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.

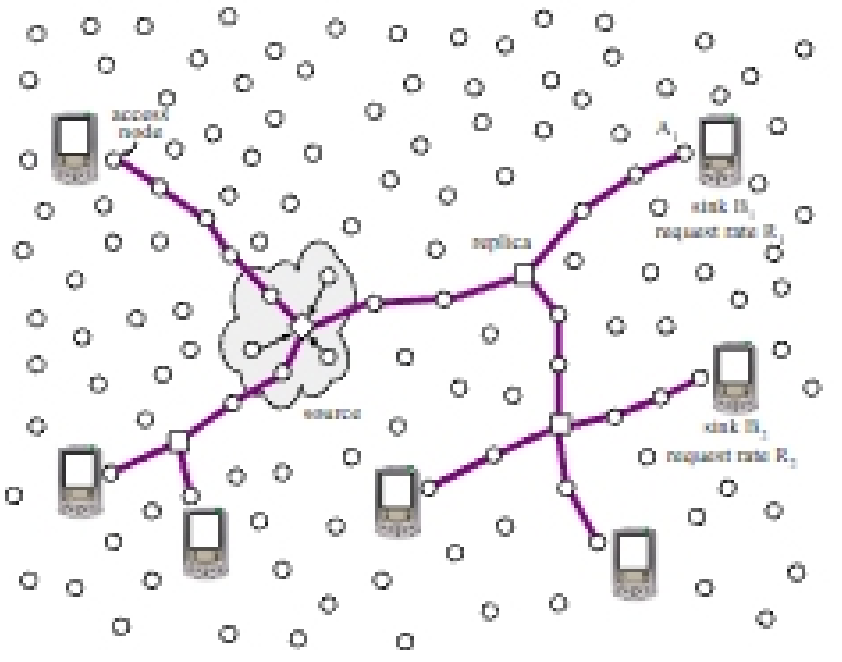

Figure 1: An example of the SEAD tree model in the sensor network.

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:

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

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 $K(r) - K(c)$.