Matching Data Dissemination Algorithms to Application Requirements *

ISI-TR-571 16 April 2003

John Heidemann† Fabio Silva† Deborah Estrin†‡

† USC/Information Sciences Institute
4676 Admirality Way
Marina del Rey, CA, USA 90292

‡ Computer Science Department
University of California, Los Angeles
Los Angeles, CA, USA 90095

{johnh,fabio,estrin}@isi.edu

ABSTRACT

A distinguishing characteristic of wireless sensor networks is the opportunity to exploit characteristics of the application at lower layers. This approach is encouraged by device resource constraints, and acceptable because devices are inexpensive and numerous enough that they can be dedicated to specific applications. Many data dissemination protocols have been proposed for multi-hop communication in sensor networks, each evaluated in some scenario. The premise of this paper is that, if protocols are designed to exploit application requirements, then no one protocol can be optimized for all applications. Instead, a family of protocols are needed, with guidance to match protocol to application. We show through field experiments with two tracking applications that choice of diffusion algorithm can affect application performance by 40–60%. These applications motivate the design of two new diffusion algorithms: pull and one-phase push diffusion. We describe these algorithms in comparison to previous algorithms, then systematically explore their performance as the number of sinks and sources, the traffic rate and node placement varies, and with and without geographic proximity in node placement and with and without geographically scoped communication. We characterize algorithm performance and highlight the effect of the choice of algorithm parameters. The end result of this work are guidelines to help application developers to match dissemination algorithms to application performance requirements.

1. INTRODUCTION

Data dissemination approaches in sensor networks have adopted application-specific, *data-centric* communications protocols to reduce overhead by avoiding levels of abstraction and to support application involvement in communication. This opportunity and flexibility has prompted exploration of an even wider range of data dissemination protocols, from protocols supporting attribute-addressed, peer communication (such as directed diffusion [17]) to protocols supporting tree-based communication to a single sink with an SQL abstraction [23]. Optimizations now being considered include geographic routing [18, 30], multipath routing (for example, see [1]), rendezvous-based approaches [4, 25], and many others.

Application involvement in sensor-network communications is an important complement to basic dissemination algorithms. Application-specific constraints and optimizations greatly reduce communications cost by replacing communication with computation in the network. As sensor network deployment grows, we expect to see a growing range of application techniques. Today, a common class of applications uses a sensor net to communicate data from the net to a single sink, possibly with opportunistic data processing along the way [17, 23]. More complex applications do controlled in-network processing, for purposes such as collaborative signal processing [27], or to localize computation with nested queries [15], or to place or control database operators [3, 23]. Other applications involve increasingly sophisticated operation, with multiple kinds of distributed interaction and communication, including point-to-point state transfer and region-based suppression [22].

As the choices of protocols and the sophistication of applications grows, an important problem is selection of which communications algorithms best match applications, and how the two interact. In some cases semantics decide the question, for example, if the application requires geographic routing, GPSR's "route to node nearest this point" is a good match. But when multiple implementations provide similar semantics, other factors dominate the choice.

Originally directed diffusion implied both a programming interface and a specific routing implementation [17]. Since then, a programming interface has been defined that separates the API and naming abstraction from the routing implementation [15]. We are beginning to explore multiple different kinds of routing algorithms underneath this common API. The original directed diffusion algorithm employed flooding of interests with localized algorithms [17]. While appropriate to applications with a small

*This work was supported by DARPA under grant DABT63-99-1-0011 as part of the SCADDS project.

number of data receivers, its overhead becomes high when many nodes become interested in data since each one sends its interests throughout the network. Motivated by application performance requirements, we have developed two new algorithms that support the diffusion API: *push diffusion*, an algorithm optimized for many receivers but few senders, and *one-phase pull*, an algorithm optimized for many senders and few receivers. Additional work has explored augmenting this mechanisms with geographic scoping [30], and explored rendezvous-based approaches [4, 25].

In these cases, multiple implementations provide similar semantics, and so performance considerations dominate the choice of algorithm. Although documentation provides brief descriptions of the algorithms, in practice we have observed that it is often difficult for application developers to match communication algorithms to their needs. Application developers are not networking experts, and evaluation of alternative dissemination algorithms is difficult to do in the field where configuration constraints, limited debugging tools, developing applications, and limited time preclude systematic comparisons. Yet we have observed that a careful choice of algorithm can cut overall traffic in half relative to alternatives.

This paper is motivated by this combination of large performance gains possible from carefully matching algorithm and application, and by the corresponding difficult and limited information about how to make this choice. We highlight algorithm differences that affect performance (Section 2) and describe field experiments that show the performance gains of 40–60% from improved algorithm choices (Section 3). We then systematically compare how algorithm performance changes as a function of the number of sources and sinks, geographic proximity and geographically scoped communication, and sending rate (Section 4). By emulating a radio network and evaluating real implementations of these algorithms we identify cases where asymptotic performance is not predictive of algorithm performance over typical sensor network sizes. For example, even though the overhead of push diffusion grows linearly with the number of sources, constant factors make push control overhead one-third that of one-phase pull with five sources and five sinks (Section 4.2).

The key contributions of this paper are to demonstrate the importance of matching dissemination algorithm to application, to describe push and one-phase pull, two new implementations of the diffusion API that are tuned to match two new classes of applications, and to offer guidelines about matching algorithms to applications with a thorough evaluation of implementation performance.

2. SUMMARY OF DIFFUSION ALGORITHMS

Early visions [13] of directed diffusion identified its key characteristics: localized algorithms, named data, and support for in-network processing. Diffusion adopted a declarative, publish/subscribe API that isolates data producers and consumers from the details of the underlying data dissemination algorithms [8]. The key abstraction of this API is that data is identified by a set of attributes, data producers (or *sources*) generate data it by *publishing*, data consumers (or *sinks*) *subscribe* to data, and it is the business of the diffusion implementation to insure that data travels from publisher to subscriber efficiently. Diffusion encourages applications to influence data flow through the use of filters [16] and in-network processing, but many applications require only attribute-selected data and allow diffusion to completely control routing.

Many different algorithms can match publishers and subscribers without change to the high-level API or semantics. Initial work with diffusion used an algorithm we would now call two-phase pull [17] where data consumers seek out data sources, and then sources search to find the best possible path back to subscribers. GEAR optimizes the process of finding sources by using geographic information to constrain the search process [30].

While we found these protocols ideal for some applications, increasing experience with a broader range of applications suggested that two-phase pull, is a poor match for some applications. As described in Section 3, some applications have many sources and sinks cross-subscribed to each other, a case that results in a large amount of control traffic, even with geographic scoping. To address this problem, this paper introduces *push* diffusion, an algorithm that reverses the role of data publishers and subscribers, causing data sources to actively search for consumers.

