Minimum-Energy Asynchronous Dissemination to Mobile Sinks in Wireless Sensor Networks

Hyung Seok Kim*
School of Electrical Engineering
Seoul National University
Seoul, Republic of Korea
hskim@cisl.snu.ac.kr

Tarek F. Abdelzaher
Department of Computer Science
University of Virginia
Charlottesville, VA 22904
zaher@cs.virginia.edu

Wook Hyun Kwon
School of Electrical Engineering
Seoul National University
Seoul, Republic of Korea
whkwon@cisl.snu.ac.kr

ABSTRACT

Data dissemination from sources to sinks is one of the main functions in sensor networks. In this paper, we propose SEAD, a Scalable Energy-efficient Asynchronous Dissemination protocol, to minimize energy consumption in both building the dissemination tree and disseminating data to mobile sinks. SEAD considers the distance and the packet traffic rate among nodes to create near-optimal dissemination trees. The sinks can move without reporting their location to the tree while receiving data updates successfully. Our evaluation results illustrate that SEAD consumes less energy on building and maintaining a dissemination tree to multiple mobile sinks compared to other approaches such as directed diffusion, TTDD, and mobile ad hoc multicast.

Categories and Subject Descriptors

C.2.1 [**Computer-communication Networks**]: Network Architecture and Design—*wireless communication, distributed networks*; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

General Terms

Design

Keywords

Sensor network, asynchronous dissemination, minimum energy, mobility

1. INTRODUCTION

A sensor network is a multi-hop ad hoc wireless network of hundreds or thousands of unattended sensors. The sensor nodes collect useful information such as acoustic, light, and seismic measurements, and play a dual role as both data generators and routers. These sensor nodes communicate through wireless channels and are powered by limited disposable batteries. Data sources in sensor networks are usually locations where environmental activities of interest take place [6, 12, 8]. The monitoring terminals, called *sinks*, gathering the sensor readings, may be mobile PDAs carried by users or may be static access points. The sinks monitoring the sensor nodes may have different service requirements such as the desired data refresh rate and the end-to-end delay between the source and the sink. An example of a sensor network application is a group of mobile decontaminating robots or soldiers with wireless computing devices that use the sensor network for monitoring the chemical or radioactive contamination level of some region.

Energy is identified as the most crucial resource in sensor networks due to the difficulty of recharging batteries of thousands of devices in remote or hostile environments. The energy consumption of each sensor node is dominated by the cost of transmitting and receiving messages [22]. The importance of optimizing communication energy is supported by measurements from prototypes of sensor network devices such as MICA2 [7] and their predecessors [13]. When sinks are mobile in sensor networks, communication consists of three main parts: building the dissemination tree (d-tree), disseminating data, and maintaining linkage to mobile sinks. Our algorithm addresses energy savings at each of these three levels.

We propose a Scalable Energy-efficient Asynchronous Dissemination protocol (SEAD), a distributed self-organizing protocol that saves communication energy. It extends prior work [2] in that sinks are mobile and high network density is not assumed. Unlike overlay multicast [5, 10] in mobile ad hoc networks, SEAD does not use mobile sinks as intermediate members of the tree. This precludes frequent changes of the dissemination path due to sink mobility. When mobile sinks join the tree, SEAD does not use flooding to find an entry to the tree, which is in contrast to approaches such as [15] and [25]. A disadvantage of flooding is that it costs much energy and incurs unnecessary collisions. In SEAD, a stationary sensor node takes the mobile sink's place for building an optimal dissemination tree. Data dissemination paths to these stationary terminals are selected to minimize energy cost. As sinks move away from their terminals, the forwarding delay to the sink increases. A trade-off exists between minimizing that delay and saving energy spent on

*Work on this paper was conducted when H.S.Kim was a visiting researcher at University of Virginia.

SenSys'03, November 5–7, 2003, Los Angeles, California, USA.

reconfiguring the tree. In this paper, we show that it is possible to achieve considerable savings in power consumption expended on communication to mobile sinks at the expense of a moderate increase in path delay. Exploration of this trade-off is the main principle that underlies the design of our protocol.

The remainder of this paper is organized as follows. Section 2 presents the assumptions and basic service model. Sections 3 and 4 describe the SEAD protocols to construct and maintain the d-tree for mobile sinks and minimize energy consumption. A comparative performance evaluation using simulation is presented in Section 5. Section 6 reviews related work. The paper concludes with Section 7.

2. ASSUMPTIONS AND BASIC MODEL

This section presents the basic model of the sensor network which SEAD targets, where multiple *mobile* sinks receive sensor readings from the source at varying rates. The network model for SEAD makes the following basic assumptions:

- Each sensor node is assumed to be aware of its own geographic location. The network can use location services such as [4] and [1] to estimate the locations of the individual nodes. The location estimation does not require GPS at every node.
- After having been deployed, sensor nodes remain stationary at their initial locations.
- The sensor nodes are homogeneous and wireless channels are bidirectional. Each sensor node has a constrained battery energy.
- Sensor nodes communicate with sinks by delivering data across *multiple* hops. That is to say, sources and sinks are typically much further apart than a single radio radius.

2.1 Overview of the Algorithm

One source generates the sensory update traffic possibly on behalf of a group of local sensors. The update traffic is time-varying, depending on the volatility of the environment and the type of sensors involved. An environment that changes frequently will generate more update traffic than a quiescent environment. The average update rate of the source is denoted by U. The data updates are disseminated along a tree to the mobile sinks in an asynchronous manner. Each branch of the tree may have its own update rate depending on the desired refresh rate of the downstream observers. To detect failures or packet loss in the sensor network, a minimum update rate U_m is enforced. If a source has no new sensor readings, it disseminates *idle* messages along the tree at rate U_m. If a node in the tree receives no messages including *idle* messages from the source for a period longer than $1/U_m$, the node contacts its parent. If its parent has failed and gives no response, the node asks for a new parent by sending an error message to the source of the d-tree.

When a mobile sink wants to join the d-tree, it selects one of its neighboring sensor nodes to send a join query to the source of the tree. The selected sensor node is called the sink's *access node*. The access node is used to represent the moving sink when the optimal d-tree is built. Static access nodes amortize the overhead in the presence of mobility. Access nodes keep track of the current position of the corresponding mobile nodes. The tree delivers data to the fixed access node. In turn, the access node delivers the data to the sink without exporting the sink's location information to the rest of the tree. The tree is updated only when the access node changes (as opposed to every time some node moves). As the sink moves, no new access node is chosen until the hop count between the access node and the sink exceeds a threshold. The value of this threshold allows trade-offs to be made between path delay and energy spent on reconstructing the tree.

