. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 832–839, Nagoya, Japan, August 1997.

[SEFJ97] Puneet Sharma, Deborah Estrin, Sally Floyd, and Van Jacobson. Scalable timers for soft state protocols. In *Proceedings of the IEEE Infocom*, pages 222–229, Kobe, Japan, April 1997. IEEE.

[Sen93] S. Sen. Minimal cost set covering using probabilistic methods. In *Proceedings 1993 ACM/SIGAPP Symposium on Applied Computing*, pages 157–164, 1993.

[SGAP00] Katayoun Sohrabi, Jay Gao, Vishal Ailawadhi, and Gregory J. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications Magazine, Special Issue on Smart Spaces and Environments*, 7(5):16–27, October 2000.

[SHBR96] Ruud Schoonderwoerd, Owen Holland, Janet Bruten, and Leon Rothkrantz. Ants for Load Balancing in Telecommunications Networks. Technical Report HPL-96-35, Hewlett-Packard Laboratories Bristol, 1996.

[SJ98] F. Stajano and A. Jones. The thinnest of clients: Controlling it all via cellphone. *ACM Mobile Computing and Communications Review*, 2(4):46–53, October 1998.

[SK97] M. Stemm and R.H. Katz. Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEICE Transactions on Communications*, E80-B(8):1125–1131, August 1997.

[SR98] S. Singh and C.S. Raghavendra. PAMAS: Power aware multi-access protocol with signalling for ad hoc networks. *ACM Computer Communication Review*, 28(3):5–26, July 1998.

[Sut90] Richard S. Sutton. Reinforcement learning architectures for animats. In *From Animals to Animats*, pages 288–296. The MIT Press, 1990.

[TM80] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Math. Japonica*, 24(6):573–577, 1980.

[TSS+97] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

[Tur52] A. M. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of Royal Society of London*, 237(B):37–72, August 1952.

[Wal99] Jim Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(10):76–82, October 1999.

[Wax88] Bernard M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, December 1988.

[Wax93] Bernard M. Waxman. Performance evaluation of multipoint routing algorithms. In *Proceedings of the IEEE Infocom*, pages 980–986, San Francisco, California, March 1993.

[Wei91] M. Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):66–75, September 1991.

[WH92] Roy Want and Andy Hopper. Active badges and personal interactive computing objects. *IEEE Transactions on Consumer Electronics*, 38(1):10–20, February 1992.

[WHFG92] Roy Want, Andy Hopper, Veronica Falcao, and Jon Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91–102, January 1992.

[Win87] P. Winter. Steiner problem in networks: A survey. *Networks*, 17(2):129–167, 1987.

[WJH97] A. Ward, A. Jones, and A. Hopper. A new location technique for active office. *IEEE Personal Communications Magazine*, 4(5):42–47, October 1997.

[WTZ99] Lan Wang, Andreas Terzis, and Lixia Zhang. A new proposal for RSVP refreshes. In *Proceedings of the International Conference on Network Protocols*, pages 163–172, Toronto, Canada, October 1999. IEEE.

[XHE01] Ya Xu, John Heidemann, and Deborah Estrin. Geography-informed energy conservation for ad hoc routing. In *Proceedings of the ACM/IEEE Mobicom*, pages 70–84, Rome, Italy, July 2001. ACM.

[YGE01] Yan Yu, Ramesh Govindan, and Deborah Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. Technical Report UCLA/CSD-TR-01-0023, UCLA, May 2001.

[YHE02] Wei Ye, John Heidemann, and Deborah Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings of the IEEE Infocom*, New York, USA, June 2002.

[YHK95] W. Yeong, T. Howes, and S. Kille. Lightweight directory access protocol. RFC 1777, Internet Request For Comments, March 1995.

[ZGE02] Yonggang Jerry Zhao, Ramesh Govindan, and Deborah Estrin. Residual Engery Scans for Monitoring Wireless Sensor Networks. In *IEEE Wireless Communications and Networking Conference*, Orlando, FL, March 2002.