An advantage of push is that it requires only one phase where control traffic needs to be widely disseminated to find sinks, unlike the two phases needed in two-phase pull. Inspired by this observation, we developed *one-phase pull*, a third diffusion algorithm that simplifies two-phase pull by eliminating one side of the search.

We briefly review two-phase pull and GEAR and then introduce push and one-phase pull below.

2.1 Two-phase pull diffusion

Initial work with diffusion used an algorithm we would now call two-phase pull [17]. A subscriber, or data *sink*, identifies data by a set of attributes. This information propagates through the network in an *interest* message. In principle, information cached from prior runs, other constraints (such as geographic information as in GEAR), or application-specific filters can can be used to optimize the distribution of interests. Without such information, however, interests must be flooded throughout the network to find any data sources. As they are distributed, nodes establish *gradients*, state indicating the next-hop direction of other nodes interested in the data.

When an interest arrives at a data producer, that *source* begins producing data. (To conserve power, nodes may avoid producing data before being triggered, or they may produce and store such data locally.) The first data message sent from the source is marked as *exploratory*[1] and is sent to all neighbors that have matching gradients. As with interest messages, this transfer could be limited using additional information or application involvement, but by default it is sent to all nodes. When exploratory data reaches the sink, the sink *reinforces* its preferred neighbor, establishing a reinforced gradient towards the sink. (Preference being given to the lowest latency neighbor, possibly modified by other concerns such as link quality or energy.) The reinforced neighbor reinforces its neighbor in turn, all the way back to the data source or sources, resulting in a chain of reinforced gradients from all sources to all sinks.

Subsequent data messages are not marked exploratory, and are sent only on reinforced gradients rather than to all neighbors.

Nodes can also generate *negative reinforcements* if they receive data that is not relevant to them. Typically this occurs when topology changes and multiple gradients accidentally point to the same node. A negative reinforcement corrects this situation.

[1]Prior work [17] used the term “low-rate” data for this concept.

Gradients are managed as soft-state, thus both interests and exploratory data occur periodically to refresh this state. Interests are sent every *interest interval*, exploratory data every *exploratory interval*. In the basic implementation, the interest interval is 30s, the exploratory interval 90s. Since application data is marked exploratory (rather than there being an explicit exploratory control message), if the application send rate is lower than one event every 90s, the exploratory rate will also be reduced.

2.2 Geographically scoped data with GEAR

The physical nature of a sensor network's deployment makes geographically scoped queries natural. If nodes know their locations, then geographic queries can influence data dissemination, limiting the need for flooding to the relevant region.

GEAR (Geographic and Energy-Aware Routing) extends diffusion when node locations and geographic queries are present [30]. GEAR is an extension to an existing diffusion algorithm that replaces network-wide communication with geographically constrained communication. When added to two-phase pull diffusion, GEAR's subscribers actively send interests into the network. However, queries expressing interest in a region are sent *towards* that region using greedy geographic routing (with support for routing around holes); flooding occurs only when interests reach the region rather than sent throughout the whole network. Exploratory data is sent only on gradients set up by interests, so the limited dissemination of interests also reduces the cost of exploratory data.

GEAR provides a first example of application-specific diffusion. It optimizes diffusion for applications and networks that have geographically scoped queries. GEAR-extended versions of push and one-phase pull are also available and described below.

2.3 Push diffusion

Two-phase pull works well for applications where a small number of sinks collects data from the sensor net, for example, a user querying a network for detections of some tracked object. Another class of applications involves sensor-to-sensor communication within the sensornet. A simple example of this class of application might have sensors operating at a low duty cycle most of the time, but when one sensor detects something it triggers nearby sensors to become more active and vigilant. This problem was described by researchers at Sensoria, University of Wisconsin, and PARC. A characteristic of this class of application is that there are many sensors interested in data (activation triggers), and many that can publish such data, but the frequency of triggers actually being sent is fairly rare. Two-phase pull diffusion behaves poorly for this application, because all sensors actively send interests and maintain gradients to all other sensors even though nothing is detected.

Push diffusion was designed for this application. Although the API is the same as two-phase diffusion (except for a flag to indicate "push"), in the implementation, the roles of the source and sink are reversed. Sinks become passive, with interest information kept local to the node subscribing to data. Sources become active; exploratory data is sent throughout the network without interest-created gradients. As with two-phase pull, when exploratory data arrives at a sink a reinforcement message is generated and it recursively passes back to the source creating a reinforced gradient, and non-exploratory data follows only these reinforced gradients. Push can also take advantage of GEAR-style geographic optimizations.

Push is thus optimized for a different class of applications from two-phase pull: applications with many sources and sinks, but where sources produce data only occasionally. Push is not a good match for applications with many sources continuously generating data since such data would be sent throughout the network even when not needed.

2.4 One-phase pull diffusion

A benefit of push diffusion compared to two-phase pull is that it has only one case where information is sent throughout the network (exploratory data) rather than two (interests and exploratory data). In large networks without geographically scoped queries, minimizing flooding can be a significant benefit. Inspired by efficiency of pull for some applications, we revisited two-phase pull to eliminate one of its phases of flooding.

One-phase pull is a subscriber-based system that avoids one of the two phases of flooding present in two-phase pull. As with two-phase pull, subscribers send interest messages that disseminate through the network, establishing gradients. Unlike two-phase pull, when an interest arrives at a source it does not mark its first data message as exploratory, but instead sends data only on the preferred gradient. The preferred gradient is determined by the neighbor who was the first to send the matching interest, thus suggesting the lowest latency path. Thus one-phase pull does not require reinforcement messages, the lowest latency path is implicitly reinforced.

One-phase pull has two disadvantages compared to two-phase pull. First, it assumes symmetric communication between nodes since the data path (source-to-sink) is determined by lowest latency in the interest path (sink-to-source). Two-phase pull reduces the penalty of asymmetric communication since choice of data path is determined by lowest-latency exploratory messages, both in the source-to-sink direction. However, two-phase pull still requires some level of symmetry since reinforcement messages travel reverse links. Although link asymmetry is a serious problem in wireless networks, many other protocols require link symmetry, including 802.11 and protocols that use link-level acknowledgments. We assume that the MAC layer will allow diffusion to identify asymmetric links.

Second, one-phase pull requires interest messages to carry a flow-id. Although flow-id generation is relatively easy (uniqueness can be provided by MAC-level addresses or probabilistically with random assignment and periodic reassignment), this requirement makes interest size grow with number of sinks. By comparison, though, with two-phase pull the number of interest messages grows with proportion to the number of sinks, so the cost here is lower. Second, the use of end-to-end flow-ids means that one-phase pull does not use only local information to make data dissemination decisions.

2.5 Rendezvous approaches

Push and pull diffusion have active sources and sinks, respectively. When a scenario has groups involving a mixed number of sources and sinks, one would like a middle ground where each do some of the work. *Rendezvous* protocols share the effort by arranging for sources and sinks to meet in some predetermined way.

