Directed Diffusion for Wireless Sensor Networking

Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva

***Abstract*—Advances in processor, memory and radio technology will enable small and cheap nodes capable of sensing, communication and computation. Networks of such nodes can coordinate to perform distributed sensing of environmental phenomena. In this paper, we explore the *directed diffusion* paradigm for such coordination. Directed diffusion is data-centric in that all communication is for named data. All nodes in a directed diffusion-based network are application-aware. This enables diffusion to achieve energy savings by selecting empirically good paths and by caching and processing data in-network (*e.g.,* data aggregation). We explore and evaluate the use of directed diffusion for a simple remote-surveillance sensor network analytically and experimentally. Our evaluation indicates that directed diffusion can achieve significant energy savings and can outperform idealized traditional schemes (*e.g.,* omniscient multicast) under the investigated scenarios.**

1 Introduction

In the near future, advances in processor, memory and radio technology will enable small and cheap nodes capable of wireless communication and significant computation. The addition of sensing capability to such devices will make distributed microsensing—an activity in which a collection of nodes coordinate to achieve a larger sensing task—possible. Such technology can revolutionize information gathering and processing in many situations. Large scale, dynamically changing, and robust *sensor networks* can be deployed in inhospitable physical environments such as remote geographic regions or toxic urban locations. They will also enable low maintenance sensing in more benign, but less accessible, environments: large industrial plants, aircraft interiors etc.

To motivate our research, consider this simplified model of how such a sensor network will work. One or more human operators pose, to any node in the network, questions of the form: "How many pedestrians do you observe in the geographical region X?", or "Tell me in what direction that vehicle in region Y is moving". These queries result in sensors within the specified region being *tasked* to start collecting information. Once individual nodes detect pedestrians or vehicle movements, they might collaborate with neighboring nodes to disambiguate pedestrian location or vehicle movement direction. One of these nodes might then report the result back to the human operator.

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 2).

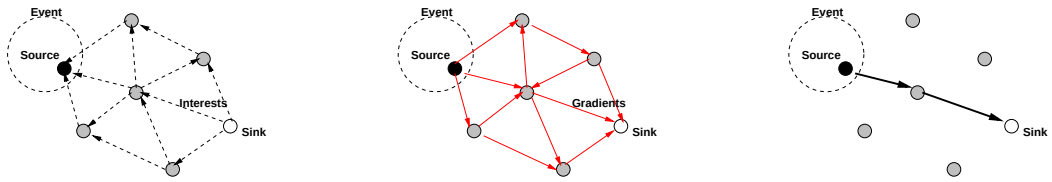
Using this communication paradigm, our example might be implemented as follows. The human operator's query would be transformed into an interest that is *diffused* (*e.g.,* broadcasted, geographically routed) towards nodes in regions X or Y. When a node in that region receives an interest, it activates its sensors which begin collecting information about pedestrians. When the sensors report the presence of pedestrians, this information returns along the reverse path of interest propagation. Intermediate nodes might *aggregate* the data, *e.g.,* more accurately pinpoint the pedestrian's location by combining reports from several sensors. An important feature of directed diffusion is that interest and data propagation and aggregation are determined by *localized interactions* (message exchanges between neighbors or nodes within some vicinity).

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 this paper, we describe directed diffusion and illustrate one instantiation of this paradigm for sensor query dissemination and processing. 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 4). We have also implemented directed diffusion on several small sensor platforms; we describe our implementation design and our experiences in Section 5.

In this paper, we outline the directed diffusion paradigm, explain its key features, and describes in some detail a particular instantiation of the directed diffusion paradigm for a vehicle tracking sensor network (Section 2). We specify what local rules achieve the desired behavior of interest and data propagation. In doing so, we show how the directed diffusion paradigm differs from traditional networking, and qualitatively argue that this paradigm offers scaling, robustness and energy efficiency benefits. We quantify some of these benefits via detailed packet-level simulation of directed diffusion (Section 4) and an implementation (Section 5).

C. Intanagonwiwat, R. Govindan, J. Heidemann, and F. Silva are with USC/Information Sciences Institute. D. Estrin is with University of California, Los Angeles. An earlier version of this work appeared in Proceedings of the ACM Mobicom 2000 [17]. This work was supported by the Defense Advanced Research Projects Agency under grant DABT63-99-1-0011.

[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.



(a) Interest propagation

(b) Initial gradients set up

(c) Data delivery along reinforced path

Figure 1: A simplified schematic for directed diffusion.

2 Directed Diffusion

Directed diffusion consists of several elements: interests, data messages, gradients, and reinforcements. An *interest* message is a query or an interrogation which specifies what a user wants. Each interest contains a description of a sensing task that is supported by a sensor network for acquiring data. Typically, *data* in sensor networks is the collected or processed information of a physical phenomenon. Such data can be an *event* which is a short description of the sensed phenomenon. In directed diffusion, data is *named* using attribute-value pairs. A sensing task (or a subtask thereof) is disseminated throughout the sensor network as an *interest* for named data. This dissemination sets up *gradients* within the network designed to "draw" events (*i.e.*, data matching the interest). Specifically, a *gradient* is direction state created in each node that receives an interest. The gradient direction is set toward the neighboring node from which the interest is received. Events start flowing towards the originators of interests along multiple gradient paths. The sensor network *reinforces* one, or a small number of these paths. Figure 1 illustrates these elements.

In this section, we describe these elements of diffusion with specific reference to a particular kind of sensor network—one that supports a location tracking task. As we shall see, several design choices present themselves even in the context of this specific instantiation of diffusion. We elaborate on these design choices while describing the design of our sensor network. Our initial evaluation (Section 4) focuses only a subset of these design choices. Different design choices result in different variants of diffusion (see also [16] for another variant). Moreover, even though we describe this diffusion variant for rate-based applications, diffusion also works for event-triggered applications.

2.1 Naming

In directed diffusion, task descriptions are *named* by, for example, a list of attribute-value pairs that describe a task. A vehicle tracking task might be described as (this is a simplified description, see Section 2.2 for more details):

```
type = wheeled vehicle       // detect vehicle location
interval = 20 ms             // send events every 20 ms
duration = 10 seconds        // for the next 10 seconds
rect = [-100, 100, 200, 400] // from sensors within
                             // rectangle
```

For ease of exposition, we choose the subregion representation to be a rectangle defined on some coordinate system; in practice, this might be based on GPS coordinates.

Intuitively, the task description specifies an interest for data matching the attributes. For this reason, such a task description is called an *interest*. The data sent in response to interests are also named using a similar naming scheme. Thus, for example, a sensor that detects a wheeled vehicle might generate the following data (see Section 2.3 for an explanation of some of these attributes):

```
type = wheeled vehicle    // type of vehicle seen
instance = truck          // instance of this type
location = [125, 220]     // node location
intensity = 0.6           // signal amplitude measure
confidence = 0.85         // confidence in the match
timestamp = 01:20:40      // event generation time
```

Given a set of tasks supported by a sensor network, then, selecting a naming scheme is the first step in designing directed diffusion for the network. For our sensor network, we have chosen a simple attribute-value based interest and data naming scheme. In general, each attribute has an associated value range. For example, the range of the `type` attribute is the set of codebook values representing mobile objects (vehicles, animal, humans). The value of an attribute can be any subset of its range. In our example, the value of the `type` attribute in the interest is that corresponding to wheeled vehicles

There are other choices for attribute value ranges (*e.g.*, hierarchical) and other naming schemes (such as intentional names [1]). To some extent, the choice of naming scheme can affect the expressivity of tasks, and may impact performance of a diffusion algorithm. In this paper, our goal is to gain an initial understanding of the diffusion paradigm. For this reason, the exploration of possible naming schemes is beyond the scope of this paper, but we have begun that exploration elsewhere [14] (see also Section 5).

