Directed Diffusion *

Fabio Silva[†] John Heidemann[†]
Ramesh Govindan[†‡] Deborah Estrin[†¶]

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

‡ *Computer Science Department*
University of Southern California
Los Angeles, CA, USA 90089

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

{fabio,johnh,govindan,estrin}@isi.edu

February 10, 2004

1 Introduction

Traditional sensing models assume one or a few powerful sensors and centralized computation. Today, technological trends enable the creation of inexpensive, small, intelligent devices for sensing and actuation. If many small sensors can work together as a *sensor network*, they provide several advantages over traditional centralized sensing. By placing the sensor close to the object being sensed, signal processing and target discrimination problems in sensing can be greatly simplified. By communicating over several short hops rather than one long hop, energy consumed in communication can be reduced [29]. Moreover, by processing data *in* the network, often the amount of data transfered can be reduced, further saving energy [22].

Motivated by robustness, scaling, and energy efficiency requirements, this paper examines a new data dissemination paradigm for such sensor networks. This paradigm, which we call *directed diffusion*[1], is data-centric. Data generated by sensor nodes is named by attribute-value pairs. A node requests data by sending *interests* for named data. Data matching the interest is then "drawn" down towards that node. Intermediate nodes can cache, or transform data, and may direct interests based on previously cached data (Section 3).

Directed diffusion is significantly different from IP-style communication where nodes are identified by their end-points, and inter-node communication is layered on an end-to-end delivery service provided within the network. In directed diffusion, nodes in the network are application-aware as we allow application-specific code to run in the network and assist diffusion in processing messages. This allows directed diffusion to cache and process data in the network (aggregation), decreasing the amount of end-to-end traffic, and resulting in higher energy savings. We show that using directed diffusion one can realize robust multi-path delivery, empirically adapt to a small subset of network paths, and achieve significant energy savings when intermediate nodes aggregate responses to queries (Section 5).

*__USC/ISI Technical Report ISI-TR-2004-586.__ This work was supported by DARPA under grant DABT63-99-1-0011 as part of the SCAADS project.

[1] Van Jacobson suggested the concept of "diffusing" attribute named data for this class of applications that later led to the design of directed diffusion.

Publish/Subscribe APIs:

```
handle NR::subscribe(NRAttrVec *subscribe_attrs, const NR::Callback * cb);
int NR::unsubscribe(handle subscription_handle);
handle NR::publish(NRAttrVec *publish_attrs);
int NR::unpublish(handle publication_handle);
int NR::send(handle publication_handle,
             NRAttrVec *send_attrs);
```

Filter-specific APIs:

```
handle NR::addFilter(NRAttrVec *filter_attrs, u_int16_t priority, FilterCallback *cb);
int NR::removeFilter(handle filter_handle);
void NR::sendMessage(Message *msg, handle h, u_int16_t priority = 0);
```

Figure 1: Basic diffusion APIs for sending and receiving data, and for adding filters.

This chapter describes diffusion, starting from the point of view of an application (Section 2) and naming (Section 2.2). We realize these abstractions with lower-level primitives and several different data dissemination algorithms described in Section 3, and show how applications can influence routing (Section 4). We summarize simulation and experimentation results in Section 5.

2 Programming a Sensor Network

The innovations of diffusion are approaches to allow applications to process data as it moves through the network, and dissemination algorithms that select efficient paths through the network. Although these topics are important and we explore them in the following chapter, applications require abstractions over these details. This section presents an *application-level* view of diffusion, looking at our publish/subscribe-based API and how applications name data in the network.

2.1 The Publish/Subscribe API

We have adopted a publish/subscribe-based API for diffusion, shown in Figure 1[2]. To receive data, users or programs *subscribe* to a particular set of attributes, becoming data *sinks*. A callback function is then invoked whenever relevant data arrives at the node. Sensors *publish* data that they have, becoming data *sources*. In both cases, what data is provided or received is described by an attribute-based naming scheme described in Section 2.2. It is the job of the diffusion dissemination algorithms (Section 3) to ensure that data is communicated efficiently from sources to sinks across a multi-hop network. In general, publishing and subscribing sends messages across the network. The exact cost of these operations depends on which diffusion algorithm is used.

To allow applications to influence data as it moves through the network, users can create *filters* at each sensor node with

[2]This API was originally designed in collaboration with Dan Coffin and Dan van Hook [14]; we have since extended it to support filters.

the filter APIs in the bottom of Figure 1. Filters indicate what messages they are interested in by attributes; each time a matching message arrives at that node the filter is allowed to inspect and alter its progress in any way. Filters can suppress messages, change where they are sent next, or even send other messages in response to one (perhaps triggering further sensors to satisfy a query).

A more complete reference to directed diffusion APIs and example code is available in the diffusion manual [30].

2.2 Naming Concepts

Diffusion uses an attribute-based naming scheme to associate sources and sinks and to trigger filters. This flexible approach to naming is important in several ways. First, attribute-based naming is consistent with the publish/subscribe application-level interface (Section 2) and many-to-many communication. Diffusion's naming scheme is *data-centric*, allowing applications to focus on what data is desired rather than on individual sensor nodes. The approach also supports multiple sources and sinks, rather than simple point-to-point communication. Thus applications may subscribe to "seismic sensors in the southeast region" rather seismic sensors #15 and #35, or hosts 10.1.2.40 and 10.2.1.88.