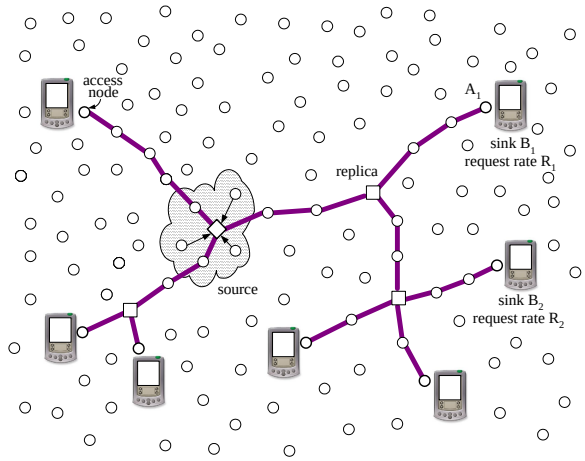


Figure 1: An example of the SEAD tree model in the sensor network.

The sensor network model consists of a set V of sensor nodes and a set B of sinks indexed by $n = 1, 2, ..., N$ and $m = 1, 2, ..., M$, respectively. Let $A = \{A_1, A_2, ..., A_M\} \subset V$ be the set of M *access nodes* for mobile sinks B = $\{B_1, B_2, ..., B_M\}$ which request data from the source at refresh rates $R = \{R_1, R_2, ..., R_M\}$. Source data is replicated at selected nodes between the source and sinks. We define a replica as a sensor node that stores a copy of the source data. The replica temporarily stores the latest data incoming from the source and asynchronously disseminates it to others along the tree. The replica needs only a very small amount of memory enough for a single data record. Replicas are members of the d-tree. The set of sources, the set of access nodes, and the set of replicas are included in the set V. An example of a d-tree using SEAD is shown in Figure 1.

The SEAD algorithm constructs and updates the d-tree dynamically. It is essentially composed of two main functions; one to add a node to the tree and one to remove a node from the tree. SEAD focuses on dissemination in which a source sends its data to multiple sinks. Dissemination has the source as the root of the d-tree as in multicast trees. If multiple sources are present in the sensor network, multiple dissemination trees will be constructed separately. There is limited room for optimization because data coming from different sources to the same receiver are in general different and cannot be aggregated without making the scheme application specific. A limited amount of application-independent data aggregation is possible and is addressed in a separate publication [11]. In this paper, we therefore do not consider such cross-tree optimizations further. In [11] the authors quantify the power savings achievable due to aggregation.

Suppose that sensor nodes $a, b \in V$ communicate with each other by delivering data across multiple hops. Let

$d(a,b)$ denote the distance between nodes a and b, and let $h(a,b)$ be the hop count between them. Nodes do not have global knowledge of the number of intermediate hops and hop lengths. However, when sensor nodes are uniformly distributed in the field, the hop count is closely related to the geographic distance. Our tree construction algorithm therefore uses geographic distance to estimate the hop count. We also use the geographic distance $d(a,b)$ for defining the energy cost, as the energy is proportional to the number of single hop broadcasts needed to propagate a message along the path. The energy consumed for communication between nodes a and b is also proportional to the packet length and packet sending rate between these two nodes. The packet length in our underlying platform is fixed. Therefore, the energy cost for multi-hop data transfer from a to b is proportional to the distance $d(a,b)$ between the two nodes multiplied by the packet sending rate P_{ab} :

$$Energy\_cost(a,b) \propto d(a,b)P_{ab}. \quad (1)$$

3. D-TREE CONSTRUCTION

When constructing the d-tree, SEAD attempts to construct a minimum-cost weighted Steiner tree. This is distinguished from the more commonly used minimum spanning tree problem in that it is permitted to construct or select replicas at intermediate points (other than the source and sinks) to reduce the cost of the tree. Members of the tree interact with each other to deliver content to the sinks. Each node on the d-tree rooted at the source maintains a pointer to its parent as well as to each of its children. SEAD is an overlay network that sits on top of location-based routing protocols such as simple geographical forwarding (GF). Communication between nodes follows the underlying routing protocol. SEAD locates intermediate destinations for the packets, or replicas.

The SEAD protocol consists of four phases: *subscription query*, *gate replica search*, *replica placement*, and *d-tree management*. At the subscription query phase, a sink directs a join query to the source via its access node. At the gate replica search phase, a gate replica is determined, which serves as the grafting point (on the existing tree) from which a branch to the new access point is extended. The replica placement phase locally readjusts the tree in the neighborhood of the gate replica to further reduce communication energy. The constructed tree is managed to accommodate mobile sinks or defective regions such as a group of congested or failed nodes. The following sections describe details of each of the first three phases. The d-tree management is presented in Section 4.

3.1 Subscription Query

After being deployed, each static sensor node finds out its neighbors. Two nodes are said to be neighbors if they can communicate directly (i.e., within a single hop). If a node receives no response from a previously recorded neighbor, the node removes it from its neighbor table. Mobile sinks beacon periodically to determine their neighbors.

A mobile sink B_i selects the nearest of its adjacent nodes as the access node A_i right before the sink B_i sends a *join* query to a source via the access node. The access node directly sends the join query to the source via the underlying routing protocol. The access node is a stationary sensor node, which represents the mobile sink when constructing the d-tree. The join query message contains the location of the access node A_i and the sink's desired update rate R_i.

3.2 Gate Replica Search

When extending a d-tree, the resulting performance depends mostly upon the replica chosen to feed the new access node in the current d-tree (i.e., upon the choice of the gate replica). Most multicast models [6, 5] in wireless ad hoc networks exploit *flooding* in order to find a current tree member close to the sink. Those models are not cost-efficient because flooding consumes much energy and causes unnecessary collisions. Also, when the desired refresh rates of the sinks are varied, the geographically closest node is not always the best choice for feeding the sink. Both proximity and desired sink refresh rate must be considered in connecting the new access node to the d-tree. A gate replica should be chosen which offers the least cost increase after it is connected to the access node. We propose a search algorithm to find the gate replica effectively.

In our algorithm, each node n in the tree has a set $C(n)$ of children. It maintains a downstream rate Q_c^n for each child $c \in C(n)$. The node n should send data to each child c at rate Q_c^n, which is the maximum among the refresh rates requested by the sinks served from the downstream branches. The algorithm starts when a source receives a query indicating a sink's desired refresh rate, R_i.

Each level of replica r, including the source receiving that message, runs a recursive search as follows. If the node has a parent and the downstream rate $Q_r^{p(r)}$ of the node r's parent $p(r)$ is lower than the desired update rate R_i, it is changed to R_i. Let E_r denote the set of r's ancestors. When a replica r is connected to the access node A_i, the additional cost, $K(r)$, of connecting the access point to r is calculated as

$$K(r) = R_i d(r, A_i) + \sum_{m \in E_r} \|R_i - Q_m^{p(m)}\| d(p(m), m) \quad (2)$$

where

$$\|z\| = \begin{cases} z, & if \quad z \geq 0 \\ 0, & if \quad z < 0 \end{cases} \quad (3)$$

and $p(m)$ is the replica m's parent. This cost can be calculated recursively. Consider the incremental cost, $K(r) - K(c)$ where r is c's parent, i.e., $r = p(c)$. This cost is calculated as follows:

$$\begin{aligned} K(r) - K(c) = & \quad (4) \\ & R_i d(r, A_i) + \sum_{m \in E_r} \|R_i - Q_m^{p(m)}\| d(p(m), m) \\ & - R_i d(c, A_i) - \sum_{m \in E_c} \|R_i - Q_m^{p(m)}\| d(p(m), m) \\ = & \quad R_i d(r, A_i) - R_i d(c, A_i) - \|R_i - Q_c^{p(c)}\| d(p(c), c) \\ = & \quad R_i d(r, A_i) - R_i d(c, A_i) - \|R_i - Q_c^{r}\| d(r, c) \end{aligned}$$

which relates the cost at node r to that at its child c. If the request rate R_i is not larger than the downstream rate Q_c^r of the child, the second term in (4) is zero. The node r calculates $K(r) - K(c)$ for each of its children c. If all the results for $K(r)$ are less than zero, that is, $K(r)$ is less than $K(c)$ for all the children $C(r)$, this recursive forwarding terminates and the replica r becomes the gate replica. If not, it forwards the message to a child with the least overhead $K(c)$, in other words, the child that maximizes $K(r) - K(c)$.