2.2 Interests and Gradients

The named task description of Section 2.1 constitutes an *interest*. An interest is usually injected into the network at some (possibly arbitrary) node in the network. We use the term *sink* to denote this node.

2.2.1 Interest Propagation

Given our choice of naming scheme, we now describe how interests are *diffused* through the sensor network. Suppose that a task, with a specified `type` and `rect`, a `duration` of 10 minutes and an `interval` of 10ms, is instantiated at a particular node in the network. The `interval` parameter specifies an event data rate; thus, in our example, the specified data rate is 100 events per second. This sink node records the task; the task state is purged from the node after the time indicated by the `duration` attribute.

For each active task, the sink periodically *broadcasts* an interest message to each of its neighbors (more efficient methods to send the interest will be discussed later). This initial interest contains the specified `rect` and `duration` attributes, but contains a much larger `interval` attribute. Intuitively, this initial interest may be thought of as *exploratory*; it tries to determine if there indeed are any sensor nodes that detect the wheeled vehicle To do this, the initial exploratory interest specifies a low data rate (in our example, 1 event per second) [2]. In Section 2.4, we describe how the desired data rate is achieved by reinforcement. Then, the initial interest takes the following form:

```
type = wheeled vehicle
interval = 1s
rect = [-100, 200, 200, 400]
timestamp = 01:20:40                    // hh:mm:ss
expiresAt = 01:30:40
```

Before we describe how interests are processed, we emphasize that the interest is soft state [19, 29, 32] that will be periodically *refreshed* by the sink. To do this, the sink simply re-sends the same interest with a monotonically increasing `timestamp` attribute. This is necessary because interests are not reliably transmitted throughout the network. The refresh rate is a protocol design parameter that trades off overhead for increased robustness to lost interests.

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

An entry in the interest cache has several fields. A `timestamp` field indicates the timestamp [3] of the last received matching interest. The interest entry also contains several `gradient` fields, up to one per neighbor. Each gradient contains a `data rate` field requested by the specified neighbor, derived from the `interval` attribute of the interest. It also contains a *duration* field, derived from the `timestamp` and `expiresAt` attributes of the interest, and indicating the approximate lifetime of the interest. This duration must be longer than the network delay.

When a node receives an interest, it checks to see if the interest exists in the cache. If no matching entry exists (where a match is determined by the definition of distinct interests specified above), the node creates an interest entry. The parameters of the interest entry are instantiated from the received interest. This entry has a single gradient towards the neighbor from which the interest was received, with the specified event data rate. In our example, a neighbor of the sink will set up an interest entry with a gradient of 1 event per second towards the sink. For this, it must be possible to distinguish individual neighbors. Any locally unique neighbor identifier may be used for this purpose. Examples of such identifiers include 802.11 MAC addresses [8], Bluetooth [13] cluster addresses, or locally unique ephemeral identifiers [11]. If there exists an interest entry, but no gradient for the sender of the interest, the node adds a gradient with the specified value. It also updates the entry's `timestamp` and `duration` fields appropriately. Finally, if there exists both an entry *and* a gradient, the node simply updates the `timestamp` and `duration` fields.

In Section 2.3, we describe how gradients are used. When a gradient expires, it is removed from its interest entry. Not all gradients will expire at the same time. For example, if two different sinks express indistinct interests with different expiration times, some node in the network may have an interest entry with different gradient expiration times. When all gradients for an interest entry have expired, the interest entry itself is removed from a cache.

After receiving an interest, a node may decide to re-send the interest to some subset of its neighbors. To its neighbors, this interest *appears to originate from the sending node*, although it might have come from a distant sink. This is an example of a *local interaction*. In this manner, interests *diffuse* throughout the network. Not all received interests are re-sent. A node may suppress a received interest if it recently re-sent a matching interest.

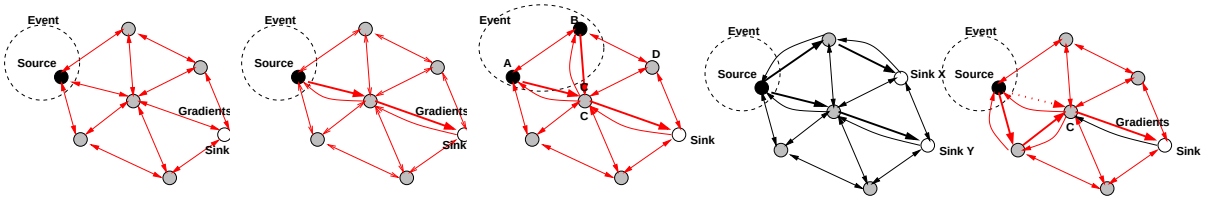
Generally speaking, there are several possible choices for neighbors (Figure 2). The simplest alternative is to *re-broadcast* the interest to all neighbors. This is equivalent to flooding the interest throughout the network; in the absence of information about which sensor nodes are likely to be able to satisfy the interest, this is the only choice. This is also the alternative that we simulate in Section 4. In our example sensor network, it may also be possible to perform geographic routing, using some of the techniques described in the literature [23, 34, 7]. This can limit the topological scope for interest diffusion, thereby resulting in energy savings. Finally, in an immobile sensor network, a node might use cached data (see Section 2.3) to direct interests. For example, if in response to an earlier interest, a node heard from some neighbor *A* data sent by some sensor within the region specified by the `rect` attribute, it can direct this interest to *A*, rather than broadcasting to all neighbors.

[2] This is not the only choice, but represents a performance tradeoff. Since the location of the sources is not precisely known, interests must necessarily be diffused over a broader section of the sensor network than that covered by the potential sources. As a result, if the sink had chosen a higher initial data rate, a higher energy consumption might have resulted from the wider dissemination of sensor data. However, with a higher initial data rate, the time to achieve high fidelity tracking is reduced.

[3] This may require time synchronization among nodes in the network. However, time can be synchronized using GPS [22], NTP [25], or using post-facto messaging [12].

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

Figure 2: Partial Design Space for Diffusion



(a) Gradient establishment (b) Reinforcement (c) Multiple sources (d) Multiple sinks (e) Repair

Figure 3: Illustrating different aspects of diffusion.

2.2.2 Gradient Establishment

Figure 3(a) shows the gradients established in the case where interests are flooded through a sensor field. Unlike the simplified description in Figure 1(b), notice that every pair of neighboring nodes establishes a gradient towards each other. This is a crucial consequence of local interactions. When a node receives an interest from its neighbor, it has no way of knowing whether that interest was in response to one it sent out earlier, or is an identical interest from another sink on the "other side" of that neighbor. Such two-way gradients can cause a node to receive one copy of low data rate events from each of its neighbors. However, as we show later, this technique can enable fast recovery from failed paths or reinforcement of empirically better paths (Section 2.4), and does not incur persistent loops (Section 2.3).