Second, diffusion attributes provide some structure to a message. By identifying separate fields, data dissemination algorithms can use application data to influence routing. For example, application-specific, geographic information can limit where diffusion must look for sensors. In addition, treating messages as sets of attributes simplifies application and protocol extensions (a need also suggested for future Internet-based protocols [7]).

Finally, attributes serve to associate messages with sources, sinks, and filters via *matching*. If the attributes in a sink's subscription match those of source's publication, diffusion must send any published data to the sink.

2.3 Matching in Naming

Each set of attributes in diffusion is a set of (key, type, operator, value) tuples. The most important parts of an attribute are the *key* and *value*, which together specify the meaning of the data (longitude, temperature, detection confidence, etc.) and its actual contents (118.40817 degrees, 98.6 degrees, 80%, etc.) The *type* defines how the value field is interpreted: as a string, integer or floating point type, or as uninterpreted binary data (blobs).

The operator field allows attributes to not only contain data, but to express simple constraints. There are two classes of operators: first, IS, the *actual* operator, is used to indicate a specific value. The second group includes binary comparisons (EQ, NE, LE, GT, LE, GE, corresponding to equality, inequality, less than, etc.) and "EQ_ANY" (which matches anything); these are collectively called *formal* operators.

Actuals are statements about data. So "latitude IS 33.9425, longitude IS 118.40817" might indicate a location, or "sensor IS seismic, value IS 7.0, confidence IS 80" might indicate a specific sensor reading.

Formals allow one to select sets of sensors, thus indicating which publish and subscribe operations should be connected. Thus, a subscription might indicate "latitude GT 33.5, latitude LT 34.0, sensor EQ seismic" to indicate seismic sensors in some area.

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 2: Our one-way matching algorithm.

Formals and actuals can be mixed and used in publications, subscriptions, or filters.

The exact process of determining which publications and subscriptions are related is called matching. A *one-way match* compares all formal parameters of one attribute set against the actuals of the others (Figure 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*.

Matching is used to associate publications and subscriptions and to activate filters as messages flow through the network.

Although matching is reasonably powerful, it does not perfectly cover all scenarios or tasks. Matching strikes a balance between ease of implementation and flexibility. For example, while attributes can easily define a square, they cannot directly operate on arbitrarily complex sensor detection regions. We expect applications to use attributes for rough matching and refine matching with application-specific code (such as with filters, Section 4).

For detailed examples of naming in diffusion, please see the diffusion manual [30] or [18].

3 Directed Diffusion Protocol Family

Publish/subscribe provides an application's view to a sensor network, and attribute-based naming a detailed way to specify which sources and sinks communicate. The "glue" that binds the two are the directed diffusion algorithms for data dissemination. In a traditional network, communication is effected by routing, usually based on global addresses and routing metrics. Instead, we use the term data dissemination to emphasize the lack of global addresses, reliance on local rules, and, as described in the Section 4, the use of application-specific in-network processing.

The original, two-phase directed diffusion uses several control messages to realize our publish/subscribe API: sinks send *interest* messages to find sources, sources use *exploratory data* messages to find sources, and positive and negative

protocol	sink	source
two-phase pull	interest* (every interest interval)	
		exploratory data* (every exploratory interval)
	positive reinforcement (response to exp. data)	
		data (rate defined by app.)
one-phase pull	interest* (every interest interval)	
		data
push		exploratory data* (every exploratory interval)
	positive reinforcement (response to exp. data)	
		data

Table 1: Comparison of interactions in diffusion algorithms. Asterisks (*) indicate messages that are sent to all nodes (flooded or geographically scoped). All algorithms also have negative reinforcement messages.

reinforcement messages select or prune parts of the path. Early work [22] identified these primitives, described the concept of diffusion, and evaluated a specific algorithm that we now call *two-phase pull* diffusion. We found this algorithm ideal for some applications but as our experience with sensor networks applications grew, we found two-phase pull a poor match for other classes of applications.

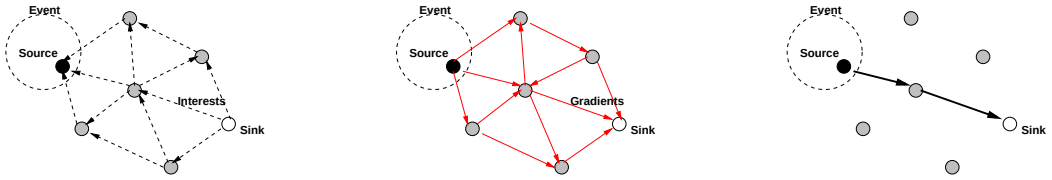
We see diffusion not as a single algorithm, but as a *family* of algorithms built from these primitives. Other algorithms provide better performance for some applications. We have recently made two additions to the diffusion protocol family: *one-phase push* and *one-phase pull* [17].

Another way to optimize diffusion performance is to use physical or application-specific information. The physical nature of a sensor network's deployment makes geographically scoped queries natural, prompting the development of geographically-aided routing protocols such as *GEAR* [34], *GPSR* [26], and *rumor routing* [8]. Application-specific information can also be exploited using filters (described in Section 4).

We expect application designers to match an appropriate algorithm with their application's requirements. Table 1 compares the interactions of the algorithms; we describe them below in more detail and review their performance in Section 5.4. More detail is the subject of current [22, 24, 17] and ongoing research.