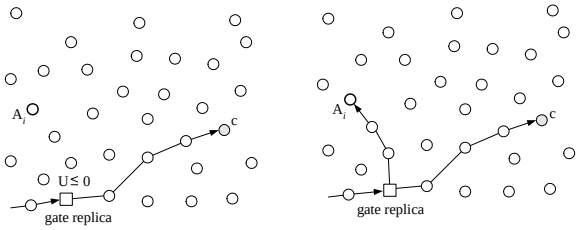
If the message approaches a leaf node (another access node), the access node's parent is set to the gate replica. Because locations of the mobile sink and its access node are changed as time goes on, they cannot be used as the gate replica. The gate replica search algorithm is formalized as follows:

procedure GateReplicaSearch(R_i, A_i)
Compute (4) for each child with R_i and A_i.
if results of (4) < 0 for all
children **or** all children are leaf nodes **then**
This node becomes the gate replica.
else
Forward *search* to the child with larger (4).
end

Observe that SEAD does not use hop count in the computation of path cost. Instead, it estimates cost based on physical distance. This approximation is used because costs of alternate paths are compared before messages are transmitted when searching for a gate replica. At that point in time, information regarding exact hop counts is not available.

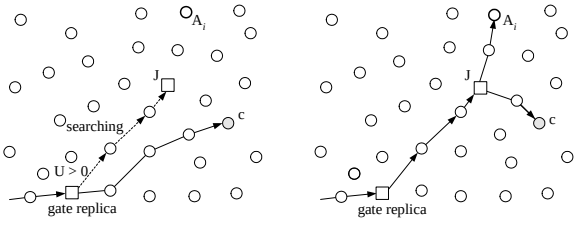
3.3 Replica Placement

The objective of the replica placement algorithm is to locally adjust the tree around the gate replica to produce an optimal d-tree from the source to the access node.



(a) gate replica computing U (b) non-replica mode

Figure 2: Connection in the non-replica mode.



(a) searching a junction replica J (b) juction replica mode

Figure 3: Searching a new replica in the junction replica mode.

The gate replica always has at least one child. There are two ways to connect the access node to the gate replica. One is to connect it as a child of the gate replica. This option adds no replicas to the tree, and is called *non-replica mode*. The other is to create a child for the gate replica to feed the access node and some of the gate replica's original children. Let us call the new child replica a *junction replica* and this connection option the *junction mode*. Which of the two options is better depends upon node locations and rate limitations. The replica placement phase compares the non-replica mode cost $U_{non\_replica}$ to a junction replica mode cost $U_{jreplica}$ and selects the better option so that the access node joins the tree in a way that minimizes the energy cost. If a junction replica that minimizes the cost exists, the gate replica connects one of its children and the access node to it. Otherwise, the gate replica makes the access node its child without a junction replica.

When the source has at least one child, the gate replica g calculates the cost of the non-replica mode $U_{non\_replica}(c)$ for each child $c \in C(g)$ as

$$U_{non\_replica}(c) = d(g, A_i)R_i + d(g, c)Q_c^g \quad (5)$$

which represents the energy cost when the gate replica g is parent of the access node A_i as shown in Figure 2. The energy cost equation (5) follows from the energy cost equation (1). The gate replica g finds the neighbor node n among its adjacent nodes within a singe hop range. Then it calculates the energy cost $U_{jreplica}(c)$ for each child c as

$$U_{jreplica}(c) = \min_{n \in W}\{d(g,n)\max(R_i, Q_c^g) + d(n, A_i)R_i + d(n,c)Q_c^g\}. \quad (6)$$

where W is a set of neighbors. After getting both $U_{nonreplica}(c)$ and $U_{jreplica}(c)$, the gate replica finds one of its children $c \in C(g)$ to maximize

$$U = U_{non\_replica}(c) - U_{jreplica}(c) \quad (7)$$

If the maximum value of U is not positive, a junction replica is not created and the gate replica is connected directly to the access node. This mode is shown in Figure 2. Otherwise, the gate replica finds child c and neighbor n that maximizes U. The child c becomes the sibling of the access node A_i. A *jreplica_search* message indicating node c is forwarded to the neighbor n. Upon receiving the message, node n repeats the aforementioned process relative to its neighbors and forwards the message recursively.

During the recursive forwarding, if no neighbor is found to make $U_{jreplica}$ smaller, the current node becomes the new junction replica J. If dead ends are met, a control message is sent upstream to make the previous node become a junction replica. The replica J stores the downstream rate Q_c^J and the desired update rate R_i of the sink i. It registers the gate replica g as its parent and the access node A_i as its child. The gate replica g replaces the propagation rate Q_J^g with the result of $\max\{R_i, Q_c^g\}$. This *junction replica* mode is shown in Figure 3. The denser the sensor network is, the more chances there are that a better replica can be found. The replica placement algorithm is formalized as follows:

procedure ReplicaPlacement(c, R_i, A_i)
if this node is a gate replica **then**
Compute $U_{non\_replica}$ from (5) for each child.
Compute $U_{jreplica}$ and n from (6) for each child.
Find a child c to maximize U from (7).
else
Compute n, U from (6) and (7) with given c.
if $U < 0$ **then**

This node becomes a junction replica or non-replica mode.
else
Forward *jreplica_search* message indicating c to the neighbor
n.
end

Observe that while asynchronous multicast for sensor networks has also been addressed in [2], the latter does not support mobility. Moreover, it solves a weighted Steiner tree problem in a continuous Euclidean space then attempts to map the resulting Steiner points onto locations of sensor nodes. It is implicitly assumed that a sensor can be found sufficiently close at any arbitrary point in the Euclidean space. This assumption becomes increasingly inaccurate when the network density is decreased. In contrast, the algorithm presented in this paper considers only existing sensors for possible replica locations. Hence, the protocol does not make inherent assumptions on network density and will work in the presence of voids or large gaps between sensor nodes.

4. D-TREE MANAGEMENT

4.1 Sink Mobility

The second part of the protocol lies in maintaining connectivity between mobile sinks and their access nodes. Figure 4 demonstrates this algorithm. As mentioned earlier, the sink selects one neighbor as its access node as shown in Figure 4(a). The sink renews the nearest node to communicate with its AR periodically. The access node is informed of the identity of the new nearest node using a *PathSetup* message as shown in Figure 4(a-c).