Note that for our sensor network, a gradient specifies both a data rate and a direction in which to send events. More generally, a gradient specifies a *value* and a direction. The directed diffusion paradigm gives the designer the freedom to attach different semantics to gradient values. We have shown two examples of gradient usage. Figure 1(c) implicitly depicts binary valued gradients. In our sensor networks, gradients have two values that determine event reporting rate. In other sensor networks, gradient values might be used to, for example, probabilistically forward data along different paths, achieving some measure of load balancing (Figure 2).

In summary, interest propagation sets up state in the network (or parts thereof) to facilitate "pulling down" data towards the sink. The interest propagation rules are *local*, and bear some resemblance to join propagation in some Internet multicast routing protocols [10]. One crucial difference is that join propagation can leverage unicast routing tables to direct joins towards sources, whereas interest propagation cannot.

In this section, we have described interest propagation rules for a particular type of task. More generally, a sensor network may support many different task types. Interest propagation rules may be different for different task types. For example, a task type of the form "Count the number of distinct wheeled vehicles in rectangle R seen over the next T seconds" cannot leverage the event data rate as our example does. However, some elements of interest propagation are similar to both: the form of the cache entries, the interest re-distribution rules *etc.* In our implementation (Section 5), we have culled these similarities into a *diffusion* substrate at each node, so that sensor network designers can use a library of interest propagation techniques (or, for that matter, rules discussed in the subsequent sections for data processing and reinforcement) for different task types.

2.3 Data Propagation

A sensor node that is within the specified `rect` processes interests as described in the previous section. In addition, the node tasks its local sensors to begin collecting samples (to save power, sensors are off until tasked). In this paper, we do not discuss the details of target recognition algorithms. Briefly, these algorithms simply match sampled waveforms against a library of pre-sampled, stored waveforms. This is based on the observation that a wheeled vehicle has a different acoustic or seismic footprint than, for example, a human being. The sampled waveform may match the stored waveform to varying extents; the algorithms usually associate a degree of confidence with the

match. Furthermore, the intensity of the sampled waveform may roughly indicate distance of the signal origin, though perhaps not direction.

A sensor node that detects a target searches its interest cache for a matching interest entry. In this case, a matching entry is one whose `rect` encompasses the sensor location, and the `type` of the entry matches the detected target type. When it finds one, it computes the highest requested event rate among all its outgoing gradients. The node tasks its sensor subsystem to generate event samples at this highest data rate. In our example, this data rate is initially 1 event per second (until reinforcement is applied, Section 2.4). The source then sends to each neighbor for whom it has a gradient, an event description every second of the form:

```
type = wheeled vehicle  // type of vehicle seen
instance = truck        // instance of this type
location = [125, 220]   // node location
intensity = 0.6         // signal amplitude measure
confidence = 0.85       // confidence in the match
timestamp = 01:20:40    // local event generation time
```

This *data* message is, in effect, unicast individually to the relevant neighbors. (The exact mechanism used is a function of the radio's MAC layer and can have a significant impact on performance as evaluated in Section 4.4.)

A node that receives a data message from its neighbors attempts to find a matching interest entry in its cache. The matching rule is as described in the previous paragraph. If no match exists, the data message is silently dropped. If a match exists, the node checks the *data cache* associated with the matching interest entry. This cache keeps track of recently seen data items. It has several potential uses, one of which is loop prevention. If a received data message has a matching data cache entry, the data message is silently dropped. Otherwise, the received message is added to the data cache and the data message is re-sent to the node's neighbors.

By examining its data cache, a node can determine the data rate of received events [4]. To re-send a received data message, a node needs to examine the matching interest entry's gradient list. If all gradients have a data rate that is greater than or equal to the rate of incoming events, the node may simply send the received data message to the appropriate neighbors. However, if some gradients have a lower data rate than others (caused by selectively reinforcing paths, Section 2.4), then the node may *down-convert* to the appropriate gradient. For example, consider a node that has been receiving data at 100 events per second, but one of its gradients (*e.g.*, set up by a second sink originating an indistinct task with a larger `interval`) is at 50 events per second. In this case, the node may only transmit every alternate event towards the corresponding neighbor. Alternately, it might interpolate two successive events in an application-specific way (in our example, it might choose the sample with the higher confidence match).

Loop prevention and down-conversion illustrate the power of embedding application semantics in all nodes (Figure 2). Although this design is not pertinent to traditional networks, it is feasible with application-specific sensor networks. Indeed, as we show in Section 4.4, it can significantly improve network performance.

[4] In our simulations in Section 4, as a simplification, we include the data rate in the event descriptions.

2.4 Reinforcement for Path Establishment and Truncation

In the scheme we have described so far, the sink initially and repeatedly diffuses an interest for a low-rate event notification. We call these *exploratory* events, since they are intended for path setup and repair. We call the gradients set up for exploratory events *exploratory* gradients. Once a source detects a matching target, it sends exploratory events, possibly along multiple paths, towards the sink. After the sink starts receiving these exploratory events, it *reinforces* one particular neighbor in order to "draw down" real *data* (*i.e.*, events at a higher data rate that allow high quality tracking of targets). We call the gradients set up for receiving high quality tracking events *data* gradients.

2.4.1 Path Establishment Using Positive Reinforcement

In general, this novel feature of directed diffusion is achieved by *data driven* local rules. One example of such a rule is to reinforce any neighbor from which a node receives a previously unseen event. To reinforce this neighbor, the sink re-sends the original interest message but with a smaller `interval` (higher data rate):

```
type = wheeled vehicles
interval = 10ms
rect = [-100, 200, 200, 400]
timestamp = 01:22:35
expiresAt = 01:30:40
```

When the neighboring node receives this interest, it notices that it already has a gradient towards this neighbor. Furthermore, it notices that the sender's interest specifies a higher data rate than before. If this new data rate is also higher than that of any existing gradient (intuitively, if the "outflow" from this node has increased), the node must also reinforce at least one neighbor. How does it do this? The node uses its data cache for this purpose. Again, the same local rule choices apply. For example, this node might choose that neighbor from whom it first received the latest event matching the interest. Alternatively, it might choose all neighbors from which new events were recently received. This implies that we reinforce that neighbor only if it is sending exploratory events. Obviously, we do not need to reinforce neighbors that are already sending traffic at the higher data rate. This is the alternative we evaluate in Section 4. Through this sequence of local interactions, a path is established from source to sink transmission for data.

The local rule we described above, then, *selects an empirically low-delay path* (Figure 3(b) shows the path that can result when the sink reinforces the path). It is very reactive to changes in path quality; whenever one path delivers an event faster than others, the sink attempts to use this path to draw down high quality data. However, because it is triggered by receiving one new event, this could be wasteful of resources. More sophisticated local rules are possible (Figure 2), including choosing that neighbor from which the most events have been received, or that neighbor which *consistently* sends events before other neighbors. These choices trade off reactivity for increased stability.