[ZSR02] F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration for tracking applications. *IEEE Signal Processing Magazine*, March 2002.

Appendix A

ISI Implementation of Directed Diffusion

We have so far described directed diffusion using a specific application as an example. However, it would be desirable to avoid re-implementing diffusion mechanisms for every new application. To this end, the SCADDS group has implemented a generic diffusion substrate and ported the code to multiple platforms. On top of this substrate, several applications have been implemented.

A.1 Diffusion Substrate

The generic diffusion substrate exports two Application Programming Interfaces (APIs): a *network routing* API, and a *filter* API. The former is invoked on sources and sinks, while the latter enables in-network processing of events.

A.1.1 Network API

The C++ *Network Routing* API is based on a publish-subscribe paradigm and is summarized in Figure A.1 (see [CHG+01] for a complete specification and example source code). The interface supports two operations. Sinks can *subscribe* to named

```
handle NR::subscribe(NRAttrVec *subscribeAttrs,
                     const NR::Callback * cb);
int NR::unsubscribe(handle subscription_handle);
handle NR::publish(NRAttrVec *publishAttrs);
int NR::unpublish(handle publication_handle);
int NR::send(handle publication_handle,
             NRAttrVec *sendAttrs);
```

Figure A.1: Basic diffusion API.

events (for a particular set of attributes) and sources can *publish* events. The diffusion substrate hides the details of how published data is delivered to subscribers, the routing algorithms we have described in Chapter 2. Events are sent and arrive asynchronously. When events arrive at a node, they trigger callbacks to relevant applications (those that have subscribed with matching attributes).

Applications that generate information publish that fact and then *send* specific data. The attributes specified in the publish call must match the subscription. If there are no active subscriptions, published data does not leave the node. As a further optimization sensor nodes may wish to avoid generating data that has no takers. In this case the application would *subscribe for subscriptions* and would be informed when subscriptions arrive or terminate.

In diffusion, data is named using a collection of attribute-value pairs. A key feature of the network routing API is that it defines a generic attribute class. Each attribute has several fields:

- the *key* indicates the semantics of the attribute (latitude, longitude, frequency). Attributes are identified by unique keys (implemented as simple 32-bit numbers in practice) drawn from a central authority.

- the *operator* describes how the attribute will match when two attributes are compared. The operator field defines how data messages and interests interact. Operators include the usual binary comparisons (EQ, NE, LE, GT, LE, GE, corresponding to equality, inequality, less than, *etc.*), "EQ_ANY" (which matches anything), and IS. "IS" allows users to specify an *actual* (literal or bound) value, while all the other operators specify *formal* (a comparison or unbound) parameters for comparison.

- the *type* indicates what algorithms to run when matching attributes. We have currently implemented 32-bit integer, both 32 and 64-bit floats, string, and blob (uninterpreted) attributes.

- the actual *value* of the attribute, and its *length* (if length is not implicit from the type)

This approach has the advantage of standardizing the syntax and structure of attributes, and allowing applications to reuse attribute handling and matching code. A *one-way match* compares all formal parameters of one attribute set against the actuals of the others (Figure A.2). Any formal parameter that is missing a matching actual in the other attribute set causes the one-way match to fail (for example, "confidence GT 0.5" must have an actual such as "confidence IS 0.7" and would not match "confidence IS 0.3", "confidence LT 0.7", or "confidence GT 0.7"). Two sets of attributes have a *complete match* if one-way matches succeed in both directions. In other words, attribute sets A and B match if the one-way match algorithm succeeds from both A to B and B to A.

```
one-way match:
given two attribute sets A and B
for each attribute a in A where a.op is a formal {
    matched = false
    for each attribute b in B where a.key = b.key and b.op is an actual
        if a.val compares with b.val using a.op, then matched = true
    if not matched then return false (no match)
}
return true (successful one-way match)
```

Figure A.2: Our one-way matching algorithm.