3.1 Two-Phase Pull Diffusion

The purpose of directed diffusion is to establish efficient *n*-way communication between one or more sources and sinks. Directed diffusion is a data-centric communication paradigm that is quite different from host-based communication in traditional networks. To describe the elements of diffusion, we take the simple example of a sensor network designed for tracking animals in a wilderness refuge.



(a) Interest propagation

(b) Initial gradients set up

(c) Data delivery along reinforced path

Figure 3: A simplified schematic for directed diffusion.

Suppose that a user in this network would like to track the movement of animals in some remote sub-region of the park. The user would subscribe to "animal-track" information, specified by a set of attributes. Sensors across the network publish animal-track information.

The user's application subscribes to data using a list of attribute-value pairs that describe a task using some task-specific naming scheme. Intuitively, attributes describe the data that is desired by specifying sensor types and possibly some geographic region. The user's node becomes a *sink*, creating an *interest* of attributes specifying a particular kind of data.

The interest is propagated from neighbor-to-neighbor towards sensor nodes in the specified region. A key feature of directed diffusion is *that every sensor node can be task-aware*—by this we mean that nodes store and interpret interests, rather than simply forwarding them along. In our example, each sensor node that receives an interest remembers which neighbor or neighbors sent it that interest. To each such neighbor, it sets up a *gradient*. A gradient represents both the direction towards which data matching an interest flows, and the status of that demand (whether it is active or inactive and possibly the desired update rate). After setting up a gradient, the sensor node redistributes the interest to its neighbors. When the node can infer where potential sources might be (for example, from geographic information or existing similar gradients), the interest can be forwarded to a subset of neighbors. Otherwise, it will simply broadcast the interest to all of its neighbors.

Sensors indicate what data they may generate by publishing with an appropriate set of attributes. They thus become potential *sources*. As interests travel across the network, sensors with matching publications are triggered and the application activates its local sensors to begin collecting data. (Prior to activation we expect the node's sensors would be in a low-power mode). The sensor node then generates *data* messages matching the interest. In directed diffusion, data is also represented using an attribute-based naming scheme.

Data is cached at intermediate nodes as it propagates toward sinks. Cached data is used for several purposes at different levels of diffusion. The core diffusion mechanism uses the cache to suppress duplicate messages and prevent loops, and it can be used to preferentially forward interests. (Since the filter core is primarily interested in an exact match, as an optimization, hashes of attributes can be computed and compared rather than complete data.) Cached data is also used for application-specific, in-network processing. For example, data from detections of a single object by different sensors may be merged to a single response based on sensor-specific criteria.

The initial data message from the source is marked as *exploratory* and is sent to all neighbors for which it has matching gradients. The initial flooding of the interest, together with the flooding of the exploratory data, constitutes the first phase of two-phase pull diffusion. If the sink has multiple neighbors, it chooses to receive subsequent data messages for the same interest from a preferred neighbor (for example, the one which delivered the first copy of the data message). To do this, the sink *reinforces* the preferred neighbor, which, in turn reinforces its preferred upstream neighbor, and so on. The sink may also *negatively reinforce* its current preferred neighbor if another neighbor delivers better (lower latency) sensor data. This negative reinforcement propagates neighbor-to-neighbor, removing gradients and tearing down and existing path if it is no longer needed [22]. Negative reinforcements suppress loops or duplicate paths that may arise due to changes in network topology.

After the initial exploratory data message, subsequent messages are sent only on reinforced paths. (The path reinforcement, and the subsequent transmission of data along reinforced paths, constitutes the second phase of two-phase pull diffusion). Periodically the source sends additional exploratory data messages to adjust gradients in the case of network changes (due to node failure, energy depletion, or mobility), temporary network partitions, or to recover from lost exploratory messages. Recovery from data loss is currently left to the application. While simple applications with transient data (such as sensors that report their state periodically) need no additional recovery mechanism, we are also developing retransmission scheme for applications that transfer large, persistent data objects [31].

This simplified description points out several key features of diffusion, and how it differs from traditional networking. First, diffusion is data-centric; all communication in a diffusion-based sensor network uses interests to specify named data. Second, all communication in diffusion is neighbor-to-neighbor or hop-by-hop, unlike traditional data networks with end-to-end communication. Every node is an “end” in a sensor network. A corollary to this previous observation is that there are no “routers” in a sensor network. Each sensor node can interpret data and interest messages. This design choice is justified by the task-specificity of sensor networks. Sensor networks are not general-purpose communication networks. Third, nodes do not need to have globally unique identifiers or globally unique addresses for regular operation. Nodes, however, do need to distinguish between neighbors. Fourth, because individual nodes can cache, aggregate, and more generally, process messages, it is possible to perform coordinated sensing close to the sensed phenomena. It is also possible to perform in-network data reduction, thereby resulting in significant energy savings. Finally, although our example describes a particular usage of the directed diffusion paradigm (a query-response type usage, see Figure 3), the paradigm itself is more general than that; we discuss several other usages next.

3.2 Push Diffusion

Two-phase pull diffusion 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. Push diffusion was motivated by applications such as these being developed 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.

One-phase push diffusion (or just push diffusion) was designed for this application. Although the API is the same as two-phase pull 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. Section 5.4.1 presents a performance comparison of push and two-phase pull diffusion for such an application.