Two further cases are expected to occur during the sink's movement. The first case is collapsing the forwarding path as shown in Figure 4(d). When the sink migrates towards node D, where D is closer to an earlier node on the stored forwarding path (e.g., node A), the sink commands node D to set up the path with node A. Node D receives data from node C for a while to avoid losing data that has already been sent via the old path. If multiple nodes (e.g., node B and node C) are within a hop length from node D, the sink selects the one that appears earlier on the forwarding path (i.e., node B) for connection to node D.

The second case is to replace the existing access node with a new access node when the sink moves far enough away. End-to-end delay and hop count are considered for the new access node. The end-to-end delay D_i is defined as the delay it takes for data from the source S to arrive at the sink B_i. Delay D_i is calculated as

$$\begin{aligned} D_i &= h(S,B_i)\rho_t P + h(S,B_i)(\text{AQD}) \\ &+ \sum(\text{each hop distance})\rho_p \end{aligned} \quad (8)$$

where P is the data packet length and AQD is the average queuing delay per hop. The first term of (8) is transmission delay and ρ_t(sec/bytes) is a constant. The second term is round-trip propagation delay and ρ_p(sec/km) is a constant. The propagation delay is negligible because the sensor field size will typically not be more than tens of kilometers and the transmission speed of the sensor network is low compared to the speed of light. Therefore, the delay D_i is proportional to the hop count between nodes in the tree.

$$D_i \simeq h(S,B_i)P\rho_t + h(S,B_i)(\text{AQD}) \propto h(S,B_i). \quad (9)$$

The hop count between the source and the access node is fixed unless a sensor node fails, but the hop count between the sink and its access node increases or decreases as the mobile sink moves. The algorithm decides whether or not to change the access node based on total hop count. Equivalently, if the hop count between the sink and its access node goes beyond a threshold H_i, it replaces its access node A_{old} with new one A_{new}. The sink selects A_{new} among its neighbors as shown in Figure 4(e). Before changing the access node, the sink executes a simple hello protocol and selects the new access node. The new access node joins the d-tree. Then, the sink sends *disconnect* to the old access node. When the old access node leaves the tree, it notifies its parent of the fact.

H_i offers a configurable trade-off between maximum delay and power consumption. In general, the cost of changing the access node is higher than the cost of forwarding the data a few additional hops. Hence, increasing H_i saves on tree reconstruction energy at the expense of increasing end-to-end delay. Eventually, changing the access node provides a shorter path to the sink thereby minimizing long-term energy consumption. The amount of savings depends on the motion pattern. If the motion of the sink is radially away from the source, changing the access node may not curtail the path significantly. On the other hand, if the motion is circular around the source, changing the access node can save a lot.

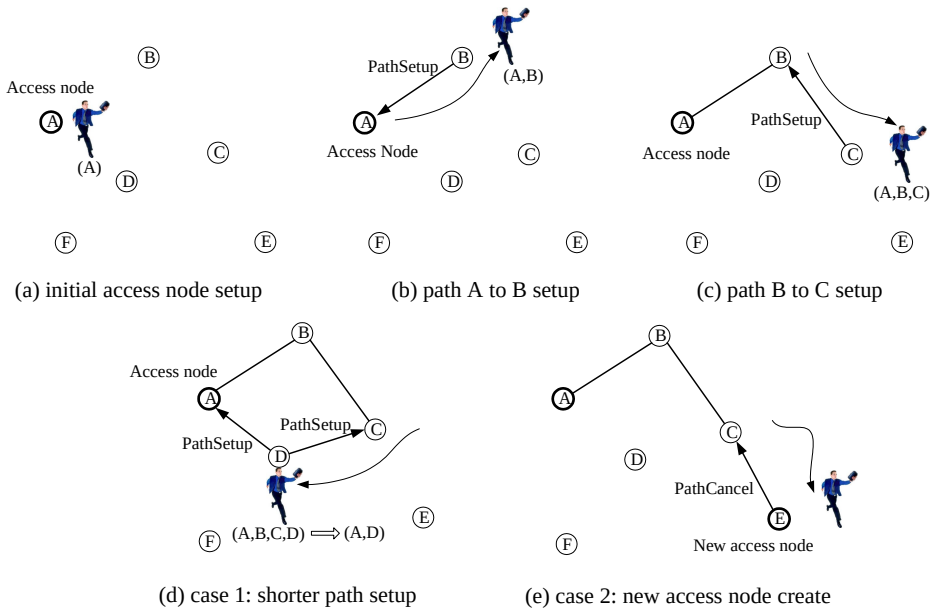
When the access node location is changed, the replica placement phase runs again in order to try to reconstruct the dissemination tree. As an optimization, if distance between the new access node and the old gate replica increases by less than a threshold T_m, the new access node A_{new} is connected to the old gate replica with no further modifications to the tree. If threshold T_m is exceeded, the sink sends its new position to the source to find the new gate replica as described in Section 3. The optimization prevents the gate replica search from occurring too frequently.

4.2 Leaving D-Tree

A sink B_i leaves the d-tree by sending a *leave* message to its access node. The access node A_i requests its parent to delete A_i from its list of children and stop forwarding data to it as shown in Figure 5(a). If a single child is left, the parent replica is removed and its parent takes over the child. This is shown in Figure 5(b). If more than one child are left, the location of the replica is adjusted using the replica placement protocol described earlier.

4.3 System Lifetime

When sinks are stationary, sensor nodes report data to sinks along the same path repeatedly. Therefore, members in the tree consume more energy than non-members, which may eventually cause network partitioning. To increase the system lifetime, energy consumption should not concentrate on some nodes to the exclusion of others. The SEAD model has mobile sinks. Thus the d-tree structure keeps altering gradually. If the mobility pattern is well-balanced in the field, the system lifetime may not be a critical problem. For a longer system lifetime, fanout of junction nodes can be

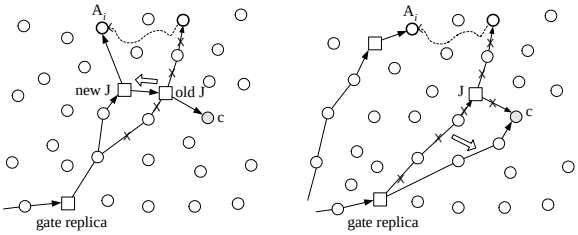


(a) initial access node setup (b) path A to B setup (c) path B to C setup

(d) case 1: shorter path setup (e) case 2: new access node create

Figure 4: Construction of path from access node to mobile sink in the sensor network.

restricted. A replica has to transmit data to each child. Thus the replica that has more children may consume more energy. Suppose that the fanout of the node is restricted to three and a new sink's gate replica has three children. The gate replica will not select the non-replica mode. Instead, it will choose the junction replica mode for connection to the new sink, thereby not increasing its fanout. Failure of the source is another factor to consider in system lifetime. We assume that the source location generally contains multiple sensors. Thus, if the sending node at the source fails, it will be replaced by a nearby sensor node.