This matching style is similar to the rules used in other attribute-based languages (for example, Linda [CG85] and INS [AWSBL99]), but the ISI diffusion implementation adds two-way matching and a range of operators in addition to equality. When multiple attributes and operators are present they are effectively "anded" together; all formals must be satisfied for a match to be successful. This approach strikes a balance between ease of implementation and flexibility. The simple bounded set of operators can be implemented in tens of lines of code and yet supports, for example, rectangular regions.

To see how diffusion and attribute matching interact, we continue the example from Chapter 2 where a user asks a sensor network to track four-legged animals. The user's query translates into an interest with the attributes (type EQ four-legged-animal-search, interval IS 10ms, duration IS 10 minutes, x GE –100, x LE 200, y GE 100, y LE 400). Also, an implicit "class IS interest" attribute is added to identify this message as an interest (as opposed to data). This interest specifies five conditions: detection of animals in a particular region specified by a rectangle. It also provides

information about how frequently data should be returned and how long the query should last.

Sensors in the network are programmed with animal search routines (either by pre-programming at deployment time or by downloading mobile code). Such sensors would watch for interests in animals by expressing *interests about interests* with attributes (class EQ interest, type IS four-legged-animal-search, x IS 125, y IS 220). When the user's interest arrives at the sensor it would activate its sensor using the parameters provided (duration and interval) and reply if it detects anything.

When the sensor detects something the data message would include attributes (type IS four-legged-animal-search, instance IS elephant, x IS 125, y IS 220, intensity IS 0.6, confidence IS 0.85, timestamp IS 1:20, class IS data). This message satisfies the original interest. It encodes as attributes additional information about what was seen and what confidence the sender has in its detection.

This example illustrates the details of a specific query. It shows how named data provides a convenient way of encoding information, and how geometry and well-known attributes allow simple matching rules work for this application. Although this example uses several attributes, some applications may use only a subset of these methods, omitting geographic constraints (in a small sensor network) or using a single attribute (when there is only one sensor type). We have found that these primitives provide good building blocks for a range of applications; we describe these in Section A.3.

Although matching is reasonably powerful, it does not perfectly cover all scenarios or tasks. Simple matching in these cases can approximate what is required, and application-specific code can further refine the choice. For example, perfect

rectangles aligned with the coordinate system are insufficient to describe arbitrary geometric shapes. Non-rectangular shapes can be accomplished either by multiple queries, or by using the smallest bounding rectangle and having the application ignore requests inside the rectangle but outside the required region. Similarly, applications can use general attributes that are clarified with sub-attributes or parameters (type IS animal-search, subtype IS four-legged). Filters (described next) also allow applications to influence processing.

A.1.2 Filter API

While the publish/subscribe API allows end-points to send and receive data, in-network processing with the *Filter* API is key to diffusion performance. Application-specific *modules* can install *filters* in the diffusion substrate to influence data as it moves through the network. Each filter is specified using a list of attributes to match incoming data. When a data event that matches a filter is received, the substrate passes the event to the application module. The module may perform some application-specific processing on the event; it may aggregate the data, generate reinforcements, or even issue new subscriptions using the network routing API. In the event that the event matches filters belonging to more than one application module, a static priority ordering determines which module is handed the event first. That module may then decide to also allow other application modules corresponding to lower-priority filters to handle the event, or may choose not to do so.

Applications provide filters before deployment of a sensor network, or in principle filters could be distributed as mobile code packages at run-time. Filters have access to internal information about diffusion, including gradients and lists of neighbor

nodes. Filters are typically used for in-network aggregation, collaborative signal processing, caching, and similar tasks that benefit from control over data movement. In addition to these applications, we have found them very useful for debugging and monitoring.