In the Internet multicast domain [11], distance-vector-based protocols such as DVMRP have actives sources, link-state-based protocols such as MOSPF have active sinks, and protocols such as PIM-SM adopt a rendezvous approach [9]. In sensor networks,

protocols such as GHT adopt a physically distributed rendezvous point for sensor data storage [25], while approaches such as rumor routing exploit physical properties (the intersection of two non-parallel lines) for a different kind of rendezvous [4].

Largely unexplored is the ability to accomplish rendezvous at the application-level, possibly exploiting application-specific constraints. Nested queries [15] represent one approach to this, as does application-level clustering (for one example, see [21]).

Because rendezvous approaches are quickly evolving we do not examine them more closely in this paper, but identify this class of protocols as an area for future work.

3. APPLICATION PERFORMANCE WITH DIFFERENT DIFFUSION ALGORITHMS

We have just described a series of diffusion algorithms that were designed in response to application needs. This section describes two applications developed or inspired by other researchers that benefit from push and GEAR, and it quantifies the performance gains in switching diffusion algorithms.

3.1 Push vs. two-phase pull diffusion

Our first application considers trade-offs in push against two-phase pull versions of diffusion. In two-phase pull, data sinks are active, sending out interests, while sources are passive until interests arrive. By contrast, with push, data sources are active, sending out data when it arrives. Push is designed for the case when there are many active sinks (listening for data), but relatively few nodes actually generating data. A common case of this kind of application is where many nodes are *cross-subscribed* to each other but mostly quiescent, all waiting for a triggering event to happen.

We explored this kind of application in the BAE sensor network testbed composed of 15 Sensoria WINSng 2.0 nodes. (These are 32-bit embedded computers with megabytes of memory and two independent, frequency-hopping radios that send data at about 20kb/s.) The application was inspired by applications at University of Wisconsin and PARC that employ cross-subscription. However, because those applications were not available to us at the time, we implemented a comparable application with a field of seven sensor nodes, all cross-subscribed to each other. When any one sensor changes state, all sensors send their readings to a triggered node that aggregates these readings and sends the aggregated result to the user. To control traffic, sensors were set to generate readings every 5s and to change state every minute.

Figure 1 shows a trace of communication rates across this experiment, where each point represents the number of packets sent over the last 30s. Two things stand out about this graph. First, the application's traffic is quite bursty. Second, push (the dotted line) is able to consistently out-perform two-phase pull (the solid line), transferring the same data with about 60% fewer messages.

Part of the saving in this experiment is because push is better suited to this application than two-phase pull. With many nodes cross-subscribed to each other, each will be frequently sending out interested messages to the network. With push, these interests are not sent; the only flooded messages are exploratory data.

If the sensors pushed relatively few detection events, the benefits of push would be greater still. In this case, data is sent every 5s from each sensor to the others and so sensors are not quiescent. We examine the effect of data generation rate in Section 4.5,

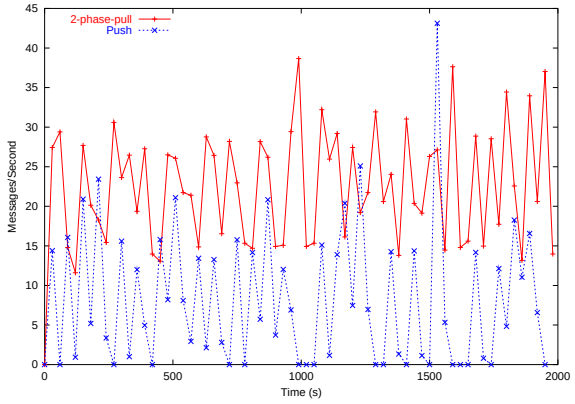


Figure 1: Push vs. two-phase pull diffusion with a cross-subscription application.

but as an area of future work we plan to redo this experiment in simulation to verify our intuition.

3.2 Geographic constraints

Researchers at Xerox PARC have suggested Information Driven Sensor Querying [7], an information-theoretic approach to sensor-net tracking. With their approach, one node (the leader) keeps track of the current target estimate. It periodically computes which other sensor can add the most information about the target location and then transfers leadership to that node through a process called *state transfer*. To keep system state consistent, leader election includes a suppression process where a leader informs other nodes not to become active, duplicate leaders themselves. Suppression messages are sent when the target is first detected and as it moves through the network. State transfer messages occur twice each second.

This application should benefit from push in the same way as the previous application (Section 3.1). In addition, suppression and state transfer are both geographically-scoped actions. To investigate the benefits of geographically scoped communications jointly with them we evaluated this application both with and without GEAR [30]. This application runs over 18 WINSng 2.0 nodes in a testbed with nodes from PARC and ISI. Sensor data in this case is generated by one or two human pulling a cart with pre-recorded acoustic data mimicking a large vehicle. The first simulated vehicle starts at 120s, the second at about 170s.

Figure 2 shows the message rates for this application. As can be seen, geographic scoping reduces message counts by 40%. This reduction is due to scoping of suppression messages. State transfer messages in this application are sent to a single point and so are also geographically directed, however this early implementation of push with GEAR did not support constraint of messages to a single point, only to regions, and so state transfer messages were flooded. We would expect a larger reduction in control overhead now that push with GEAR constrains control traffic directed to a point.

3.3 Discussion

These case studies illustrate the importance of matching the application to an appropriate data dissemination algorithm.

They also illustrate the complexity of selecting the best algo-

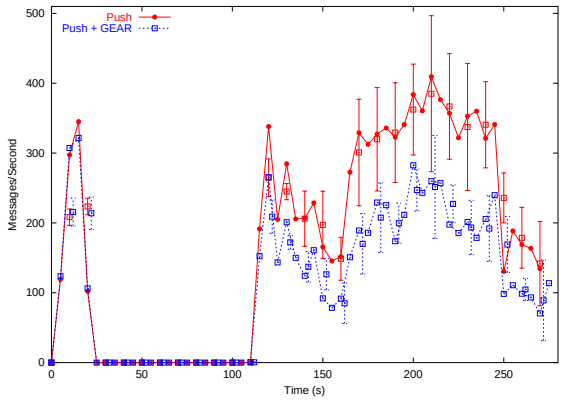


Figure 2: Push diffusion with and without GEAR over the IDSQ application.

rithm for a given application. Application designers are experts in their field, not networking, and so do not always have the best perspective to chose between several similar algorithms. The effects of selecting a diffusion algorithm can easily be masked by application errors. Our comparison of algorithms below is a first step to provide guidance to application designers, but an important area of future work is tools to help visualize and debug communication patterns in distributed, sensor-network applications.

To some extent it is a misstatement to suggest that there is a best algorithm for a single application. A sophisticated application like IDSQ has different patterns of communication in different parts of the application, and so requires *different* diffusion algorithms for different parts of the application. This supports our claim that a range of general and application-specific communication protocols are required for efficient data dissemination in sensor networks, both for different applications, and even in a single application.