(a) jucton replica adjustment (b) gate replica hand-off

Figure 5: Readjusting replica considering sink mobility.

4.4 Avoiding Defective Points or Areas

Because SEAD builds an application-layer overlay network, sensor node failures are resolved by the underlying routing protocols. Advanced greedy forwarding such as GPSR, GEAR [26], SLURP, and LAR are available for the routing layer. When a single or a few nodes fail for some reason, the routing layer can make packets go around the void. However, some greedy routing protocols forward packets by geographic location, without consideration to possible voids. SEAD can overcome this problem by changing the topology of the tree. For example, if one replica cannot send data to a current child, the replica assigns a reachable node to take over the communication thereby avoiding failed nodes or network voids.

4.5 Complexity and Resource Requirements

The description of the SEAD protocol may at first appear to be too complex to implement on low-power motes. Sensor nodes usually have scanty resources. For example, MICA2 motes [7] are equipped with a 4 MHz microprocessor, 4 kB of data memory, and 128 kB of code space. In this subsection we argue that at its core, our algorithm does not impose severe resource requirements. First, since sinks are laptops or PDAs with much more resources (e.g., memory, energy, speed), the algorithm exploits them to maintain forwarding chains from gate replicas to sinks. To maintain the dissemination tree itself, each replica needs to store only its children, a single parent, and the downstream rates for children. The number of children is usually no more than three or four (a junction replica mode will be selected for more children creating new junction replicas as opposed to new branches). Therefore, no burden is incurred on memory space. The replica placement algorithm finds a path by forwarding one message recursively from parent to best child. This approach has a lower complexity compared to involving all children in the computation. Greedy forwarding makes each node maintain a neighbor list. If the network is so dense that the number of neighbors is too large, dominant neighbors for each direction can be selected to save memory space. Hence, during normal tree operation, the memory and processing overheads on individual motes are minimal, making this algorithm a viable candidate for practical implementation.

5. EVALUATION

We evaluated SEAD using extensive simulation on NS-2. We use the specification of MICA2 [7], a popular sensor node prototype, to make the simulation adhere to the real

hardware parameters of sensor networks. A MICA2 mote is a wireless device that runs TinyOS; a small, open source, operating system developed by UC Berkeley for sensor networks. Its current sensor options include light, temperature, magnetic, acoustic, and barometric pressure. Power is supplied via disposable AA batteries. Our simulation follows the power consumption model of the MICA2 motes that requires about 0.080W for transmitting and about 0.025W for receiving. The transceiver in the simulation has a $250m$ radio range at 433 MHz, which is the case with the radio transceiver of a MICA2 mote.

The SEAD algorithm and several pre-existing protocols are simulated using the NS-2.26 simulator (http: //www.isi.edu/nsnam). Our goal in simulating SEAD is to test how well it actually conserves energy. The sensor network in our simulation consists of ($300 \leq N \leq 500$) sensor nodes in a $2000m \times 2000m$ grid or 3-5 nodes per $200m \times 200m$. We use the two-ray ground model as the radio propagation model and an omni-directional antenna having unity gain in the simulation. Each query packet is 36 bytes long and the data packet has 64 bytes to facilitate comparisons with other protocols. The default number of sinks is 8. Three different sources generate different data at an average interval of 6 seconds. The sinks send queries to the sources one after another to join their trees. The desired update rates are generated at random. The energy consumption is measured in terms of Joules per node. The default mobile sink speed is set to 10 m/sec (i.e., the fastest human speed).

Estimating positions of nodes occurs at initialization. It is not uncommon to assume in sensor networks that nodes know their location because such knowledge is needed for common sensor network applications such as tracking. Hence, the overhead of acquiring location knowledge would be incurred anyway (for application-related reasons) and is not introduced by our protocol. Once location estimates are acquired, no further cost is incurred. The simulation uses greedy forwarding as a routing protocol, based on the estimated positions of nodes. It is appropriate for the sensor network and has been used in other work [21, 25]. It makes a greedy decision to forward a packet to a neighbor if it has the shortest geographic distance to the destination among all neighbors and is closer to the destination than the forwarding node. If a node cannot find such a neighbor, it uses local flooding to find a next hop with a route to the destination.

To evaluate the performance of SEAD, we compare it to Directed Diffusion (DD) [15]. We also compare SEAD to TTDD [25] for sensor networks, as well as a multicast protocol, ADMR [16], for mobile ad hoc networks. DD code can be obtained from the NS-2 package and others are taken from a related web site (http://irl.cs.ucla.edu /GRAB), and the Monarch project (http://www.monarch. cs.rice.edu/multicast_extensions.html), respectively. Section 6 related work describes these protocols.

We use two main metrics to evaluate the performance of SEAD; namely, energy consumption per node, and average end-to-end delay. Energy consumption includes that of tree construction, data dissemination, and the sink mobility management. Our main concern is how well the dissemination tree is built for energy saving by considering the length of paths and data-disseminating rates. We therefore mainly focus on energy consumption for data dissemination and sink mobility management. Delivering a query to the source and the tree construction are conducted for the initial short period.

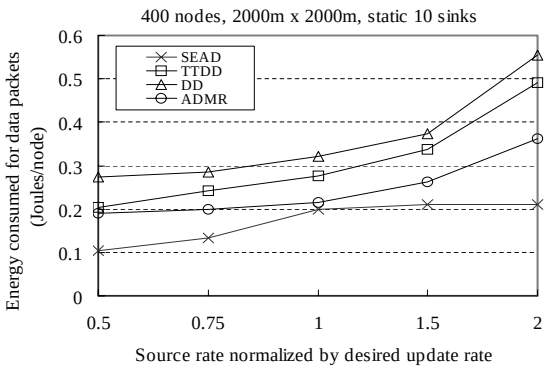


Figure 6: Energy dissipated for data packets versus the source rate normalized by the desired update rate.

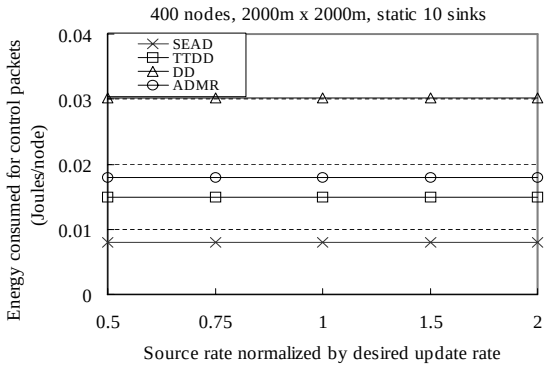


Figure 7: Energy consumed for control packets versus the source rate normalized by the desired update rate.

The average end-to-end delay is measured by averaging delay from the source to all sinks. Residual energy distribution and delivery success ratio are also investigated. Residual energy distribution indicates the percentage of nodes with the same residual energy levels, and is a measure of fairness in energy consumption and system lifetime. The success ratio is the ratio of the number of successfully delivered data messages to the number of the data messages that should have been delivered according the sinks' refresh rates. Sink mobility follows a random waypoint model [9].