Continuing our example, a filter can be used to suppress concurrent detections of four-legged animals from different sensors. It would register interest in detection interests and data with attributes (type IS four-legged-animal-search). It could then record what the desired interval is, then allow exactly one reply every interval units of time, suppressing replies from other sensors. A more sophisticated filter could count the number of detecting sensors and add that as an additional attribute, or it could generate some kind of aggregate "confidence" rating in some application-specific manner. In this example filtering may discard some data, but by reducing unnecessary communication it will greatly extend the system's operational lifetime.

Filter-specific APIs are shown in Figure A.3. A filter is primarily a callback procedure (the `cb` specified in addFilter) that is called when matching data arrives. Rather than operate only on attribute vectors, filters are given direct access to messages that include identifiers for the previous and next immediate destinations. The SCADDS group is currently evaluating using this additional level of control to optimize diffusion, for example using geographic information to avoid flooding exploratory interests. We expect these interfaces to be extended as we gain more experience with how filters are used and what information they require.

Finally, these APIs have been designed to favor an event-driven programming style, although they have been successfully used in multi-threaded environments such as WINSng 1.0. The SCADDS group has targeted event-driven programming

```
handle addFilter(NRAttrVec *filterAttrs,
          int16_t priority, FilterCallback *cb);
int NR::removeFilter(handle filter_handle);
void sendMessage(Message *msg, handle h,
                 int16_t agent_id = 0);
void sendMessageToNext(Message *msg, handle h);
```

Figure A.3: Filter APIs.

to avoid synchronization errors and to avoid the memory and performance overheads of multithreading. Evidence is growing that event-driven software is well suited to embedded programming, particularly on very memory-constrained platforms [HSW+00].

Also we allow filters and applications to run in the same or different memory address spaces from each other and the diffusion core. Single-address space operation is necessary for very small sensor nodes that lack memory protection and as a performance optimization. Multiple address spaces may be desired for robustness to isolate filters of different applications from each other.

A.2 Platforms

Details of the C++-level APIs [CHG+01] and how the attribute system affects applications [HSI+01] are available elsewhere. The diffusion substrate is about 7,000 lines of C++ code (downloadable from `http://www.isi.edu/scadds`). The SCADDS group has ported the full diffusion code to two different platforms. The first is the WINSng 1.0 nodes (Figure A.4(b)), a Windows-CE-based, commercial sensor node with a specialized preprocessor containing signal processing hardware [PK00]. A preprocessor interfaces to various kinds of sensors, allowing the main CPU to operate

with a very low duty cycle. The radio on this device is a proprietary TDMA-based frequency-hopping radio.

Our second platform [CEE+01] is an x86-based sensor node (Figure A.4(a) assembled using largely off-the-shelf hardware (a PC/104, small-form-factor motherboard with a 66MHz AMD 486-class CPU and 16MB of RAM and flash disk), radio (a packet-controller-based 418MHz radio with 13kb/s throughput from Radiometrix, Ltd.), and software (Linux). Since the diffusion substrate is relatively platform independent, porting from one platform to another was quite simple.

The SCADDS group has also implemented *micro-diffusion*, a bare subset of diffusion designed to run on Motes with tiny 8-bit processors and only 8KB of memory (Figure A.4(c)). It is distinguished by its extremely small memory footprint and a complementary approach for deployment to our full system. Micro-diffusion retains only gradients, condenses attributes to a single tag, and supports only limited filters. As a result it adds only 2050 bytes of code and 106 bytes of data to its host operating system. (By comparison, our full system requires a daemon with static sizes of 55KB code, 8KB data, and a library at 20KB code, 4KB data.) Micro-diffusion is implemented as a component in TinyOS [HSW+00] that adds 3250B code and 144B of data (including support for radio and a photo sensor), so the entire system runs in less than 5.5KB of memory. Micro-diffusion is statically configured to support 5 active gradients and a cache of 10 packets of the 2 relevant bytes per packet. Although reduced in size, the logical header format is compatible with that of the full diffusion implementation and the SCADDS group is implementing software to gateway between the implementations. Although we do not currently provide filters in micro-diffusion, they are an essential component of enabling in-network aggregation

in diffusion, and we plan to add them. The SCADDS group intends to leverage on the ability to reprogram motes over the air [HSW+00] to program filters dynamically.