3.3 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, and 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. As such, it is reasonable to assume that detecting and filtering such links will be done at the MAC layer, allowing one-phase diffusion to work.

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.

3.4 Using Geographic Cues to Limit Flooding

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 [34]. GEAR is an extension to existing diffusion algorithms that replaces network-wide communication with geographically constrained communication. When added to one-phase or 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.

For one-phase push diffusion, GEAR uses the same mechanism to send exploratory data messages containing a destination region towards that region. This avoids flooding by allowing data senders to push their information only to subscribers within the desired region, which in turn will send reinforcements resulting in future data messages following a single path to the subscriber. In Section 5.4.2, we present a field experiment showing a performance comparison of push diffusion with and without GEAR using the PARC IDSQ application.

We have also implemented GPSR [26] in the filter framework as an alternative to GEAR.

4 Facilitating In-Network Processing

Filters are our mechanism for allowing application-specific code to run in the network and assist diffusion and processing. Applications provide filters before deployment of a sensor network, or in principle filters could be distributed as mobile code packages at run-time. Filters register what kinds of data they handle through matching; they are then triggered each time that kind of data enters the node. When invoked, a filter can arbitrarily manipulate the message, caching data, influencing how or where it is sent onward, or generating new messages in response. Uses of filters include routing, in-network aggregation, collaborative signal processing, caching, and similar tasks that benefit from control over data movement, as well as debugging and monitoring.

Filters use only one-way matching. A message entering a node triggers a filter if the attributes specified by the filter match the attributes in the message, but it does not require matching in the other direction. This approach allows filters to process data more generally with the publish-subscribe API.

The filter core is the system component responsible for interconnecting all hardware devices, applications, and filters. Even though logically messages pass from filter to filter, in practice, all messages pass through the filter core, which shepherds messages from filter to filter, according to filter priorities.

Priorities, defined at filter configuration, give a total ordering of all filters in a system. While message attributes select which filters can process a message, priorities specify the *order* in which those filters act.

Priorities are needed because the attributes of an incoming message may match multiple filters. In this case, filter priorities indicate which filter is invoked first. As described later in Section 2, once a filter receives a message, it has total control over where the message will go next. A filter can pass the message to the next filter, modify the message and then send it, suppress it, generate messages in response of it, etc. Filters can also use the filter API to override the order of message processing by changing the priority field and/or messages attributes. Thus a knowledgeable filter can direct a message anywhere in the diffusion stack. Since the contents or priority can change any time a message leaves a filter, all messages are always sent to the filter core, not immediately to the next filter.

4.1 Implemented filters

In this section we describe the set of filters that we have implemented and designed. As shown in figure 4, the filter core interacts with all filters (rectangles), applications (circles at the top right), and radio hardware (the lozenge at the bottom). Solid and dashed rectangles represent existing and planned filters, respectively. The core is responsible for dispatching all messages as they pass through the system and for suppressing duplicate messages.

Basic diffusion is implemented in the *two-phase pull* filter. This filter maintains gradients representing the state of any existing flows to all neighbors and is responsible for forwarding data messages using reinforced paths, in addition to periodically send out reinforcement messages and interests.

GEAR is a pair of filters that can optionally surround the two-phase pull filter to implement Geographic and Energy-Aware Routing [34]. Lacking prior information (such as geographic information or prior saved state), basic diffusion floods interests to all nodes in the network. GEAR overrides this behavior to forward messages with geographic assistance (interests are sent basically toward their geographic destination, but around any holes in the topology). GEAR consists of two filters, a *pre-processing* filter that sits above the two-phase pull filter to handle GEAR-specific beacon messages and to remove transient geographic information on arrival, and a *geographic routing* filter that acts after the two-phase pull filter to forward interests in a good direction.

Ben Greenstein and Xi Wang have each implemented versions of *GPSR* [26] as filters. GPSR, like GEAR, uses geographic information to make informed neighbor selection when forwarding packets. One implementation was done as an extension of diffusion (as described above), another as a stand-alone routing module (independent of diffusion).

Reliable Multi-Segment Transport (RMST) is a module that allows reliable transfers of large (multi-packet), uninterpreted data across unreliable links [31]. RMST is being used to investigate the trade-offs among MAC, transport, and application reliability. As a filter, it has two interesting characteristics. First, it caches data locally to support loss recovery, similar to approaches taken in reliable multicast [15] and SNOOP TCP [3], but at all hops rather than at the end-points or at base-