5.1 Asynchronous Dissemination

First, we evaluate the performance of asynchronous dissemination. The sensor field is set to 400 sensor nodes in a $2000m \times 2000m$ area, and the number of sinks is set to 10. In this experiment, sinks do not move to exclude the effect of sink mobility thereby picking out the performance of asynchronous dissemination. We vary the source update rate normalized by the desired average refresh rate of sinks. The desired average refresh rate is set to 6. The source update rate is varied from 3 to 12.

Figure 6 shows the graph of the energy consumption for

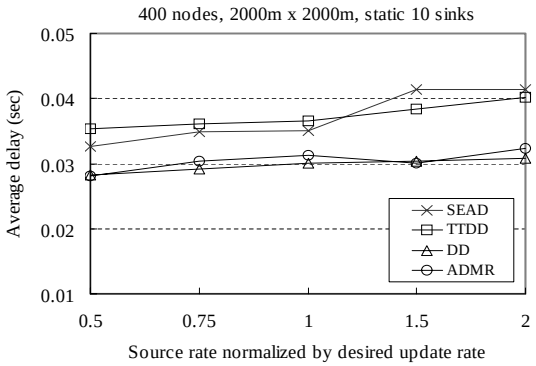


Figure 8: Average delay versus the source rate normalized by the desired update rate.

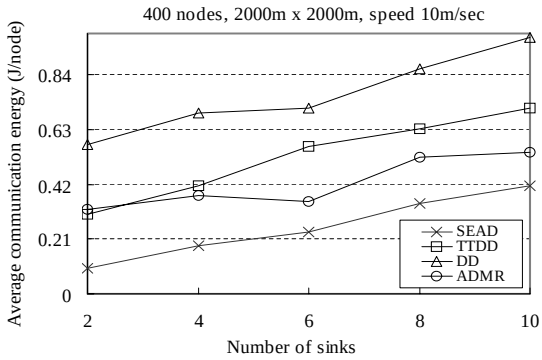


Figure 9: Energy consumption for the number of sinks.

data packets. It is observed that SEAD consumes less energy per node than DD, TTDD, and ADMR. When the source rate exceeds the requested refresh rate, the energy consumption of SEAD stops increasing because asynchronous dissemination caps the propagated update rate by the desired refresh rate of the receivers. Choosing the appropriate gate replica and placing the junction replica at the optimal position make the data-disseminating paths shortest in SEAD. We pick out energy consumption for control packets such as query or flooding. This also makes us discern energy consumed for data packets. Protocols other than SEAD consume more energy when sinks join the tree because of flooding or grid construction. DD uses query (control packets) flooding over the whole network and sends data packets through multiple paths until it finds the best path. SEAD finds the gate replica without searching the whole network or the whole tree thereby reducing the energy overhead of control packets. Figure 7 illustrates the fraction of total energy consumed by the control packets. When compared to data packets, the energy consumption for control packets is so small that it occupies little fraction of the total energy consumption in this simulation.

Figure 8 shows the average end-to-end delay at different source rates normalized by desired update rates. DD has the shortest delay because it finds unicast paths between a source and each sink without considering multicast. TTDD uses a grid structure, so its dissemination path tends to be longer than other protocols. SEAD makes junction replicas to save energy and does not use sinks as gate replicas. Thus its delay is comparable to that of TTDD. Comparing Figure 6 and Figure 8, we can appreciate the trade-off between energy and delay offered by SEAD. Our protocol is mainly concerned with saving energy to increase lifetime. We therefore deem the slight increase in message communication delay acceptable. The above energy-delay trade-off is more clearly manifested in the case of mobile sinks presented next.

5.2 Sink Mobility

SEAD is designed for mobile sinks. In this section, we evaluate the impact of sink mobility on the performance of SEAD. The sensor field has 400 sensor nodes as before. Both the average refresh rate and the average source update rate are set to 6. The hop count threshold, H_i, is set to two

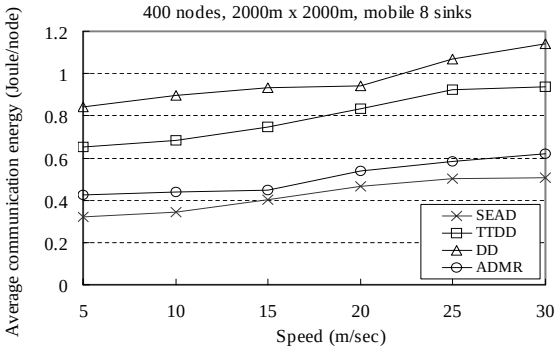


Figure 10: Energy consumption with different sink speeds.

hops. If a sink is more than two hops away from its access node, it replaces its access node with a new one. That prevents sink mobility from increasing the delay beyond a given value. Figure 9 shows energy consumption as the number of sinks is varied, when all sinks move at an average speed of 10 m/sec. In this figure, SEAD demonstrates better energy consumption than the other protocols because of better energy efficiency at each stage. The dissemination path is based on a Steiner tree to minimize the energy cost. Mobility is handled efficiently without excessive tree restructuring. In contrast, in DD, the new location of the mobile sink is propagated throughout the sensor field in order for all sensor nodes to get the sink's location. TTDD rebuilds a new multi-hop path between the sink and the grid. SEAD merely appends a new hop to the existing path thus minimizing perturbations and increasing energy savings as shown in Figure 9.

In order to examine the impact of sink speed, average energy consumption and success ratio are measured for different sink speeds (5 to 30 m/sec) in a sensor network consisting of 8 mobile sinks and 400 sensor nodes. Figure 10 plots the energy consumption as the sinks' speed changes. Observe that SEAD has the best performance at both low and high speeds of sinks.

Figure 11 is a graph showing the distribution of remaining energy for each protocol. Eight sinks are moving at a speed of 10m/sec. The d-tree is dynamically rebuilt when-

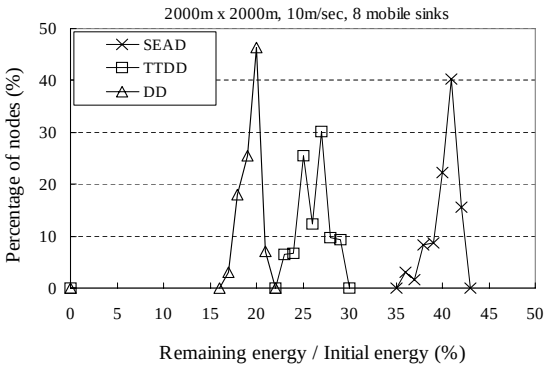


Figure 11: Remaining energy distribution of the sensor nodes.