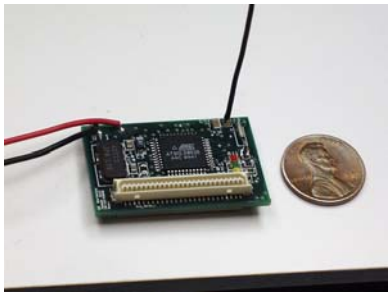
Motes and micro-diffusion can be used in regions where there is need for dense sensor distribution, such as distributing photo sensors in a room to detect change in light or temperature sensors for fine grained sensing. They provide the necessary sensor data processing capability, with the ability to use diffusion to communicate with less resource-constrained nodes (for example, PC/104-class nodes). Motes can also be used to provide additional multi-hop capability under adverse wireless communication conditions.

The SCADDS group thus envisages deployment of a tiered architecture with both larger and smaller nodes. Less resource-constrained nodes will form the highest tier and act as gateways to the second tier. The second tier will be composed of motes connected to low-power sensors running micro-diffusion. Most of the network "intelligence" is programmed into the first tier. Second-tier nodes will be controlled and their filters programmed from these more capable nodes.

All of the SCADDS implementations build upon a simple radio API that supports broadcast or unicast to immediate neighbors. Neighbors must have some kind of identifier, but it is not required to be persistent. We can use persistent identifiers (for example, Ethernet MAC addresses) or operate with ephemerally assigned identifiers [EE01a].

(a) The SCADDS PC/104 node

(b) WINSng 1.0 node



(c) UCB Rene Mote

Figure A.4: Diffusion operational platforms.

A.3 Applications

To date, the SCADDS group has built several applications on top of the diffusion substrate:

- A collaborative detection application in which several nodes perceive a target, and increase their confidence in the detection by exchanging events with other nodes. This application was developed in collaboration with researchers at BAE systems and the Pennsylvania State University. Specifically, BAE systems contributed signal processing code and systems integration, while PSU provided sensor fusion algorithms [BI96]. The combined system used our system to communicate data between sensors using named data and diffusion.

- An application in which nodes with photo-sensors cue nodes containing acoustic sensors when they detect activity. These latter sensors transmit sound to users. This application represents two-level diffusion routing, a slightly more sophisticated form than the application described in Chapter 2. We called this a *nested-query* or *correlated-query* application (Section 3.3.1.2).

- A distributed database system where a user node tasks query proxies that are active on multiple sensors to collaborate in object detection [BGMS99]. Researchers at Cornell have used our system to provide communication between an end-user database and application that represents and visualizes a sensor field and query proxies in each sensor node. This application used attributes to identify sensors running query proxies and to pass query byte-codes to the proxies. *They also originated the idea of using a nested approach for nested*

queries. Future work includes understanding what network information is necessary for database query optimization and alternative approaches for nested queries.

- An application for adaptively self-configuring a sensor network to achieve low loss communication in dense deployments without reducing network lifetime.

The SCADDS group is also evaluating the use of diffusion for tracking applications with researchers. Our experiences with diffusion implementation have been discussed in greater detail in [HSI+01].

A.4 Experiences

Our experiences with diffusion have been quite positive, even with very limited radio and compute resources (our operational platforms have radios with about 13kb/s bandwidth and the compute power of an Intel 80486). Our observations about diffusion fall in to operational and application-level issues.