2.4.2 Path Establishment for Multiple Sources and Sinks

In describing reinforcement so far, we may have appeared to implicitly describe a single-source scenario. In fact, the rules we have described work with multiple sources. To see this, consider Figure 3(c). Assume initially that all initial gradients are exploratory. According to this topology, data from both sources reaches the sink via both of its neighbors **C** and **D**. If one of the neighbors, say **C** has consistently lower delay, our rules will only reinforce the path through **C** (this is depicted in the figure). However, if the sink hears **B**'s events earlier via **D**, but **A**'s events [5] earlier via **C**, the sink will attempt to draw down high quality data streams from *both* neighbors (not shown). In this case, the sink gets both sources' data from both neighbors, a potential source of energy inefficiency. Such problem can be avoided with some added complexity [16].

Similarly, if two sinks express identical interests, our interest propagation, gradient establishment and reinforcement rules work correctly. Without loss of generality, assume that sink **Y** in Figure 3(d) has already reinforced a high quality path to the source. Note however, that other nodes continue to receive exploratory events. When a human operator tasks the network at sink **X** with an identical interest, **X** can use the reinforcement rules to achieve the path shown. To determine the empirically best path, **X** *need not wait* for data—rather, it can use its data cache to immediately draw down high quality data towards itself.

2.4.3 Local Repair for Failed Paths

So far, we have described situations in which reinforcement is triggered by a sink. However, in directed diffusion, *intermediate* nodes on a previously reinforced path can apply the reinforcement rules. This is useful to enable *local repair* of failed or degraded paths. Causes for failure or degradation include node energy depletion, and environmental factors affecting communication (*e.g.*, obstacles). Consider Figure 3(e), in which the quality of the link between the source and node **C** degrades and events are frequently corrupted. When **C** detects this degradation—either by noticing that the event reporting rate from its upstream neighbor (the source) is now lower, or by realizing that other neighbors have been transmitting previously unseen location estimates—it can apply the reinforcement rules to discover the path shown in the figure. Eventually, **C** negatively reinforces the direct link to the source (not shown in the figure). Our description so far has glossed over the fact that a straightforward application of reinforcement rules will cause all nodes downstream of the lossy link to also initiate reinforcement procedures. This will eventually lead to the discovery of one empirically good path, but may result in wasted resources. One way to avoid this is for **C** to interpolate location estimates from the events that it receives so that downstream nodes still perceive high quality tracking.

[5] Note that in directed diffusion, the sink would not be able to associate a source with an event. Thus, the phrase "**A**'s events" is somewhat misleading. What we really mean is that data generated by **A** that is distinguishable in content from data generated by **B**.

2.4.4 Path Truncation Using Negative Reinforcement

The algorithm described above (Section 2.4.1) can result in more than one path being reinforced. For example (Figure 4(a)), if the sink reinforces neighbor **A**, but then receives a new event from neighbor **B**, it will reinforce the path through **B** [6]. If the path through **B** is consistently better (*i.e.*, **B** sends events before **A** does), we need a mechanism to *negatively reinforce* the path through **A**.

One mechanism for negative reinforcement is soft state, *i.e.*, to time out all data gradients in the network unless they are explicitly reinforced. With this approach, the sink would periodically reinforce neighbor **B**, and cease reinforcing neighbor **A**. All gradients along the path through **A** would eventually degrade to being exploratory gradients. Another approach, and one that we evaluate in this paper, is to explicitly degrade the path through **A** by sending a negative reinforcement message to **A**. In this rate-based diffusion, the negative reinforcement is the interest with the lower data rate. When **A** receives this interest, it degrades its gradient towards the sink. Furthermore, *if all its gradients are now exploratory*, **A** negatively reinforces those neighbors that have been sending data to it (as opposed to exploratory events) [7]. This sequence of local interactions ensures that the path through **A** is degraded rapidly, but at the cost of increased resource utilization.

To complete our description of negative reinforcement, we need to specify what local rule a node uses in order to decide whether to negatively reinforce a neighbor or not. Note that this rule is orthogonal to the choice of mechanism for negative reinforcement. One plausible choice for such a rule is to negatively reinforce that neighbor from which no new events have been received (*i.e.*, other neighbors have consistently sent events before this neighbor) within a window of N events or time T. The local rule we evaluate in Section 4 is based on a time window of T, chosen to be 2 seconds in our simulations. Such a rule is a bit conservative and energy inefficient. For example, even if one event in ten was received first from neighbor A, the sink will not negatively reinforce that neighbor. Other variants include negatively reinforcing that neighbor from which fewer new events have been received.

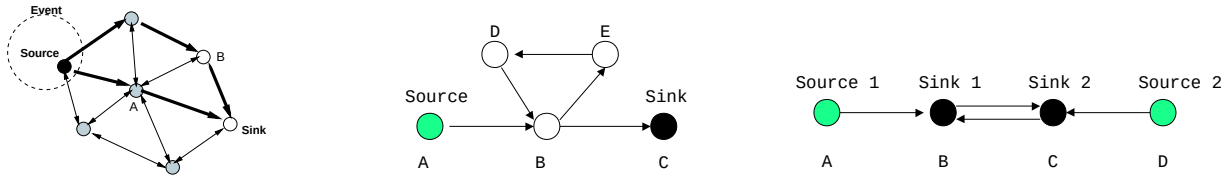
2.4.5 Loop Removal Using Negative Reinforcement

In addition to suppressing high-delay or lossy paths, our local rule for negative reinforcement is also used for loop removal because the looping paths never deliver events first [8] (Figure 4(b)). Although the looping message will be immediately suppressed using a message cache, in general, we would still benefit from truncating the looping paths for resource savings. However, such loop removal is not always appropriate, specifically for some shared high-rate gradient maps with multiple sources and

[6] This path may or may not be completely disjoint from the path through neighbor **A**.

[7] This local rule works even if the path through **A** and the path through **B** are partially joint. The joint links will not be negatively reinforced unless both paths are negatively reinforced.

[8] Given that only neighbors that sent the exploratory event first are reinforced, one may expect that the looping paths would never be reinforced (particularly for single-source-single-sink scenarios). However, the reinforced paths in a given round of exploratory events may differ from those in the previous rounds. Although no looping path is reinforced, a union of reinforced paths from multiple rounds may contain loops.



(a) Multiple paths (b) A removable loop (c) An unremovable loop

Figure 4: Negative reinforcement for path truncation and loop removal.

sinks. For example (Figure 4(c)), if both sources send distinguishable events, the gradient **B**-**C** and **C**-**B** should not be truncated because each of them is necessary for delivering events for a particular source-sink pair. Although such gradients may deliver some looping events, they also consistently deliver new events. With our conservative rule for negative reinforcement, those gradients will not be negatively reinforced.

Furthermore, even without loops, it is still reasonable to keep our negative reinforcement rule conservative so that useful paths will not be truncated. For example (Figure 3(c)), both sources may consistently send distinguishable events but they may also send identical events once in a while. Although originating from different sources, the identical events are considered duplicates for diffusion [9]. The path from one of the sources will be truncated if the negative reinforcement rule is too aggressive against duplicates. Conversely, given our conservative rule, no source will be negatively reinforced.

2.5 Discussion

In introducing the various elements of directed diffusion, we also implicitly described a particular *usage*—interests set up gradients drawing down data. The directed diffusion paradigm itself does not limit the designer to this particular usage. Other usages are also possible, such as the one in which nodes may propagate data in the absence of interests, implicitly setting up gradients when doing so. This is useful, for example, to spontaneously propagate an important event to some section of the sensor field. A sensor node can use this to warn other sensor nodes of impending activity. Moreover, other design choices for each element of diffusion are also possible (see Figure 2).