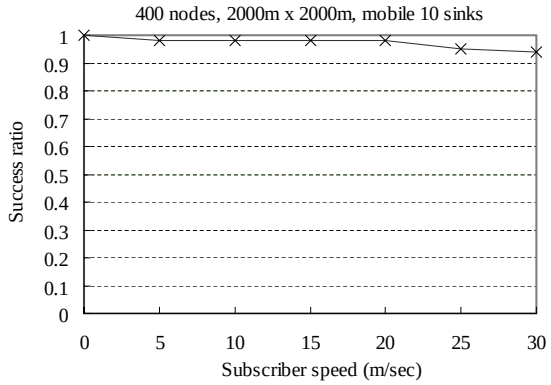


Figure 12: Success ratio of mobile sinks.

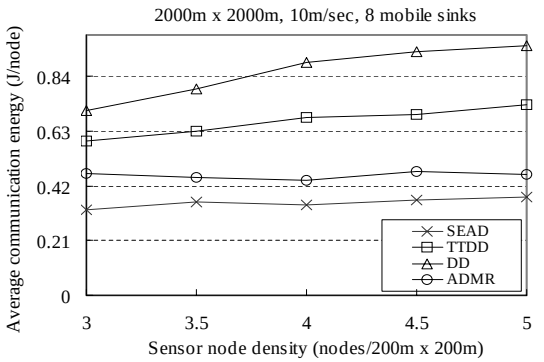


Figure 13: Energy per node with different sensor node density.

ever necessary. Because sinks move within the simulated field, energy consumption does not concentrate on several nodes. DD, which uses flooding to reconnect mobile sinks to the tree spends more energy, but the distribution of remaining energy has a very small variance indicating that energy consumption is fair over the whole network. As shown in Figure 11, the remaining energy with DD is distributed within a narrow band between 17%-21%. In the presence of mobility, TTDD and SEAD have advantages over DD. TTDD intrinsically has longer dissemination paths but hot areas are possible, unlike DD, due to local flooding. Although SEAD is saving more energy per node, it does not distribute energy consumption fairly. The distribution of remaining energy with SEAD is a little broader than with DD. However, the graph implies that SEAD's energy savings have a good influence on the lifetime of the sensor network.

Figure 12 shows the success ratio in packet delivery as the sinks' speed changes. The success ratio always remains beyond 0.9, up to a speed of 30 m/sec. This indicates that SEAD disseminates most of the data successfully even to high-speed sinks. This is because a new path is made on the basis of the current path and because the sinks do not participate in the tree as intermediate members.

5.3 Impact of the Node Density

We evaluate the impact of node density on the SEAD protocol. It is an explicit goal that the protocol perform equally well regardless of the node density. The node density may vary according to the application and cost. In this experiment, the number of sensor nodes is varied from 300 to 500. Eight sinks move at 10 m/sec. Both the average sink refresh rate and the average source update rate are set to 6. Figure 13 shows the energy consumption per node at different sensor node densities. Figure 14 shows the end-to-end delay. The node density has little influence on the energy per node in SEAD although more neighbors overhear data from a sender at high density. This is because there are more chances that better energy cost paths can be found in a higher density network. SEAD shows the best performance in energy consumption for all density values.

As shown in Figure 14, DD has a shorter delay. TTDD has a slightly longer delay and more energy consumption due to the restriction that paths be aligned with a grid. ADMR exhibits poor performance in the mobile sink case compared to the stationary sink case because it uses sinks as intermediate members in the tree. In SEAD, the delay decreases as density grows due to greedy forwarding which then finds better routes to the destination.

5.4 End-to-end Delay of SEAD

In the following experiment, we investigate the average end-to-end delay as a function of the number of mobile sinks and their speed. The number of sensor nodes is 400. Both the average refresh rate and the average source update rate are set to 6. The first experiment is to measure the end-to-end delay for the different number of sinks when the sinks move at a speed of 10 m/sec. Figure 15 shows that the end-to-end delay increases as the number of sinks increases. This effect is the result of increasing the depth of the d-tree. The slope of the curve decreases with the number of sinks since there are more chances that a new sink can exploit an existing path.

The second experiment is to measure the end-to-end de-

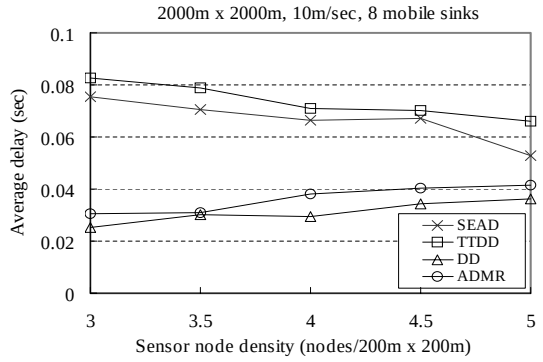


Figure 14: Average delay with different sensor node density.

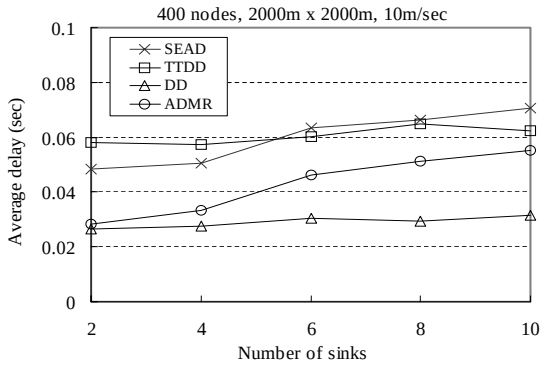


Figure 15: Average delay with different number of sinks.

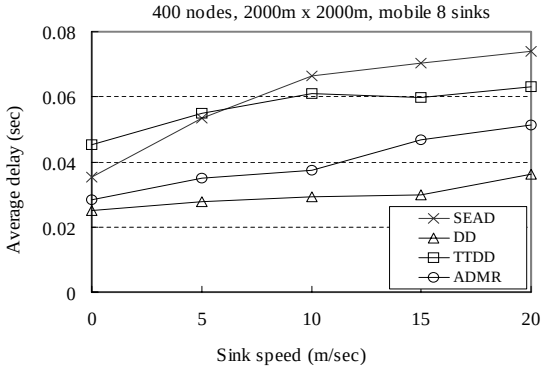


Figure 16: Average delay with different sink speeds.

lay at different sink speeds. The number of sinks is set to 8. Figure 16 shows that the end-to-end delay increases as the sink speed increases. The slope of the curve decreases with sink speed because a new lower delay path is built more often at higher speeds. The delay experiments show that SEAD performs competitively with existing protocols such as TTDD, but does not outperform all others as it does with energy consumption. SEAD is a protocol to minimize energy. Hence, its delay is longer than that of other approaches that focus more on delay minimization. SEAD is most useful for applications with less strict delay requirements, compared to their energy needs. We believe that this is the case with most sensor network applications, since a primary concern of sensor networks is to prolong lifetime. A few tens of milliseconds increase in average delay seems like an acceptable price to pay for such lifetime savings.

6. RELATED WORK

Wireless sensor networks have unique characteristics that generic wireless networks do not have. The number of sensor nodes may be orders of magnitude higher than the number of nodes in generic wireless networks. The node density is also much higher. Global identification-based addressing requires a large amount of overhead. Hence, stationary sensor node addresses are based on geographic location rather than a global identification number. Sensor nodes are unattended and thus limited resources cannot be replenished. For these reasons, traditional networking paradigms are not directly applicable to this scenario. There has been a lot of work in recent years to develop new paradigms and services for sensor networks, including several efforts on data dissemination taking into account the unique features of sensor networks.