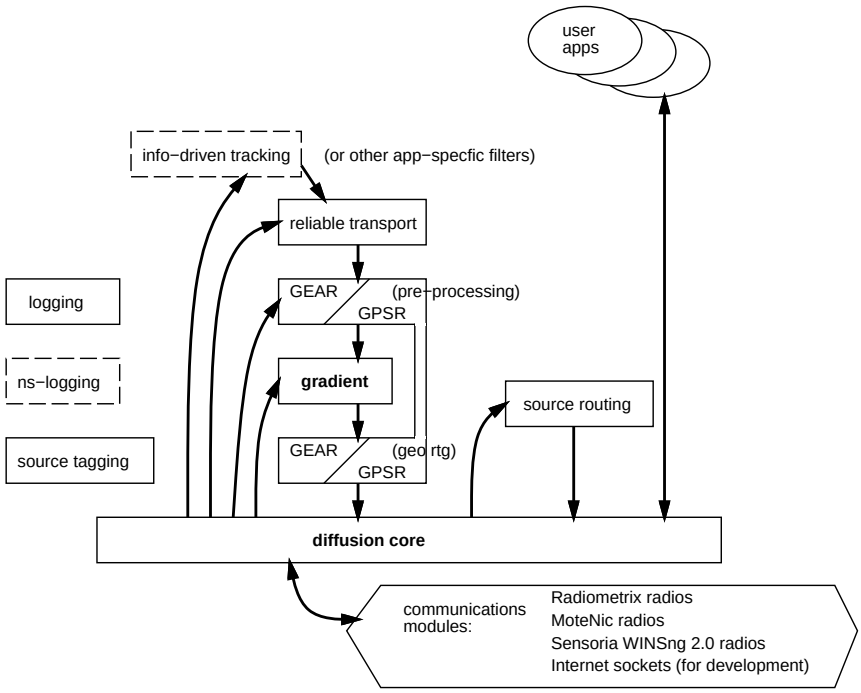


Figure 4: Current and planned filters in diffusion and how they interact.

stations only. Second, it implements a *back channel*, the reverse of the reinforced path created by the gradient filter. This back channel is used to propagate negative acknowledgment messages from the receiver to the sender.

The *information-driven tracking filter* is an example of how application-specific information can assist routing, proposed by researchers at Xerox PARC [35]. An important application of sensor networks is object tracking—multiple sensors may collaborate to identify one or more vehicles, estimating their position and velocity. Which sensors collaborate in this case is dependent on the direction of vehicle movement. They have proposed using current vehicle estimates (or "belief state") to involve the relevant sensors in this collaboration while allowing other sensors to remain inactive (conserving network bandwidth and battery power). While GEAR uses generic (geographic) information to reduce unnecessary communication, the information-driven tracking filter uses application-specific information to further reduce communication. As other applications are explored we expect to develop other application-specific filters similar to the information-driving tracking filter.

One use of filters is logging information for debugging. We have implemented a *logging* filter for this purpose, and we are considering implementing an *ns-logging* filter for simulator-specific logging. These filters are shown to the left of diffusion stack because they can be placed between any two modules.

Although this architecture was built to explore diffusion-style routing, for debugging purposes we also developed support for source routing. Source routing is provided as two filters: the *source tagging* filter functions similar to the logging filters in that it can be configured anywhere in the diffusion stack. This filter adds a record of each node that the message

passes through, much like the `traceroute` command used on the Internet. The *source routing* filter provides the opposite function. It takes a message that includes an attribute listing the path of nodes the message should take through the network and dispatches it along that path. A design principle of directed diffusion is local operation—nodes should not need to know information about neighbors multiple hops away. While source routing is directly opposite to this goal, it can be provided within our software framework, and is still sometimes a useful debugging tool.

5 Evaluation

In this section we evaluate diffusion using simulations, and real-life experiments. We start by presenting simulation results showing the performance impact of diffusion. Then, we describe nested queries, a new query model that reduces end-to-end traffic by doing application-level aggregation using diffusion's in-network processing capabilities. Later, we show examples where application performance is highly affected by choosing the best diffusion algorithm.

5.1 Implementation experience

Several implementations of diffusion have existed in simulation and on several hardware platforms. Diffusion was first implemented by Chalermek Intanagonwiwat in simulation with ns-2 [22]. The first implementations for native hardware were for Linux/x86 (desktop computers) and WINSng 1.0 sensor nodes running Windows CE (Figure 5(c)). More recent implementations added support for filters, PC/104 hardware with several kinds of radios (Figure 5(a)), and WINSng 2.0 nodes (Figure 5(d)) based on the SH-4 processor. The most recent release (3.2 as of this writing) includes nearly source-compatible support for the ns-2 simulator.

Researchers at UCLA have implemented *Tiny Diffusion*. Tiny-diffusion is a simplified version of diffusion, which runs on the resource-constrained Mica motes (figure 5(e)) running TinyOS [20] with a limited amount of memory and processing power. Although it does not include support for filters, this simplified version does support attributes as well as a simplified version of the publish/subscribe API. Different versions of tiny diffusion have implemented both two-phase and one-phase pull diffusion.

5.2 Evaluation of Diffusion Design Choices

In this section, we use packet-level simulation to explore, in some detail, the implications of some of our design choices. Such an examination complements and extends our description for the two-phase pull diffusion from Section 3.1. This section describes our methodology, compares the performance of diffusion against some idealized schemes, then considers impact of network dynamics on simulation. Please refer to [22] for a more detailed description of the simulations.

(a) Our PC/104 node

(b) An Intel Stayton node

(c) WINSng 1.0 node

(d) WINSng 2.0 node

(e) UCB Mica-1 Mote

Figure 5: Diffusion hardware platforms. The mote supports only Tiny Diffusion.