Our 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, unlike the end-to-end communication in traditional data networks. In other words, every node is an "end" in a sensor network. In the sense, 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, sensor nodes do not need to have globally unique identifiers or globally unique addresses. Nodes, however, do need to distinguish among neighbors. Finally, in an IP-based sensor network, for example, sensor data collection and processing might be performed by a collection of specialized servers which may, in general, be far removed from the sensed phenomena. In our sensor network, because every node can cache, aggregate, and more generally, process messages, it is generally desirable to perform coordinated sensing close to the sensed phenomena.

Diffusion is clearly related to traditional network data routing algorithms. In some sense, it is a *reactive* routing technique, since "routes" are established on demand. However, it differs from other ad-hoc reactive routing techniques in several ways (see also Section 6). First, no attempt is made to find one loop-free path between source and sink before data transmission commences. Instead, constrained or directional flooding is used to set up a multiplicity of paths, and data messages are *initially* sent redundantly along these paths. Second, soon thereafter, reinforcement attempts to reduce this multiplicity of paths to a small number, based on empirically observed path performance. Finally, a message cache is used to perform loop avoidance. The interest and gradient setup mechanisms themselves do not guarantee loop-free paths between source and sink.

Why this peculiar choice of design? At the outset of this research, we consciously chose to explore path setup algorithms that establish network paths using strictly *local* (neighbor-to-neighbor) communication. The intuition behind this choice is the observation that physical systems (*e.g.*, ant colonies [5]) that build up transmission paths using such communication scale well and are extraordinarily robust (see also Section 6). However, using strictly local communication implies that path setup cannot use global *topology* metrics; local communication implies that, as far as a node knows, the data that it received from a neighbor came from that neighbor [10]. This can be energy efficient in highly dynamic networks when changes in topology need not be propagated across the network. Of course, the resulting communication paths may be sub-optimal. However, the energy inefficiency due to path sub-optimality can be countered by carefully designed in-network aggregation techniques. Overall, we believe that this approach trades off some energy efficiency for increased robustness and scale.

[9] In diffusion, events are independent from their sources (*i.e.*, a node would not be able to associate a source with an event). The current instantiation of diffusion maintains only the high-rate paths along which useful (new) data are consistently sent, regardless of the sources. Thus, generally, we do not guarantee that there will be at least one high-rate path from every source to every sink (*e.g.*, when some of the sources are not generating useful data).

[10] The location information in a data message might reveal otherwise, but that information still doesn't contain topology metrics.

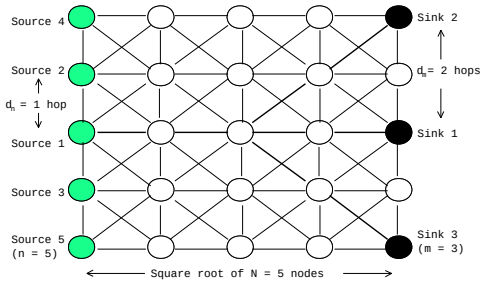


Figure 5: An example of our square grid topology

Finally, it might appear that the particular instantiation that we chose, location tracking, has limited applicability. We believe, however, that such location tracking captures many of the essential features of a large class of remote surveillance sensor networks. We emphasize that, even though we have discussed our tracking network in some detail, much experimentation and evaluation of the various mechanisms is necessary before we fully understand the robustness, scale, and performance implications of diffusion in general, and some of our mechanisms in particular. The two next sections take initial steps in this direction.

3 Analytic Evaluation

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

For analytic tractability, we analyze these three schemes in a very simple, idealized setting. We assume a square grid consisting of N nodes. In this grid, node transmission ranges are such that each node can communicate with exactly eight neighboring nodes on the grid. Figure 5 shows links between pairs of nodes that can communicate with each other. All n sources are placed along nodes on the left edge of the grid, whereas all m sinks are placed along the right edge. The first source is at the center of the left border. The i^{th} source is $d_n \lfloor \frac{i}{2} \rfloor$ hops above (if i is even) or below (if i is odd) the first source. This placement scheme is also used for sinks except that the distance between 2 adjacent sinks is d_m hops rather than d_n hops. Given that sources and sinks are vertically placed only, $\sqrt{N}$ can not be less than $max(nd_n, md_m)$.

3.1 Flooding

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

In this analytic evaluation, our measure of performance is the total cost of transmission and reception of one event from each source to all the sinks. We define cost as one unit for message transmission and one unit for message reception. These assumptions are clearly idealized in two ways. Transmission and reception costs may not be identical, and there might be other metrics of interest. We consider more realistic measures with simulation in Section 4.

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

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

The transmission cost for flooding n events (one event from each source) is nN because each node sends only one MAC broadcast per event. Conversely, each node can receive the same event from all neighbors. Thus, the reception cost for those n events is determined by $2n$ times the number of links in the network ($4n(\sqrt{N}-1)(2\sqrt{N}-1)$). The data delivery cost for flooding is $O(nN)$ which is asymptotically higher than the cost of other schemes (see Section 3.2 and 3.3).

3.2 Omniscient Multicast

In the *omniscient multicast* scheme, each source transmits its events along a shortest-path multicast tree to all sinks. In our analysis, as well as in the simulations described in Section 4, we *do not* account for the cost of tree construction. Omniscient multicast instead indicates the best possible performance achievable in an IP-based sensor network without considering overhead. We use this scheme to give the reader some intuition for how the choice of in-network processing mechanism effects performance.

For omniscient multicast, the data delivery cost is determined by twice the number of links on its source-specific shortest-path trees. However, even in this simple grid topology, there are several shortest paths for each source-sink pair. We choose the shortest path using the following simple, deterministic rule. From a sink to a source, a diagonal link is always the next hop as long as it leads to a shortest path. Otherwise, a horizontal link is selected. This path-selection rule is repeated until the source is reached. Thus, no shortest path includes vertical links. For example, if we denote a shortest path tree rooted at source j by T_j, then the number of links on T_1 has two components: the number of horizontal links ($\sqrt{N}-1$) and diagonal links ($d_m \lfloor \frac{m}{2} \rfloor (\lfloor \frac{m}{2} \rfloor + 1) - d_m \lfloor \frac{m}{2} \rfloor ((m-1) \mod 2)$). Other choices could result in a different cost, since the number of shared links on the tree could be different.

The cost of omniscient multicast C_o is the sum of the costs of n trees, one rooted at each source. If we denote by $C(T_j)$ the cost to transmit an event from source j, it turns out that we can express this cost in terms of T_1 as: $C(T_j) = C(T_1) + C(T_j - T_1) - C(T_1 - T_j)$. $C(T_j - T_k)$ is interpreted as the cost of transmission and reception along the tree formed by removing, from T_j, those links that are common to T_j and T_k. Furthermore, for ease of exposition, $C(T_j)$ can be expressed as the sum of two costs: the cost of transmission and reception along the horizontal links $H(T_j)$, and the analogous cost along the diagonal links $D(T_j)$.