Operationally, diffusion is affected by the MAC and both internal and external parameters. The choice of MAC layer is critical for sensors (as we observed in simulation, Chapter 3). The MAC of our Radiometrix radio lacks RTS/CTS and ARQ, so successful use requires careful attention to network load. Internally, diffusion includes a number of parameters (timers and the ratio of exploratory messages to data traffic) that may need to be tuned for particular applications and networks. Additional work is required for understanding how to tune these parameters. Additional work is also needed to understand how externally to monitor and control a

distributed sensor network. The SCADDS group is developing both in-band (radio-based) and out-of-band (Ethernet-based) monitoring tools to help tune and deploy sensor networks, but more work is required here.

From an application viewpoint, data-driven, publish/subscribe networking has a very different feel than traditional socket-based, Internet-style programming. For example, asking for data from a region and getting back some set of answers provides different completeness, robustness, and performance characteristics than enumerating all nodes in a region and then querying each one individually. Applications work best if they reflect diffusion's strengths rather than try and reimplement TCP-style communication (although that is possible). More work is required to understand how applications and diffusion interact efficiently, and how applications can control diffusion costs when necessary. Filters provide a valuable tool here, and the SCADDS group is currently experimenting with them for this purpose.

In general, we have been quite happy with the SCADDS implementation, specifically the use of attributes, simple matching rules, and the publish/subscribe API. The most difficult part of the implementation was coping with multiple sources and sinks when the network changes; as with multicast routing, loop suppression can be quite subtle.

Appendix B

Directed Diffusion Pseudocode

This chapter summarizes the basic diffusion algorithms and data structures in pseudocode. Based on the SCADDS implementation, this pseudocode is slightly different from that of our simulation.

There are three basic data structures: messages, route entries, and cache entries. Generally, *messages* consist of the message type (interest, data, positive and negative reinforcements) and attributes (the data payload). Data messages also include a flag indicating whether they are exploratory or not. *Route entries* relate attributes to gradients in each node. Each route entry includes an expiration timer, a list of gradients to each neighbor, and a list of upstream neighbors. An expiration timer allows the gradients to be aged. Each gradient includes a flag for indicating if the associated neighbor has expressed an interest in non-exploratory data (*i.e.,* is "marked"). Each upstream neighbor entry also includes an indication of "route quality"; whether or not we have seen new, old, or no data from that last hop. Data sources keep track of when exploratory messages should be refreshed (next exploratory time) whereas data sinks list local applications that want this kind

of data. Finally, the *message cache* in each node logically stores copies of prior messages and interests. This cache is used to suppress duplicate messages and can potentially be used for data aggregation or other kinds of in-network data processing. If duplicate suppression is the only user, as an optimization, the message cache can replace attribute contents with simply a hash of the contents to reduce storage costs.