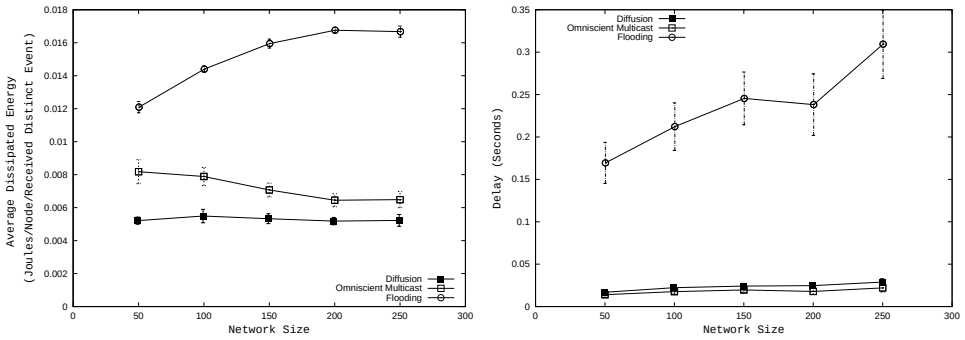
5.2.1 Goals, metrics, and methodology

We implemented a vehicle tracking instance of directed diffusion in the *ns-2* [2] simulator (the current ns release with diffusion support can be downloaded from http://www.isi.edu/nsnam/ns). Our goals in conducting this evaluation study were to: verify and complement our analytic evaluation, understand the impact of dynamics—such as node failures—on diffusion, and study the sensitivity of directed diffusion performance to the choice of parameters.

We choose two metrics to analyze the performance of directed diffusion and to compare it to other schemes: mean dissipated energy and mean delay. **Mean 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 mean work done by a node in delivering useful tracking information to the sinks. The metric also indicates the overall lifetime of sensor nodes. **Mean delay** measures the mean 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. We study these metrics as a function of sensor network size.

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 1.6Mb/s 802.11 MAC layer. Our simulations use a modified 802.11 MAC layer. To more closely mimic realistic sensor network radios [25], 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 MAC layer is not completely satisfactory, since energy efficiency provides a compelling reasons



(a) Mean dissipated energy

(b) Mean delay

Figure 6: Directed diffusion compared to flooding and omniscient multicast.

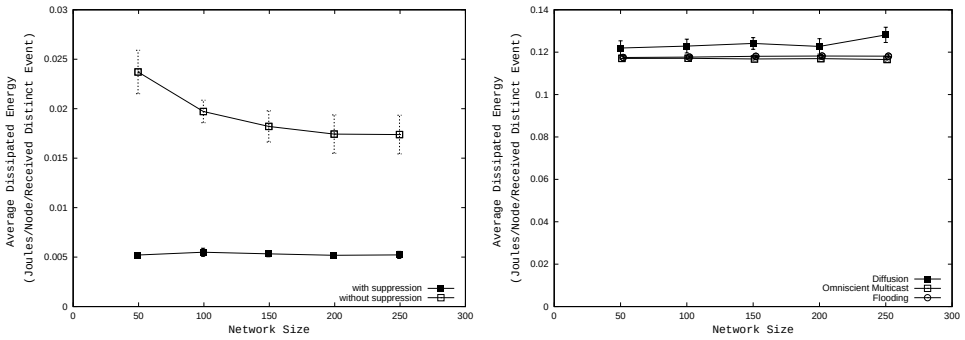
for selecting a TDMA-style MAC for sensor networks rather than one using contention-based protocols [29]. Briefly, these reasons have to do with energy consumed by the radio during idle intervals; with a TDMA-style MAC, it is possible to put the radio in standby mode during such intervals. By contrast, an 802.11 radio consumes as much power when it is idle as when it receives transmissions. In Section 5.2.3, we analyze the impact of a MAC energy model in which listening for transmissions dissipates as much energy as receiving them.

Finally, data points in each graph represent the mean of ten scenarios with 95% confidence intervals. Please refer to [22] for a more detailed description of the methodology used.

5.2.2 Comparing diffusion with alternatives

Our first experiment compares diffusion to omniscient multicast and flooding scheme for data dissemination in networks. Figure 6(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 vehicle in the specified region.

Figure 6(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



(a) Duplicate suppression

(b) High idle radio power

Figure 7: Impact of various factors on directed diffusion.

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, directed diffusion exhibits better energy dissipation than omniscient multicast and has good latency properties.

5.2.3 Effects of data aggregation

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 5.2.2 that in our simulations, we implement a simple aggregation strategy, in which a node suppresses identical data sent by different sources. As Figure 7(a) 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 truncated 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.

5.2.4 Effects of radio energy model

Finally, we evaluate the sensitivity of our comparisons (Section 5.2.2) to our choice of energy model. Sensitivity of diffusion to other factors (numbers of sinks, size of source region) is discussed in greater detail in [23].

In our comparisons, we selected radio power dissipation parameters to more closely mimic realistic sensor radios [25]. We

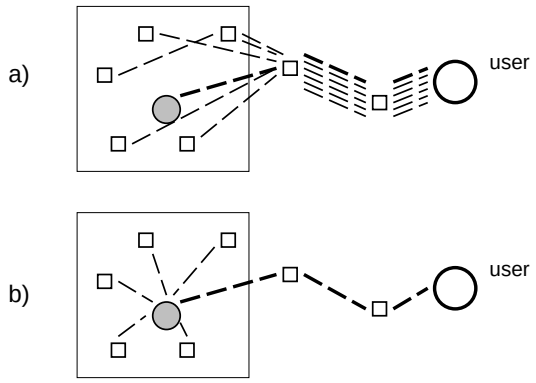


Figure 8: Two approaches to implementing nested queries. Squares are initial sensors, gray circles are triggered sensors, and the large circle is the user. Thin dashed lines represent communication to initial sensors; bold lines are communication to the triggered sensor.

re-ran the comparisons of Section 5.2.2, but with power dissipation comparable to the AT&T Wavelan: 1.6W transmission, 1.2W reception and 1.15W idle [32]. In this case, as Figure 7(b) 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.