We can then write C_o as follows:

$$C_o = \sum_{j=1}^{n} \Big\{ D(T_1) + H(T_j) + D(T_j - T_1) - D(T_1 - T_j) \Big\} \tag{2}$$

where

$$H(T_j) = 2\Big\{ \sqrt{N} - 1 - \Big(\lfloor \frac{j}{2} \rfloor d_n - min(\lfloor \lfloor \frac{j}{2} \rfloor \frac{d_n}{d_m} \rfloor, \lfloor \frac{m - (j \mod 2)}{2} \rfloor) d_m \Big) \Big\} \tag{3}$$

$$D(T_j - T_1) = 2\Big\{ \lceil \frac{m + (j \mod 2)}{2} \rceil \lfloor \frac{j}{2} \rfloor d_n + \sum_{l=1}^{min(\lfloor \lfloor \frac{j}{2} \rfloor \frac{d_n}{d_m} \rfloor, \lfloor \frac{m-(j \mod 2)}{2} \rfloor)} (d_n \lfloor \frac{j}{2} \rfloor - l d_m) \Big\} \tag{4}$$

$$D(T_1 - T_j) = 2\Big\{ \sum_{l=1}^{\lfloor \frac{m-(j \mod 2)}{2} \rfloor} min(d_n \lfloor \frac{j}{2} \rfloor, l d_m) \Big\} \tag{5}$$

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

3.3 Directed Diffusion

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

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

$$C_d = C(UT_{1\to n}) = C(T_1) + \sum_{j=2}^{n} \Big\{ H(T_j - UT_{1\to(j-1)}) + D(T_j - UT_{1\to(j-1)}) \Big\} \tag{6}$$

where

$$H(T_j - UT_{1\to(j-1)}) = H(T_j) \tag{7}$$

$$D(T_j - UT_{1\to(j-1)}) = 2\Big\{ \lceil \frac{m + (j \mod 2)}{2} \rceil d_n + \sum_{l=1}^{min(\lfloor \lfloor \frac{j}{2} \rfloor \frac{d_n}{d_m} \rfloor, \lfloor \frac{m-(j \mod 2)}{2} \rfloor)} min(d_n, d_n \lfloor \frac{j}{2} \rfloor - l d_m) \Big\} \tag{8}$$

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

3.4 Comparison

The data delivery cost of flooding C_f is several orders of magnitude higher than that of omniscient multicast C_o. However, C_o is still higher than the diffusion cost C_d because $D(T_1) - D(T_1 - T_j) \geq 0$ and $D(T_j - T_1) \geq D(T_j - UT_{1\to(j-1)})$. To validate this reasoning, the data delivery cost (normalized by network size) for directed diffusion and omniscient multicast is plotted using various parameters (*i.e.*, N, m, and n). As the number of sources and sinks increases (Figure 6(a) and 6(b)), the cost saving due to in-network processing (*e.g.*, duplicate suppression) of diffusion becomes more evident (given that C_d increases at a lower rate than C_o).

Of particular interest is the plot of cost versus network size (Figure 6(c)). Since diffusion can suppress application-level duplicates, one would expect that C_o is merely nC_d. The main reason this does not hold is that our analysis somewhat conservatively estimates diffusion costs. In practice, diffusion would have negatively reinforced several of the links that our analysis includes.

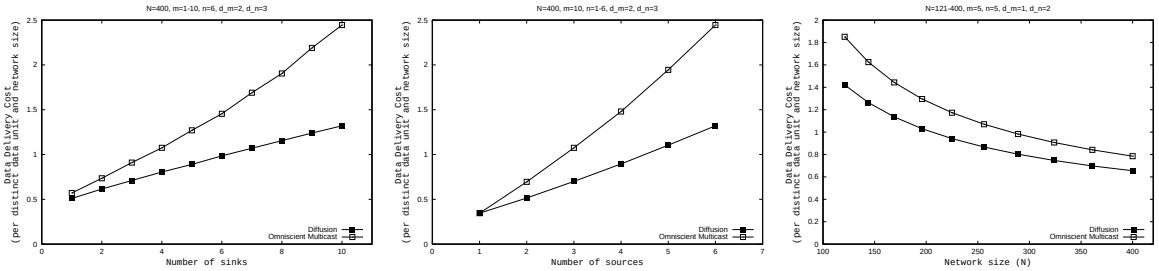
4 Simulation

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 analysis of Section 3. This section describes our methodology, compares the performance of diffusion against some idealized schemes, then considers impact of network dynamics on simulation.

4.1 Goals, Metrics, and Methodology

We implemented our 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 four-fold: First, verify and complement our analytic evaluation. Second, understand the impact of dynamics—such as node failures—on diffusion. Third, explore the influence of the radio MAC layer on diffusion performance. Finally, study the sensitivity of directed diffusion performance to the choice of parameters.

We choose three metrics to analyze the performance of directed diffusion and to compare it to other schemes: average



(a) Impact of the number of sinks

(b) Impact of the number of sources

(c) Impact of network size

Figure 6: Impact of various parameters on directed diffusion and omniscient multicast.

dissipated energy, average delay, and distinct-event delivery ratio. **Average dissipated energy** measures the ratio of total dissipated energy *per node* in the network to the number of *distinct* events seen by sinks. This metric computes the average work done by a node in delivering useful tracking information to the sinks. The metric also indicates the overall lifetime of sensor nodes. **Average delay** measures the average one-way latency observed between transmitting an event and receiving it at each sink. This metric defines the temporal accuracy of the location estimates delivered by the sensor network. **Distinct-event delivery ratio** is the ratio of the number of distinct events received to the number originally sent. A similar metric was used in earlier work to compare ad-hoc routing schemes [4]. We study these metrics as a function of sensor network size.

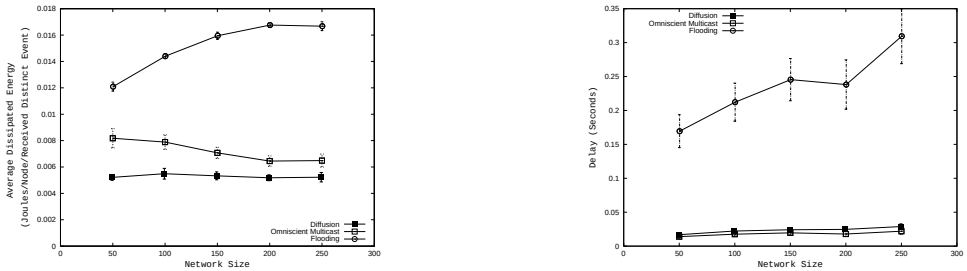
In order to study the performance of diffusion as a function of network size, we generate a variety of sensor fields of different sizes. In each of our experiments, we study five different sensor fields, ranging from 50 to 250 nodes in increments of 50 nodes. Our 50 node sensor field generated by randomly 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 1.6 Mb/s 802.11 MAC layer. Our simulations use a modified 802.11 MAC layer. To more closely mimic realistic sensor network radios [21], 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 for selecting a TDMA-style MAC for sensor networks rather than one using contention-based protocols [28]. 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 4.4, we analyze the impact of a MAC energy model in which listening for transmissions dissipates as much energy as receiving them.

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

4.2 Comparative Evaluation

Our first experiment compares diffusion to omniscient multicast and flooding scheme for data dissemination in networks. Figure 7(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.