Diffusion performance is governed by the timers (for interest refresh, loop suppression, and exploratory messages) and the timeout thresholds (for gradients and the message cache).

```
interest_initiation(attrs, sink_process) {
    // Invoked spontaneously at sinks
    re = find_or_create_re(attrs);
    list_append(re->local_sinks, sink_process);
    interest_refresh(re);
}

interest_refresh(re) {
    // Invoked at sink nodes only
    msg = create_msg(INTEREST, attrs);
    interest_handling(msg, NULL);
    schedule(interest_refresh(re),
                now + INTEREST_REFRESH_INTERVAL)
}

interest_handling(msg, last_hop) {
```

```
    // Invoked at all nodes
    re = find_or_create_re(msg->attrs, last_hop);
    gradient = find_or_create_gradient(re, last_hop);
    gradient->expire_time = now + GRADIENT_TIMEOUT_THRESHOLD;
    if (duplicate_suppress(msg)) return;
    send_to_all_neighbors(msg);
}

data_initiation(attrs) {
    // Invoked spontaneously at source nodes
    // source processing --- send it out if there are gradients
    re = find_re(attrs);
    if (!re) return;
    msg = create_msg(DATA, attrs);
    // marking policy:
    if (!re_has_any_marked_gradients(re) ||
            now > re->next_explore_time) {
        msg->exploratory = true
        re->next_explore_time = now + EXPLORE_TIME_INTERVAL;
    }
    data_handling(msg, NULL)
}

data_handling(msg, last_hop) {
    // Invoked at all nodes
```

```
    re = find_re(msg->attrs);
    if (duplicate_suppress(msg)) {
        if (re->data_quality(na) != NEW)
            re->data_quality(na) = OLD;
        return;
    }
    re->data_quality(na) = NEW;
    // sink processing
    if (re->local_sinks != NULL)  {
        if (msg->exploratory) {   // reinforce policy
            reinf_msg = create_msg(POS_REINFORCEMENT, attrs);
            send_to_one(last_hop, reinf_msg);
        }
        send_to_local_sinks(msg);
    }
    // send policy
    if (msg->exploratory)
        send_to_all_gradients(re, msg);
    else if (re_has_any_marked_gradients(re))
        send_to_all_marked_gradients(re, msg);
    else if (re->local_sinks == NULL) {
        // suppress out-of-date paths
        send_to_one(last_hop, create_msg(NEG_REINFORCEMENT, attrs));
    };
}
```

```
pos_reinforcement_handling(msg, last_hop) {
    // Invoked at all nodes
    re = find_re(msg->attrs);
    re->gradient(last_hop)->marked = false;
    next_last_hop = cache(msg->attrs)->last_hop;
    if (next_last_hop != NULL)  // propagate
        send_to_one(next_last_hop, msg);
}

neg_reinforcement_handling(msg, last_hop) {
    // Invoked at all nodes
    re = find_re(msg->attrs);
    re->gradient(last_hop)->marked = false;
    if (! re_has_any_marked_gradients(re))  // propagate
        send_to_all_neighbors(msg);
}

loop_suppression_timer(re) {
    // Runs periodically for every RE on all nodes
    foreach na in re->data_quality {
        if (re->data_quality(na) == OLD)
            send_to_one(na,
                create_msg(NEG_REINFORCEMENT, re->attrs));
        re->data_quality(na) = NONE;
```

```
    }
    schedule(loop_suppresion_timer(re),
                now + LOOP_SUPPRESSION_INTERVAL);
}


// UTILITY FUNCTIONS:

duplicate_suppress(msg) {
    cache_entry = find_cache_entry(msg->attrs);
    if (cache_entry) return true;
    cache_entry = cache_add(msg, now);
    schedule cache_expiration(cache_entry,
                now + CACHE_TIMEOUT_INTERVAL);
    return false;
}

cache_expiration(cache_entry) {
    delete cache_entry;
}

create_msg(type, attrs) {
    // create and fill in msg with type and attrs
}
```

```
find_re(attrs) {
    // search through local REs for attrs that match
}


find_or_create_re(attrs, last_hop) {
    re = find_re(attrs);
    if (!re) {
        re = new re;
        re->attrs = attrs;
        re->gradient(last_hop)->marked = false
        re->last_refresh_time = now;
        re->last_explore_time = 0;
        re->local_sinks = NULL;
        schedule(loop_suppresion_timer(re),
                now + LOOP_SUPPRESSION_INTERVAL);
    }
}


re_has_any_marked_gradients(re) {
    foreach na in re->gradient(na)
        if (re->gradient(na)->marked)
            return true;
    return false;
}
```

```
find_or_create_gradient(re, na) {
    if (exists re->gradient(na))
        return re->gradient(na);
    gradient = new gradient(not marked);
    schedule(gradient_expiration(re, na),
                now + GRADIENT_TIMEOUT_THRESHOLD);
}

gradient_expiration(re, na) {  // executed at all nodes
    if (now < re->gradient(na)->expire_time)
        return;
    delete re->gradient(na);
    if (re has no gradients && re->local_sinks == NULL)
        delete re;
}
```