5.3 Evaluation of In-Network Processing

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 8 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 [6].

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

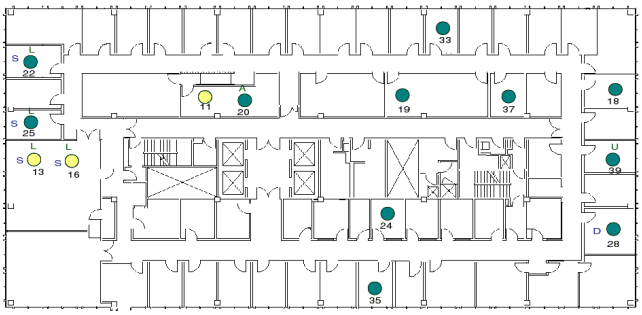


Figure 9: Node positions in our sensor testbed. Light nodes (11, 13, 16) are on the 10th floor; the remaining dark nodes are on the 11th floor. Radio range varies greatly depending on node position, but the longest stable link was between nodes 20 and 25.

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 he 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 [16]). 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 the next section we evaluate nested queries with experiments in our testbed.

5.3.1 Goals and methodology

To validate our claim about the potential performance benefits of this implementation we measure the performance of an application that uses nested queries against one that does not.

The application is similar to that described in Figure 8: a user requests acoustic data correlated with (triggered by) light sensors. For this experiment, we used our testbed of 14 PC/104 sensor nodes distributed on two floors of ISI (Figure 9). These

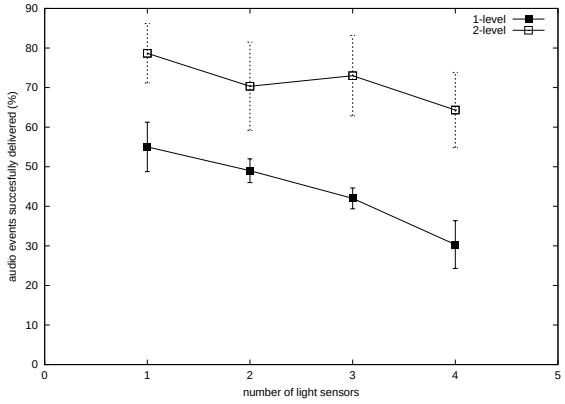


Figure 10: Percentage of audio events successfully delivered to the user.

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. We placed 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 we simulate 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 desynchronize sensors). Audio sensors generate simulated audio data each time any light sensor changes state. Light and audio data messages are about 100 bytes long.

5.3.2 Nested queries benefits

Figure 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 the network exhibits very high loss rates. 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. 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.

5.4 Application Performance with Different Diffusion Algorithms

In Section 3 we 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.

5.4.1 One-phase 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 11 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.

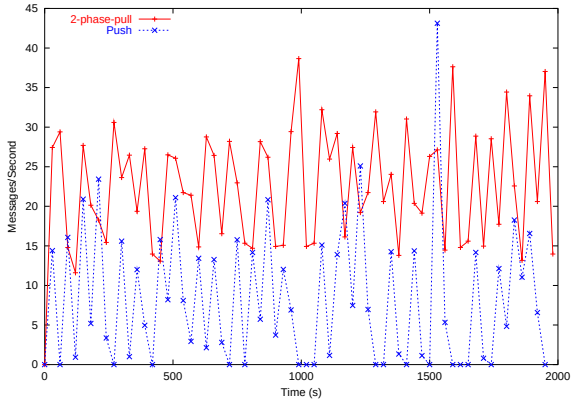


Figure 11: Push vs. two-phase pull diffusion with a cross-subscription application.

5.4.2 Geographic constraints

Researchers at Xerox PARC have suggested Information Driven Sensor Querying [13], an information-theoretic approach to sensornet 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 5.4.1). 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 [34]. This application runs over 18 WINSng 2.0 nodes in the PARC sensor network testbed. 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 12 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.

5.4.3 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 algorithm 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

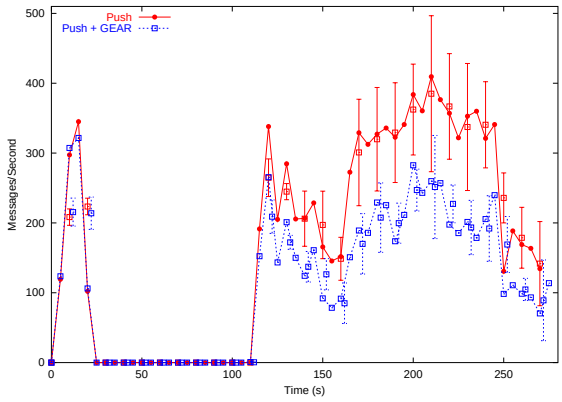


Figure 12: Push diffusion with and without GEAR over the IDSQ application.

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 [11]).