(a) Average dissipated energy

(b) Average delay

Figure 7: Directed diffusion compared to flooding and omniscient multicast.

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

Figure 7(b) plots the average delay observed as a function of network size. Directed diffusion has a delay comparable to omniscient multicast. This is encouraging. To a first approximation, in an uncongested sensor network and in the absence of obstructions, the shortest path is also the lowest delay path. Thus, our reinforcement rules seem to be finding the low delay paths. However, the delay experienced by flooding is almost an order of magnitude higher than other schemes. This is an artifact of the MAC layer: to avoid broadcast collisions, a randomly chosen delay is imposed on all MAC broadcasts. Flooding uses MAC broadcasts exclusively. Diffusion only uses such broadcasts to propagate the initial interests. On a sensor radio that employs a TDMA MAC-layer, we might expect flooding to exhibit a delay comparable to the other schemes.

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

4.3 Impact of Dynamics

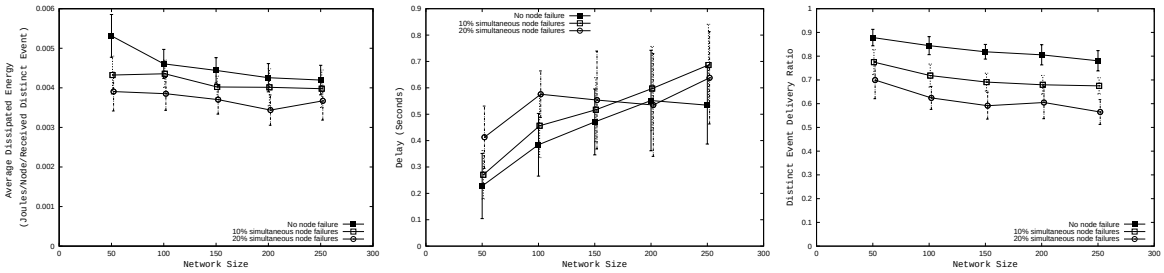
To study the impact of dynamics on directed diffusion, we simulated node failures as follows. For each sensor field, repeatedly turned off a fixed fraction (10 or 20%) of nodes for 30 seconds. These nodes were uniformly chosen from the sensor field, with the additional constraint that an equal fraction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

Our dynamics experiment imposes fairly adverse conditions for a data dissemination protocol. At any instant, 10 or 20 percent of the nodes in the network are unusable. Furthermore, we do not permit any "settling time" between node failures. Even so, diffusion is able to maintain reasonable, if not stellar, event delivery (Figure 8(c)) while incurring less than 20% additional average delay (Figure 8(b)). Moreover, the average dissipated energy actually *improves*, in some cases, in the presence of node failures. This is a bit counter-intuitive, since one would expect that directed diffusion would expend energy to find alternative paths. As it turns out, however, our negative reinforcement rules are conservative enough that several reinforced paths (high-quality paths) are kept alive in normal operation. Thus, at the levels of dynamics we simulate, diffusion doesn't need to do extra work. The lower energy dissipation results from the failure of some high-quality paths.

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

4.4 Impact of Data Aggregation and Negative Reinforcement

To explain what contributes to directed diffusion's energy efficiency, we now describe two separate experiments. In both of these experiments, we do not simulate node failures. First, we compute the energy efficiency of diffusion with and without aggregation. Recall from Section 4.2 that in our simulations, we implement a simple aggregation strategy, in which a node suppresses identical data sent by different sources. As Figure 9(b)



(a) Average dissipated energy (b) Average delay (c) Event Delivery Ratio

Figure 8: Impact of node failures on directed diffusion.

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.

The second mechanism whose benefits we quantify is negative reinforcement. This mechanism prunes off higher latency paths, and can contribute significantly to energy savings. In this experiment, we selectively turn off negative reinforcement and compare the performance of directed diffusion with and without reinforcement. Intuitively, one would expect negative reinforcement to contribute significantly to energy savings. Indeed, as Figure 9(a) shows, diffusion without negative reinforcement expends nearly twice as much energy as when negative reinforcement is employed. This suggests that even our conservative negative reinforcement rules prune off paths which deliver consistently higher latency.

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

4.5 Sensitivity Analysis

Finally, we evaluate the sensitivity of our comparisons (Section 4.2) 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 [18].

In our comparisons, we selected radio power dissipation parameters to more closely mimic realistic sensor radios [21]. We re-ran the comparisons of Section 4.2, but with power dissipation comparable to the AT&T Wavelan: 1.6W transmission, 1.2W reception and 1.15W idle [30]. In this case, as Figure 9(c) shows, the distinction between the schemes disappears. In this regime, we are better off flooding all events. This is because idle time energy utilization completely dominates the performance of all schemes. This is the reason why sensor radios try very hard to minimize listening for transmissions.

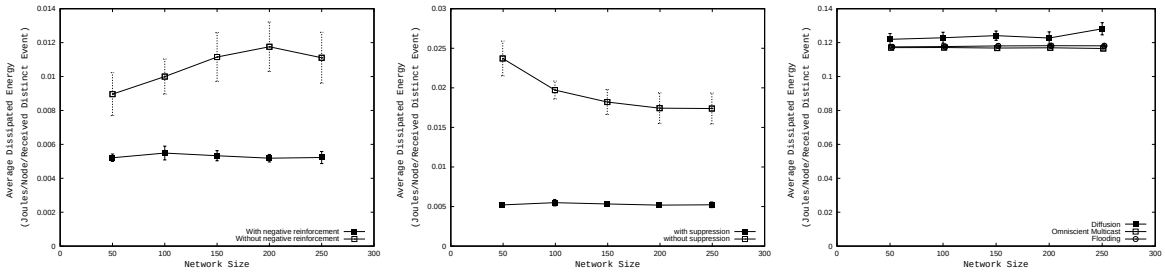
5 Implementation

We have so far described directed diffusion using a specific application as an example. However, it is desirable to avoid reimplementing diffusion mechanisms for every new application. To this end, we have implemented a generic diffusion substrate and ported this code to multiple platforms including WINSng 2.0 nodes, USC/ISI PC/104 nodes, Motes, and as a module in the *ns-2* simulator (the diffusion code can be downloaded from http://www.isi.edu/scadds/testbeds.html). On top of this substrate, several applications (*e.g.*, collaborative detection, nested query, adaptive fidelity) are developed in collaboration with researchers at BAE Systems, Cornell University, and Pennsylvania State University. Details of our platforms, applications, experiences, and how the attribute system affects applications are available elsewhere [14].

Our generic diffusion substrate exports two Application Programming Interfaces (APIs) [6]: a *network routing* API, and a *filter* API. The former is invoked on sources and sinks, while the latter enables in-network processing of events.

5.1 Network API

The network routing API is based on a publish/subscribe paradigm and was developed with Dan Coffin and Dan van Hook of MIT/Lincoln Laboratories. The interface supports two operations. Sinks can *subscribe* to named events, and sources can *publish* events. The diffusion substrate hides the details of how published data is delivered to subscribers, the routing algorithms we have described in Section 2. Events are sent and arrive asynchronously. When events arrive at a node, they trigger callbacks to relevant applications (those that have subscribed with matching attributes).