A more specific result of these field studies concerns the appropriate means to select between algorithms. We had originally assumed that diffusion could infer the correct algorithm from the user's commands. For example, if geographic information was present, GEAR optimizations would be used. This approach proved too fragile for several reasons. First, it is prone to error. A misconfigured set of attributes can be syntactically correct but will not select the intended algorithm. The application will still run, but at greatly reduced performance. This problem is quite difficult to identify and correct, because performance of a distributed system can be difficult to measure, poor performance can be due to many causes, and the difference between correct code and incorrect is subtle. Second, as the number of alternative algorithms grow, it is no longer possible to distinguish between them automatically. Often the choice between algorithms depends on characteristics of the application known only to the programmer such as the communications patterns. A self-tuning system would be ideal, but collecting information for tuning requires communication itself and so will add its own overhead. For these reasons we now select algorithms explicitly as an attribute to publish and subscribe calls. We view the algorithm attribute as a programmer-provided assertion, much as annotations are used in distributed-shared-memory systems (for example, Munin [6]).

4. SYSTEMATIC COMPARISON

Data from previous experiments described in Section 3 suggested that algorithm choice can make a large difference in performance. However, the mechanics of doing field experiments are too cumbersome to allow a thorough comparison of algorithms. To provide guidance for future application designers, we used an emulation platform to characterize wide range of application and environmental configurations. After describing this methodology, the next several sections explore these key research questions: Do one-phase pull and push diffusion provide good performance to complementary applications? What implementation issues affect their relative performance? How does sending rate affect performance? What are the benefits of geographic optimizations?

4.1 Methodology

Although early field experiments motivated this work, we could not explore the range of scenarios needed to understand these research questions in that context. Instead we used the EmStar simulation/emulation environment [12] to quickly explore a range of configurations. EmStar provides an emulation environment where real application code runs on a single machine inside a simulated framework of communication devices and radio channels. Since the scenario is controlled by a simple configuration file, rather than physically moving nodes, we were able to explore many scenarios quickly.

Our experiments place 60 nodes randomly in a square area 50m on a side. Nodes do not move once placed. Because random topologies vary greatly in level of connectivity, we took two steps to reduce the effect of topology variation on the results. First, each experiment uses "paired" topologies. Each data point on a graph represents the mean of five simulation runs (95% confidence intervals are also shown). For any set of parameters, we vary one parameter and hold the topologies identical across the five runs. For example we generated 5 different topologies with different node locations for 1 source and 1 sink, then varied choice of push and one-phase pull in each topology to get the two left-most points on each line of Figure 4(a). Second, before using any topology, we check it for rough connectedness between all sources and sinks. Rough connectedness is defined as there being a path from every source to every sink that uses hops within partial radio range (12m). We discard any topologies that have sources or sinks that are not roughly connected.

Source and sinks are placed randomly in each topology. By default, both sources and sinks are *clustered* by being selected to be geographically close to each other. To select n clustered nodes we pick a point randomly and then pick the nodes closest to that point until n nodes have been selected. To ensure that sources and sinks are not intermixed, points for sinks are selected from the lower left sixteenth of the graph, and the central point for sources from the upper right sixteenth. For unclustered cases, we simply pick n nodes randomly for the parameter being varied (sources or sinks) while the other parameter (sink or source) is clustered as before. Figure 3 shows a sample clustered topology with five sources and sinks.

Each source generates traffic with an exponential inter-generation time with a fixed mean rate. In most cases we use an *equalized* traffic model, where we hold the aggregate event rate constant with a mean of one event every 2s over the entire network. Since the number of sources varies, we adjust each source's individual event rate to $\frac{1}{n}$th the total rate. We also use two other traffic models. In

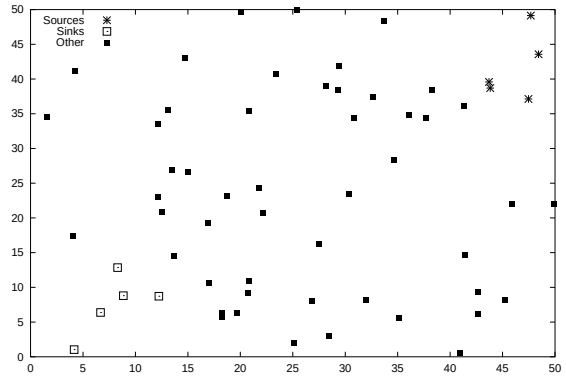


Figure 3: Sample evaluation topology with 5 sources and 5 sinks.

Figure 7 we use a *proportional* traffic model, where each source generates data at with a mean inter-generation time of 10s. With this model the number of events grows in proportion to the number of sources. Section 4.5 uses the proportional model but varies the mean. Although many sensor networks generate traffic at fixed intervals, we adopted an exponential inter-event time to avoid synchronization effects. (This approach may be useful in sensor-net applications that can tolerate non-uniform data sampling periods as well.) Table 1 shows the sizes of interests and data. For one-phase pull, data sizes grow with the number of sinks. Exploratory data messages are the same size as regular data messages. Push does not have interests, while one-phase pull does not have reinforcements. GEAR messages sizes are larger than non-GEAR messages to include location information.

Although prior work has emphasized the role of application-specific, in-network processing [17] we did not do aggregation or duplicate suppression in these experiments so that we could understand basic performance. Exploration of these aspects of diffusion is an area of future work.

We configured EmStar to use a very simple radio model. We assume 100% packet reception to a distance of 8m and then a linear fall-off of reception probability to 0% at 12m. On one hand, the lossy part of this model has a shallower fall-off than our testbed radios [29], and so this model presents a more challenging target than reality. On the other hand, this simple model does not exhibit complexities observed in real radios such as asymmetric links, interference, and strongly time-varying communication. We plan to explore the sensitivity of our experiments to radio model in future work, by using more challenging radio models and by communicating over real (but fixed location) radios.

An important metric of sensor network communication is the energy consumed. Previous work suggests that energy consumption from the radio will be a significant component of system energy consumption [24]. Unfortunately our emulated system does not currently model radio energy, so we approximate that by bytes received, normalized by the number of events generated. *[Note to reviewers: we expect to replace our bytes/event model with an energy/event model post-submission.]*

There are several parameters to the diffusion algorithms that affect performance. We held these constant in these experiments, with the interest interval set to 30s, the exploratory data interval at 90s.

There are several other parameters we either held fixed or varied in some experiments, including the number of sink nodes (ranging from 1 to 15), the number of sources, sensor nodes generating data (same range), and the data dissemination algorithm (typically either one-phase pull or push). Although by default both sources and sinks were geographically clustered, in some experiments we left one group unclustered.