The data placement heuristic in [2] presents an asynchronous multicast scheme for stationary sinks. It assumes that the number of stationary sensor nodes is ideally infinite so that an arbitrary location in the network always has a sensor node. It builds the tree by connecting specific locations obtained from a heuristic equation. If there is no node in the computed location, performance degradation occurs. SEAD finds actual sensor nodes for the optimal d-tree without deciding on a location beforehand. Thus SEAD performs well regardless of its node density as shown in Section 5. Another main difference is that the data placement heuristic of [2] does not deal with mobile sinks. It uses the stationary sinks as intermediate members of the tree and would cause too frequent reconfiguration of the tree if sinks were mobile.

Directed diffusion is a data-centric communication paradigm specifically designed for sensor networks [15]. Subscribers use flooding to spread interests to the sensor network. Sensors matching the interest send their data to the sinks along multiple paths initially, and then gradually reinforce better paths. Another dissemination protocol, SAFE [17] uses flooding that is geographically limited to forward the query to nodes along the direction of the source. SAFE uses geographically limited flooding to find the gate connecting itself to the tree. Directed diffusion and SAFE are most effective in small-to-medium size sensor networks. However, in a very large network, the initial sensor flooding may consume too much energy. The d-tree would be frequently reconstructed due to mobile sinks.

In order to reduce energy consumption, TTDD has been proposed. TTDD [25] exploits local flooding within a local cell of a grid which sources build proactively. Each source disseminates data along the nodes on the grid line to the sinks. It does not optimize the path from the source to the sinks. When a sensor communicates with a sink, the restriction of grid structure may multiply the length of a straight-line path by $\sqrt{2}$. This approach therefore incurs more energy and longer delays. In TTDD, the mobile sink renews its entire path to the dissemination point whenever it moves out of range in the local cell. Frequent renewal of the entire path to the sink may increase energy consumption and the connection loss ratio. Besides, TTDD's performance depends mainly upon its cell size. If the cell size is large, the local flooding increases energy consumption and the path between the source and the sink is longer. On the other hand, if the cell size is small, it costs much energy to build the grid and causes mobile sinks to frequently change cells.

SEAD differs from the aforementioned efforts in four fundamental ways. First, SEAD exploits a recursive algorithm that explicitly searches for the minimum energy dissemination path. Second, SEAD saves energy in managing mobile sinks. The sinks do not report their current locations to the tree. The entire path to the tree is not renewed when sinks move out of range. Third, SEAD takes into account end-to-end (source-to-sink) delay. It strikes an adjustable balance between minimum-energy dissemination and bounding the delay from the source to the mobile sinks. Another difference is that the performance of SEAD is not swayed by the node density of the sensor network. SEAD uses neither flooding nor limited flooding to construct the d-tree.

To conserve communication energy in sensor networks, some papers have discussed data aggregation of multiple sources or energy-aware routing between a sensor and a single sink. Data aggregation [22, 19] helps to reduce the amount of data transmitted from multiple sensor nodes to a sink. A leading sensor node combines one or more data packets from different sensor readings and sends the result to the sink. These in-network aggregation schemes can be used at the source in SEAD, and hence nicely complement our work. Bonfils [3] proposed a data gathering algorithm where a sink efficiently gathers data from many sources. This work places operators in the optimal positions for in-network query processing. It is different from ours in that we focus on dissemination from one source to many sinks, but it focused on gathering from many sources to a single sink. It does not take into account mobility.

In wireless ad hoc networks, the ShopParent algorithm [14] is the latest work of publish/subscribe tree construction. This greedy algorithm builds the tree in a distributed fashion and uses a spanning tree. Its model assumes that the tree includes every node in the network so that the nearest node becomes easier to search. However, like other ad hoc network studies, its model is not suitable for sensor networks with a large number of nodes in a vast field. ShopParent is different from SEAD in that it does not consider sink mobility. As a location-aware multicast algorithm in ad hoc networks, LGS [5] constructs the Steiner tree under the Takahashi-Matsuyama heuristics [24] using link costs as distances between nodes. When building an LGS tree, the connected sink selects the nearest sink, using location information received from every other sink. SEAD differs from LGS in that the location of sinks and junction nodes in the tree are not known a priori. SEAD searches for an optimal junction node among sensor nodes. Besides, SEAD does not connect a sink to another sink because the sink is mobile and tends to fail under hostile environments in sensor networks. Geocasting [18], another location-aware multicast, delivers data to a set of nodes within a specified geographical area. It is different from general multicast in that all receivers are geographically close.

SEAD is different from Internet content distribution networks [20, 23] in that SEAD is designed for wireless sensor networks with mobile sinks and makes use of geographic information to minimize energy that is the most crucial resource of the network. The d-tree is determined by both the update rate and the geographic distance to minimize the energy consumed for communication while meeting approximate end-to-end delay constraints.

7. CONCLUSIONS

In this paper, we described SEAD, a scalable energy-efficient asynchronous dissemination protocol for sending data to mobile sinks in sensor networks. It saves energy consumption in both building the d-tree and maintaining linkage to mobile sinks. SEAD strikes a balance between end-to-end delay and power consumption that favors power savings over delay minimization. The argument in favor of this approach is that energy consumption directly affects system lifetime, which has tangible economic implications. The algorithm exploits several energy-minimizing tactics. For example, it uses a stationary sensor node on behalf of the mobile sink as an end-point in the d-tree. Mobile nodes are never used to forward data to other nodes to minimize tree reconfiguration. When the sink joins the tree, the Steiner tree is built recursively so that the length of the additional branch and the data-delivering rate are minimized. Instead of flooding for joining the tree, SEAD finds the minimal cost entry to the tree for the sink using unicast. Simulations results have shown that SEAD consequently conserves the battery energy of the sensor node efficiently while delivering data without interruption to mobile sinks. The performance of SEAD is found insensitive to changes in node density. Future work of the authors will include implementing SEAD on a current sensor network prototype and exploring the design trade-offs that arise in a practical setting.

8. ACKNOWLEDGMENTS

This work is supported in part by Brain Korea 21 (BK-21), DARPA NEST grant F33615-01-C-1905, MURI grant N00014-01-1-0576, and NSF grant CCR-0208769.

9. REFERENCES

[1] J. Albowicz, A. Chen, and L. Zhang. Recursive position estimation in sensor networks. In *Proceedings of IEEE Internation Conference on Network Protocols(ICNP'01)*, pages 35–41, November 2001.

[2] S. Bhattacharya, H. Kim, S. Prabh, and T. Abdelzaher. Energy-conserving data placement and asynchronous multicast in wireless sensor networks. In *Proceedings on Mobile Systems, Applications, and Services (MobiSys)*, pages 173–186, May 2003.

[3] B. Bonfils and P. Bonnet. Adaptive and decentralized operator placement for in-network query processing.