(a) Negative reinforcement (b) Duplicate suppression (c) High idle radio power

Figure 9: Impact of various factors on directed diffusion.

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

- the *key* indicates the semantics of the attribute (latitude, longitude, frequency).
- the *operator* describes how the attribute will match when two attributes are compared. Operators include equality, other simple comparisons (inequality, less than, greater than, etc.), and "attribute present" (equals-anything).
- the *type* indicates what algorithms to run when matching attributes. We have currently implemented 32-bit integer, both 32 and 64-bit floats, string and blob (uninterpreted) attributes.
- the *value* of the attribute (its data), and its *length* (if length is not implicit from the type)

This approach has the advantage of standardizing the syntax and structure of attributes, and allowing applications to reuse attribute handling and matching code.

5.2 Filter API

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

6 Related Work

Distributed sensor networks have begun to receive attentions during the last few years. However, our work has been informed and influenced by a variety of other research efforts, which we now describe.

Distributed sensor networks are a specific instance of ubiquitous computing as envisioned by Weiser [33]. Early ubiquitous computing efforts, however, did not approach the issues of scalable node coordination, focusing more on issues in the design and packaging of small, wireless devices. More recent efforts, such as WINS [28] and Piconet [3] considered networking and communication issues for small wireless devices. The WINS project made significant progress in identifying feasible radio designs for low-power environmental sensing. Their project has focused also on low-level network synchronization necessary for network self-assembly. Our directed diffusion primitives provide inter-node communication once network self-assembly is complete. The Piconet project is more focused on enabling home and office information discovery. Their work relies on centralized infrastructures rather than self-assembly networks.

In addition, recent efforts (*i.e.,* SPIN [24] and LEACH [15]) have pointed out some of the advantages of diffusion-like application-specificity in the context of sensor networks. Particularly, SPIN showed how embedding application semantics in flooding can help achieve energy-efficiency. LEACH and diffusion explore some of these same ideas in the context of more sophisticated distributed sensing algorithms. Specifically, LEACH can achieve energy savings by processing application-level data at its cluster heads whereas diffusion can process such data anywhere in the network.

Some of the inspiration for directed diffusion comes from biological metaphors, such as reaction-diffusion models for morphogenesis [31], and models of ant colony behavior [5].

Directed diffusion borrows heavily from the literature on ad-hoc unicast routing. Specifically, it is a close kin of the class of several reactive routing protocols proposed in the literature [20, 27, 26]. Of these, it is possibly closest to [26] in its attempt to localize repair of node failures, and its deemphasis of optimal routes. The differences between ad-hoc routing and directed diffusion have already been discussed in Section 2.5.

Directed diffusion is influenced by the design of multicast

routing protocols. In particular, propagation of reinforcements and negative reinforcements are similar to joins and prunes in shared-tree construction [10]. The initial interest dissemination and gradient setup is similar to data-driven shortest-path tree setup [9]. The difference, of course, is that where Internet protocols rely on underlying unicast routing to aid tree setup, diffusion cannot. Diffusion can, however, do in-network processing of data (caching and aggregation) unlike existing multicast routing schemes.

Finally, interest dissemination, data propagation and caching in directed diffusion are all similar to some of the ideas used in adaptive Web caching [35]. In these schemes, caches self-organize themselves into a hierarchy of cooperative caches through which requests for pages are effectively *diffused*.

7 Conclusions

In this paper, we described the directed diffusion paradigm for designing distributed sensing algorithms. There are several lessons we can draw from our preliminary evaluation of diffusion. First, directed diffusion has the potential for significant energy efficiency. Even with relatively unoptimized path selection, it outperforms an idealized traditional data dissemination scheme like omniscient multicast. Second, diffusion mechanisms are stable under the range of network dynamics considered in this paper. Finally, for directed diffusion to achieve its full potential, however, careful attention has to be paid to the design of sensor radio MAC layers.

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 ACM Symposium on Operating Systems Principles*, pages 186–201, Charleston, SC, 1999.

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

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

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

[6] 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.

[7] 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).

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

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

[10] S. E. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei. The PIM Architecture for Wide-Area Multicast Routing. *IEEE Transactions on Networking*, 4(2), April 1996.

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

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

[13] The Bluetooth Special Interest Group. Bluetooth v1.0B Specification. http://www.bluetooth.com, 1999.

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

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

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

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

[18] 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.

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

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

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

[22] E.D. Kaplan, editor. *Understanding GPS: Principles and Applications*. Artech House, 1996.

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

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

[25] D.L. Mills. Internet time synchronization: The network time protocol. In Z. Yang and T.A. Marsland, editors, *Global States and Time in Distributed Systems*. IEEE Computer Society Press, 1994.

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

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

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

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

[30] 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.

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

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

[33] M. Weiser. The Computer for the 21st Century. *Scientific American*, September 1991.
[34] Yan Yu, Ramesh Govindan, and Deborah Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. Technical Report UCLA/CSD-TR-01-0023, UCLA, May 2001.
[35] L. Zhang, S. Michel, S. Floyd, V. Jacobson, K. Nguyen, and A. Rosenstein. Adaptive Web Caching: Towards a New Global Caching Architecture. In *Proceedings of the Third International Caching Workshop*, June 1998.

Chalermek Intanagonwiwat He received his B. Eng. degree in Computer Engineering from the King Mongkut's Institute of Technology at Ladkrabang (KMITL), and his M.S. and Ph.D. degrees in Computer Science from the University of Southern California (USC). His Ph.D. dissertation is "Directed Diffusion: A Data-Centric Communication Paradigm for Wireless Sensor Networks". His research interests include neural networks and large-scale wireless networks of distributed embedded systems.

Ramesh Govindan Ramesh Govindan received his B. Tech. degree from the Indian Institute of Technology at Madras, and his M.S. and Ph.D. degrees from the University of California at Berkeley. He is an Associate Professor in the Computer Science Department of the University of Southern California. His research interests include Internet routing and topology, and sensor networks.

Deborah Estrin Deborah Estrin is a Professor of Computer Science at UCLA and Director of the Center for Embedded Networked Sensing (CENS), a National Science Foundation Science and Technology Center (http://cens.ucla.edu). She received her Ph.D. in Computer Science from MIT (1985). She has served on numerous program committees and editorial boards, including SIGCOMM, Mobicom, SOSP and IEEE/ACM Transactions on Networking. She is a Fellow of the ACM and of AAAS.

John Heidemann John Heidemann is a project leader at USC/ISI and a research assistant professor at USC. At ISI he investigates networking protocols and simulation as part of the SAMAN and CONSER projects and embedded networking and sensor networking as part of the SCADDS project. He received his B.S. from University of Nebraska-Lincoln and his M.S. and Ph.D. from UCLA, and is a member of ACM, IEEE, and Usenix.

Fabio Silva Fabio Silva (ACM S'98,M'01) received his B.S. degree in Electrical Engineering from University of Campinas, Brazil in 1998 and his M.S. degree in Electrical Engineering from University of Southern California (USC), Los Angeles, CA, in 1999. Fabio joined the Computer Networks Division at USC/ISI where he does research in scalable and distributed systems. He is currently the developer and maintainer of diffusion software. His research interests are in ad-hoc, power-efficient wireless communication protocols.