4.2 Algorithm performance with different numbers of sources and sinks

Our first goal is to understand the differences between diffusion algorithms as the mix of senders and receivers varies. The designs of pull is optimized for a few sinks, while push is optimized for a few active sources. Field experiments (Section 3.1) have shown performance differences, but not carefully explored the trade-offs. Our first experiment therefore is designed to evaluate these algorithms experimentally over a wide range of parameters.

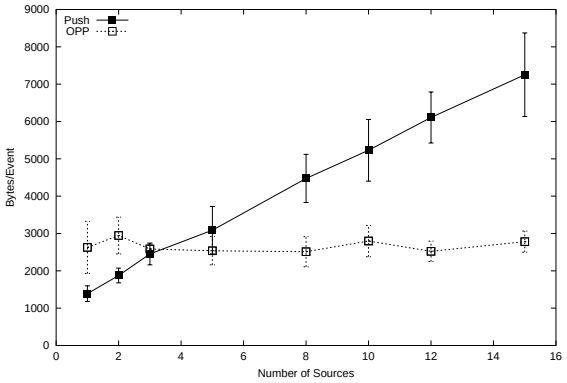
Figure 4 compares push and one-phase pull where we fix the number of sources at 1 and vary the number of sinks, and vice versa. Several observations can be made from these experiments. First, the shape of the graphs are consistent with our expectations. When we vary the number of sources (Figure 4(a)), the per-event overhead for one-phase pull remains constant, while the overhead for push grows linearly. Similarly, varying the number of sinks (Figure 4(b)) shows a linear increase in one-phase pull overhead and constant push overhead.

Because these are simple one-source/one-sink scenarios, we can also use them to evaluate the absolute cost of routing. Consider the one-source, one-sink data point on the left-most part of Figure 4(a). Push costs about 1500B/event and one-phase pull about 2500B/event. To put this into perspective, the basic message size is 85–95B (Table 1) and a typical distance across the network is 6 hops, so the minimum possible cost to handle an event, assuming an external, omniscient routing scheme, is about 540B/event. The observed control overheads are thus about 3–5$\times$ this minimum cost. This seems consistent to a comparison of data rates to control rates. For one-phase pull, interest messages are the primary control traffic, occurring at one-third the rate of message traffic (every 30s compared to every 10s). For push, control traffic is dominated by exploratory data which occurs every 90s. The costs of a flood are proportional to the size of the network. The costs of a flood of the entire network are about 10$\times$ the cost of sending a message from source to sink. Therefore we would approximate compute one-phase pull cost at about 2040B/event and push cost at about 1040B/event, slightly lower than we observe. This simple model gives us confidence that our simulations and understanding are consistent. It also demonstrates that the amount of control traffic is strongly dependent upon the data send rate, numbers of sources, and numbers of sinks. Varying control traffic is an area for future work; in the following sections we vary each of the other parameters.

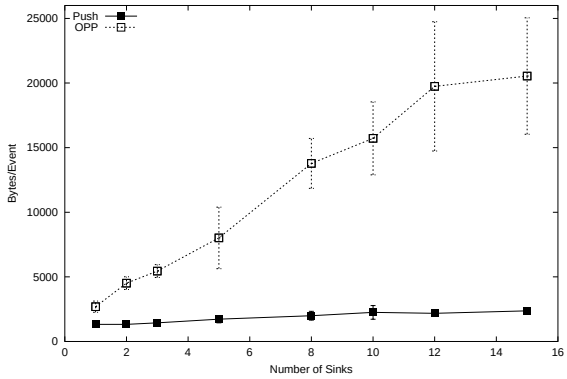
Although the trend clearly shows that one-phase pull dominates push performance as the number of sources grow, for small networks push performs better. This difference is due to the relative frequency of control traffic, with one-phase pull's interest messages occurring every 30s and push's exploratory data every 90s. Thus we expect them to show equal overhead when control traffic rates are equal at three sources, as occurs in Figure 4(a).

protocol	interest	data	reinforcements
push	—	84B	104B
one-phase pull	68B	92B + 4B/sink	—
push with GEAR	—	120B	140B
one-phase pull with GEAR	116B	108B + 4B/sink	—

Table 1: Sizes of control and data messages



(a) Varying sources (1 sink)



(b) Varying sinks (1 source)

Figure 4: Comparison of varying numbers of sources and sinks with push and one-phase-pull.

4.3 Varying the number of sinks with one-phase pull

Section 4.2 suggests that the relative performance of one-phase pull and push perform similarly for small numbers of sources and sinks. Since the amount of control traffic in one-phase pull is proportional to the number of sinks, this trade-off can easily shift even over small changes in configuration.

For example, Figure 5 compares the two algorithms with 5 sinks, a varying number of sources, and equalized loads. In this case, the constant cost of more frequent control traffic in one-phase pull leaves push more efficient over this traffic mix. As expected, overhead is on par at 15 sources. An experiment with 10 sinks provides a similar result: push outperforms one-phase pull, with parity projected at 30 sources.

Figure 6 systematically compares the overhead of one-phase pull as a function of the number of sources and sinks. As can be seen, overhead is roughly linear with the number of sinks (Figure 6(b)) independent of the number of sources (Figure 6(a)).

We normalize cost to the number of events, so we expect the the relative overhead of one-phase pull to strongly depend on aggregate event rates. Figure 7 shows the relative overhead as the number of sources grow with the proportional traffic and each source adds to the traffic load. Overhead drops with more sources because the fixed cost of one-phase pull control traffic can be amortized across a larger number of events.

4.4 Cost of adding more sinks with push

Section 4.3 suggests that constant factors make a large difference with one-phase pull. To understand if push has complementary performance, Figure 8 presents push performance as sinks and

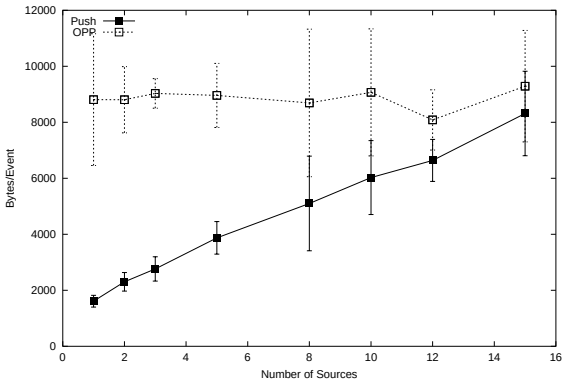
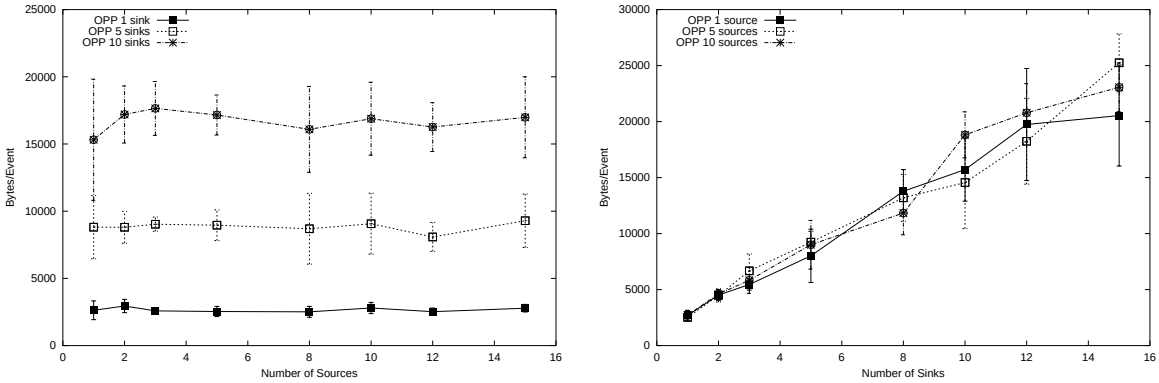