6 Related Work

Space constraints preclude a detailed summary of related work; for a more detailed study, see the related work sections of prior papers [22, 18, 24, 17].

Our publish/subscribe API was designed with researchers from MIT's Lincoln Labs [14]. The approach was inspired by prior, Internet-based publish/subscribe systems (examples include [28, 5, 27, 12]). The concept of formals and actuals in matching is derived from Linda [10], and our application of attribute based naming to sensor networks was inspired by SRM [16].

The diffusion approach to data dissemination can be compared to ad hoc routing protocols (Broch et al. survey several protocols [9] including DSR and AODV). Unlike end-to-end Internet protocols, diffusion encourages in-network processing. In-network processing in diffusion is similar to active networking [33], although the domain is quite different. DataSpace provided geographic routing [21], but not tied with attributes.

In sensor networks, Piconet provided a fairly static system with devices, concentrators, and hosts [4]. SPIN evaluates several variants of flooding for wireless sensor networks [19], and INS designed a naming system for Internet-based hosts [1]. Neither exploited in-network processing. COUGAR adopted database-like approach to sensor networks [6] and inspired our approach to nested queries.

7 Conclusion

We have described directed diffusion, a data-centric approach to information dissemination for sensor networks. Building on a publish/subscribe API and attribute-based naming, the diffusion primitives support a family of routing algorithms optimized for different applications. Filters support in-network processing to allow applications to manipulate data as it flows through the network.

Acknowledgments

Directed diffusion builds on the work of many people. Van Jacobson suggested the concept of diffusion as a communication strategy. Chalermek Intanagonwiwat designed and evaluated the basic algorithm. Dan Coffin collaborated in definition of the publish/subscribe API. Philippe Bonnet inspired our approach to nested queries. Yan Yu, Fred Stann, Ben Greenstein, Xi Wang developed filters. Philippe Bonnet and Joe Reynolds were early users of diffusion, Jim Reich and Julia Liu early users of push, and Eric Osterweil of one-phase pull diffusion.

References

[1] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremy Lilley. The design and implementation of an intentional naming system. In *Proceedings of the 17th Symposium on Operating Systems Principles*, pages 186–201, Kiawah Island, SC, USA,

December 1999. ACM.

[2] Sandeep Bajaj, Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, Padma Haldar, Mark Handley, Ahmed Helmy, John Heidemann, Polly Huang, Satish Kumar, Steven McCanne, Reza Rejaie, Puneet Sharma, Kannan Varadhan, Ya Xu, Haobo Yu, and Daniel Zappala. Improving simulation for network research. Technical Report 99-702b, University of Southern California, March 1999. revised September 1999, to appear in IEEE Computer.

[3] Hari Balakrishnan, Srinivasan Seshan, Elan Amir, and Randy H. Katz. Improving TCP/IP performance over wireless networks. In *Proceedings of the First ACM Conference on Mobile Computing and Networking*, pages 2–11, Berkeley, CA, USA, November 1995. ACM.

[4] Frazer Bennett, David Clarke, Joseph B. Evans, Andy Hopper, Alan Jones, and David Leask. Piconet: Embedded mobile networking. *IEEE Personal Communications Magazine*, 4(5):8–15, October 1997.

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

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

[7] Robert Braden, Theodore Faber, and Mark Handley. From protocol stack to protocol heap–role-based architecture. In *Proceedings of the ACM Workshop on Hot Topics in Networks I*, page to appear, Princeton, NJ, USA, October 2002. ACM.

[8] 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.

[9] 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.

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

[11] 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.

[12] Antonio Carzaniga, David S. Rosenblum, and Alexander L Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.

[13] 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.

[14] 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*, San Diego, California, USA, July 2000. SPIE. (part of SPIE International Symposium on Optical Science and Technology).

[15] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *Proceedings of the ACM SIGCOMM Conference*, pages 342–356, Cambridge, Massachusetts, August 1995. ACM.

[16] 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.

[17] John Heidemann, Fabio Silva, and Deborah Estrin. Matching data dissemination algorithms to application requirements. In *Proceedings of the ACM SenSys Conference*, page to appear, Los Angeles, California, USA, November 2003. ACM.

[18] 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.

[19] W. R. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 174–185, Seattle, WA, USA, August 1999. ACM.

[20] 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.

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

[22] 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.

[23] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. Technical Report 00-732, University of Southern California, March 2000.

[24] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva. Directed diffusion for wireless sensor networking. *ACM/IEEE Transactions on Networking*, 11(1):2–16, February 2003.

[25] William J. Kaiser. WINS NG 1.0 Transceiver Power Dissipation Specifications. Sensoria Corp.

[26] 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.

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

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

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

[30] Fabio Silva, John Heidemann, and Ramesh Govindan. *Network Routing Application Programmer's Interface (API) and Walk Through 9.0.1*. USC/Information Sciences Institute, December 2002.

[31] Fred Stann and John Heidemann. Rmst: Reliable data transport in sensor networks. In *Proceedings of the First International Workshop on Sensor Net Protocols and Applications*, pages 102–112, Anchorage, Alaska, USA, April 2003. USC/Information Sciences Institute, IEEE.

[32] 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.

[33] 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.

[34] 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.

[35] Feng Zhao, Jaewon Shin, and James Reich. Information-driven dynamic sensor collaboration for tracking applications. *IEEE Signal Processing Magazine*, page to appear, March 2002.