Figure 5: Comparison of push and one-phase pull at 5 sinks and varying numbers of sources.

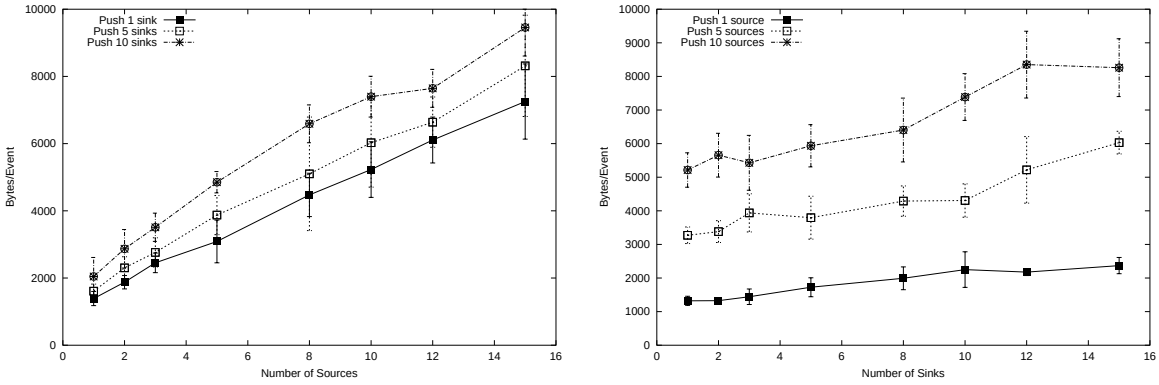
sources vary.

The primary overhead in push is due to exploratory data which must be sent throughout the network if geographic information is not available. As expected, Figure 8(a) therefore shows a linearly increasing cost with number of sources. We note that there is also a smaller linear trend as the number of sinks increase (Figure 8(b)). This is due to reinforcement messages which must travel from each sink back to sources. Reinforcements are not flooded, but they do show a noticeable cost proportional to the number of sinks, with slope proportional to the number of sources. For moderate-to-large networks, we expect that this cost will be dom-



(a) Varying sources (b) Varying sinks

Figure 6: Varying the number of sources and sinks with one-phase pull.



(a) Varying sources (b) Varying sinks

Figure 8: Varying the number of sources and sinks with push.

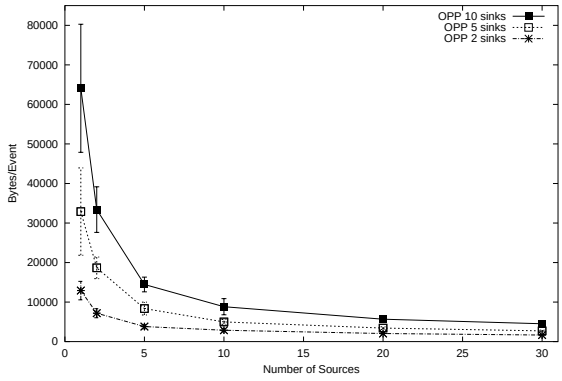


Figure 7: Comparison of one-phase pull with different numbers of sinks and sources.

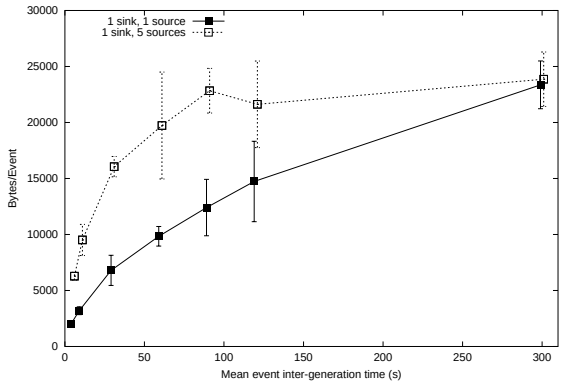


Figure 9: Varying the event generation rate with 1 sink and 1 or 5 sources with push.

inated by exploratory messages.

4.5 Varying traffic rates

We previously observed that the event rate can have a strong effect on relative overhead (in Section 4.3 and Figure 7).

Even with one source and sink there are interactions between the sending rate and diffusion algorithms. Figure 9 shows the relative per-event overhead for different event generation rates. The important relationship here is between event generation rate and exploratory data rate. Recall that with push diffusion, after each exploratory period, the next event is marked exploratory and sent throughout the network (if GEAR is not used). Other events are sent only on reinforced paths. Thus more frequent events distribute the cost of control overhead.

We can see this effect in Figure 9. The exploratory period is every 90s, so with 5 sources there is a clear knee in the curve at 90s. At this point all data messages are exploratory and flooded throughout the network; sending data slower than this rate incurs no additional overhead. Below this rate, overhead falls off as control overhead is spread across more useful data. This effect is less pronounced with only one source.

This analysis indicates that it is currently much more expensive to send data at low rates (less than the exploratory period) than at higher rates. This suggests several areas for future work. First, some applications may wish to control the frequency of control messages, allowing them to match control overhead to application needs. Second, the motivation for periodic control messages and soft-state is to deal with network dynamics. Unlike ad hoc networks, sensor networks nodes often are stationary and so only changes in radio connectivity affect connectivity. Thus sensor networks may benefit from much less frequent path computation than would be needed in a network with mobile nodes. A priori protocols analogous to DSDV or TORA may also have a larger role in sensor networks than in ad hoc networks. We expect that sensornet specific MAC-level protocols [26, 28] will play an increasing roll in ensuring that radio communications are fair and reliable even in the face of occasional channel noise.

4.6 Using geographic information

Exploiting geographic information to limit control overhead is an important optimization for sensornets where location is available.

To evaluate the benefit of GEAR and geographic scoping, Figure 10 compares push and one-phase pull with and without GEAR. Figure 10(a) examines push as the number of sources rise. Without geographic scope, push overhead rises steeply as the number of sources increase and each source generates more flooded exploratory traffic. With geographic scoping, control overhead is greatly reduced because control traffic is sent directly to the target region and need not be flooded. In fact, in absolute terms, control overhead of push with GEAR is lower than one-phase pull in Figure 4(a), although push-with-GEAR will still grow linearly with the number of sources. Figure 10(b) shows that one-phase pull benefits from GEAR in a similar way as the number of sinks rise.

Figure 10(a) also includes one additional experiment: the "push unclustered" represents the case where sources are not clustered but are randomly distributed across the entire network. Performance in this case is statistically equivalent to the clustered-source case. The reason for this result is that our experiments do not do in-network data aggregation. In systems that use aggregation or duplicate suppression (such as [17]), source clustering allows reduction in the amount of data returned to the user.

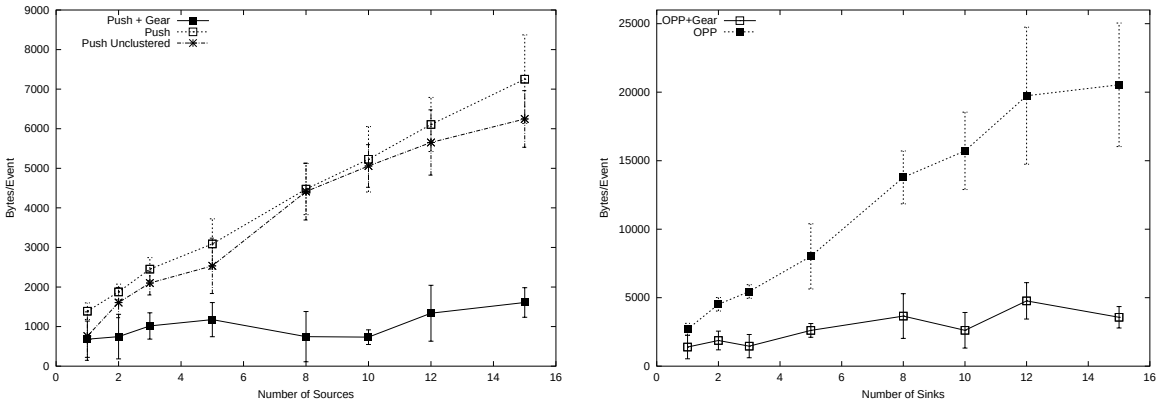
Performance with GEAR is quite good because both sources and sinks are clustered. If sinks were randomly distributed or were more widely distributed then push-with-GEAR performance would degrade to push-without-GEAR in proportion to the area of sink dispersal.

4.7 Summary of guidelines

To summarize our results:

- push works best with many sinks and few active sources
- one-phase pull works best with many sources and few sinks
- the break-even point between these algorithms depends upon control message frequency and data rates
- in networks with more than a few dozen nodes, the benefits of geographically scope queries outweigh other algorithmic choices
- sending at very low data rates (lower than the control traffic interval) has a relatively high control overhead, unless geographic scope or control rates are changed

5. RELATED WORK



(a) Varying sources (1 sink) with push

(b) Varying sinks (1 source) with one-phase pull

Figure 10: Push and one-phase pull performance with and without GEAR and geographically constrained control messages.

This paper builds upon prior work in evaluation of ad hoc routing protocols, multicast protocols, and sensor network routing protocols. We review each of these areas in turn.

Ad hoc routing protocols are related to data dissemination in sensor networks. A critical question in ad hoc routing has been choice between on-demand and a priori route computation. Broch et al. compare four ad hoc routing protocols and examine this question [5], while other researchers have proposed hybrid schemes merging the two approaches (for example, ZRP [14]. Barrett et al. instead provide a strong statistical comparison of the effects of interactions between ad hoc routing protocol, MAC protocol, and mobility model [2]. Like this prior work, we seek to put several routing algorithms into context, however, for sensor networks, the primary question is the interaction of application with source- or sink-driven routing algorithms. IP-based ad hoc routing protocols do not face this question at the IP-layer, since sinks there are always passive. The question may arise when resource discovery and applications are considered, but to our knowledge there are no studies of multi-hop, ad hoc networks consider resource discovery.

Multicast protocols in wired networks have similar trade-offs to data dissemination protocols that match sources to sinks in sensornets. Deering and Cheriton characterized multicast routing protocols with costs proportional to the numbers of sources and sinks [11], while later protocols such as PIM-SM adopted rendezvous approaches as a middle ground [10]. Unlike this prior work, diffusion can exploit application-specific information such as geographic information, and the use of diffusion in ad hoc networks presents quite different topologies and constraints. We explore these issues in this paper. Moreover, most multicast applications were both sources and sinks of data, so the variations in numbers of sources and sinks we explore have not been previously explored in the context of multicast.

A number of sensor-network-specific routing protocols have been proposed. Although some have been evaluated in absolute terms, or relative to optimal protocols, or to flooding, but comparisons of multiple protocols have been rare. Data-centric storage work has provided analysis showing when rendezvous-like protocols are important [25]. Analysis have compared address-centric and data-centric protocols [19], and more recent analytic work has begun to explore trade-offs with different diffusion algorithms [20]. To our knowledge, our work is the first to suggest that sensor network applications will be matched to the data dissemination protocol, and thus no *one* protocol will be appropriate for all situations. Our work complements the analysis in [20] by considering implementation constraints and effects (such as those in Section 4.3).

6. FUTURE WORK

We have touched upon future work several times in this paper. We highlight several directions we are planning to pursue here.

We plan to review the results of this paper in light of explicit energy consumption models, and with different kinds of radio model.

Addition of a rendezvous-based approach to push and one-phase pull may provide good performance for cases with mixed numbers of sources and sinks. An important question though is understanding the system costs at defining and meeting at a rendezvous point.

Also important are the impact of application-specific in-network processing. Previous research has found in-network processing techniques such as duplicate suppression central to good performance [17]. We plan to explore the impact of in-network processing on these results, and the use of nested queries as a rendezvous mechanism.

A second direction to explore is the optimization of data dissemination for relatively static networks. One approach would be to automatically vary control traffic rates relative to network stability.

Section 4 has considered only simple applications that send directly too each other. More complicated applications, such those employing nested or clustered computation deserve more analysis since they make exhibit different kinds of spatial locality.

Finally, there is a strong need for better performance debugging tools in distributed sensor networks.

7. CONCLUSIONS

In our work applying diffusion in real applications we found

that a single algorithm performed well with some applications but poorly with others. Exploiting the abstract, publish/subscribe nature of the diffusion API, we developed two new algorithms that work with existing diffusion applications: push, and one-phase pull. We demonstrated that these can offer significant performance benefits to some applications, but found that simple descriptions of the algorithms did not offer sufficient guidance to their use in new applications. To address this problem we systematically explored performance of these algorithms over a different numbers of sinks, sources, with and without geographic optimizations, and at with different traffic placement and rates. We demonstrated that push and one-phase pull can serve complementary applications, but that implementation details such as the rate of control messages must be considered when evaluating algorithm tradeoff. While our focus has been specific to diffusion, the need for a family of protocols applies to all general-purpose sensor-network communications frameworks.

Acknowledgments

We would like to thank Joseph Reynolds, Steve Beck, and Carol Brewer (BAE Austin) for their support running experiments remotely on their testbed. James Reich and Julia Liu (PARC) devleoped IDSQ, worked with us to improve its network performance, and took IDSQ data in a joint testbed composed of PARC and ISI nodes. We would like to thank Bhaskar Krishnamachari for discussions about alternatives and analysis of diffusion alternatives. Ramesh Govindan (USC) also contributed to the design of one-phase pull diffusion. Finally, we would like to thank Jeremy Elson (UCLA/CENS) for his help with Emstar.

8. REFERENCES

[1] Deepak Ganesan and Ramesh Govindan, Scott Shenker, and Deborah Estrin. Highly resilient, energy efficient multipath routing in wireless sensor networks. In *ACM International Symposium on Mobile Ad Hoc Networking and Computing (poster)*, Long Beach, California, USA, October 2001. ACM.

[2] Chris Barrett, Martin Drozda, Achla Marathe, and Madhav V. Marathe. Characterizing the interaction between routing and mac protocols in ad-hoc networks. In *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 92–103, Lausanne, Switzerland, June 2003. ACM.

[3] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Querying the physical world. *IEEE Personal Communications Magazine*, 7(5):10–15, October 2000.

[4] David Braginsky and Deborah Estrin. Rumor routing algorithm for sensor networks. In *Proceedings of the First ACM Workshop on Sensor Networks and Applications*, pages 22–31, Atlanta, GA, USA, October 2002. ACM.

[5] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparision of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 85–97, Dallas, Texas, USA, October 1998. ACM.

[6] John B. Carter, John K. Bennett, and Willy Zwaenepoel. Implementation and performance of Munin. In *Proceedings of the Thirteenth Symposium on Operating Systems Principles*, pages 152–164. ACM, October 1991.

[7] Maurice Chu, Horst Haussecker, and Feng Zhao. Scalable information-dirven sensor querying and routing for ad hoc hetereogeneous sensor networks. Technical Report P2001-10113, XEROX Palo Alto Research Center, May 2001.

[8] Dan Coffin, Dan 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.

[9] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei. An architecture for wide-area multicast routing. In *Proceedings of the ACM SIGCOMM Conference*, pages 126–135, University College London, London, U.K., September 1994. ACM.

[10] Stephen Deering, Deborah L. Estrin, Dino Farinacci, Van Jacobson, Ching-Gung Liu, and Liming Wei. The PIM architecture for wide-area multicast routing. *ACM/IEEE Transactions on Networking*, 4(2):153–162, April 1996.

[11] Stephen E. Deering. Multicast routing in internetworks and extended LANs. In *Proceedings of the ACM SIGCOMM Conference*, pages 55–64, Stanford, CA, August 1988. ACM.

[12] J. Elson, S. Bien, N. Busek, V. Bychkovskiy, A. Cerpa, D. Ganesan, L. Girod, B. Greenstein, T. Schoellhammer, T. Stathopoulos, and D. Estrin. EmStar: An environment for developing wireless embedded systems software. Technical Report CENS-TR-9, University of California, Los Angeles, Center for Embedded Networked Computing, March 2003. (Also submitted to SOSP-19).

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

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

[15] 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 Symposium on Operating Systems Principles*, pages 146–159, Chateau Lake Louise, Banff, Alberta, Canada, October 2001. ACM.

[16] John Heidemann, Fabio Silva, Yan Yu, Deborah Estrin, and Padmaparma Haldar. Diffusion filters as a flexible architecture for event notification in wireless sensor networks. Technical Report ISI-TR-556, USC/Information Sciences Institute, April 2002.

[17] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 56–67, Boston, MA, USA, August 2000. ACM.

[18] Brad Karp and H. T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 243–254, Boston, Mass., USA, August 2000. ACM.

[19] Bhaskar Krishnamachari, Deborah Estrin, and Stephen Wicker. The impact of data aggregation in wireless sensor networks. In *Proceedings of the IEEE International Workshop on Distributed Event-Based Systems (DEBS)*, pages 575–578, Vienna, Austria, July 2002. IEEE.

[20] Bhaskar Krishnamachari and John Heidemann. Application-specific modelling of information routing in sensor networks. under submission, April 2003.

[21] Satish Kumar, Cengiz Alaettinoglu, and Deborah Estrin. SCalable Object-tracking through Unattended Techniques (SCOUT). In *Proceedings of the IEEE International Conference on Network Protocols*, pages 253–262, Osaka, Japan, November 2000. IEEE.

[22] Juan Liu, Jie Liu, James Reich, Patrick Cheung, and Feng Zhao. Distributed group management for track initiaition and maintenance in target localization applications. In *Proceedings of the IEEE International Workshop on Information Processing in Sensor Networks*, page to appear, Palo Alto, California, USA, April 2003. IEEE.

[23] Samuel Madden, Michael J. Franklin, Joseph Hellerstein, and Wei Hong. TAG: Tiny AGgregate queries in ad-hoc sensor networks. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, Boston, Massachusetts, USA,

December 2002. USENIX.

[24] Gregory J. Pottie and William J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, May 2000.

[25] Sylvia Ratnasamy, Brad Karp, Li Yin, Fang Yu, Deborah Estrin, Ramesh Govindan, and Scott Shenker. GHT: A geographic hash table for data-centric storage. In *Proceedings of the ACM Workshop on Sensor Networks and Applications*, pages 78–87, Atlanta, Georgia, USA, September 2002. ACM.

[26] Alec Woo and David Culler. A transmission control scheme for media access in sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 221–235, Rome, Italy, July 2001. ACM.

[27] Kung Yao, Ralph E. Hudson, Chris W. Reed, Daching Chen, and Flavio Lorenzelli. Blind beamforming on a randomly distributed sensor array system. *IEEE Journal of Selected Areas in Communication*, 16(8):1555–1567, October 1998.

[28] Wei Ye, John Heidemann, and Deborah Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of the IEEE Infocom*, pages 1567–1576, New York, NY, USA, June 2002. USC/Information Sciences Institute, IEEE.

[29] Wei Ye, John Heidemann, and Deborah Estrin. A flexible and reliable radio communication stack on motes. Technical Report ISI-TR-565, USC/Information Sciences Institute, September 2002.

[30] Yan Yu, Ramesh Govindan, and Deborah Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. Technical Report TR-01-0023, University of California, Los Angeles, Computer Science Department, 2001.