Optimize Storage Placement in Sensor Networks

Bo Sheng, *Member*, *IEEE*, Qun Li, *Member*, *IEEE*, and Weizhen Mao

Abstract—Data storage has become an important issue in sensor networks as a large amount of collected data need to be archived for future information retrieval. Storage nodes are introduced in this paper to store the data collected from the sensors in their proximities. The storage nodes alleviate the heavy load of transmitting all data to a central place for archiving and reduce the communication cost induced by the network query. The objective of this paper is to address the storage node placement problem aiming to minimize the total energy cost for gathering data to the storage nodes and replying queries. We examine deterministic placement of storage nodes and present optimal algorithms based on dynamic programming. Further, we give stochastic analysis for random deployment and conduct simulation evaluation for both deterministic and random placements of storage nodes.

Index Terms—Wireless sensor networks, data storage, data query.

1 Introduction

Sensor networks deployed for pervasive computing applications, e.g., sensing environmental conditions and monitoring people's behaviors, generate a large amount of data continuously over a long period of time. This large volume of data has to be stored somewhere for future retrieval and data analysis. One of the biggest challenges in these applications is how to store and search the collected data.

The collected data can either be stored in the network sensors or transmitted to the sink. Several problems arise when data are stored in sensors. First, a sensor is equipped with only limited memory or storage space, which prohibits the storage of a large amount of data accumulated for months or years. Second, since sensors are battery operated, the stored data will be lost after the sensors are depleted of power. Third, searching for the data of interest in a widely scattered network field is a hard problem. The communication generated in a network-wide search will be prohibitive. Alternatively, data can be transmitted back to the sink and stored there for future retrieval. This scheme is ideal since data are stored in a central place for permanent access. However, the sensor network's per node communication capability (defined as the number of packets a sensor can transmit to the sink per time unit) is very limited [1], [2]. A large amount of data cannot be transmitted from the sensor network to the sink efficiently. Furthermore, the data communication from the sensors to the sink may take long routes consuming much energy and depleting of the sensor battery power quickly. In particular, the sensors around the sink are generally highly used and exhausted easily, thus, the network may be partitioned rapidly.

It is possible that, with marginal increase in cost, some special nodes with much larger permanent storage (e.g., flash memory) and more battery power can be deployed in sensor networks. These nodes back up the data for nearby sensors and reply to the queries. The data accumulated on each storage node can be transported periodically to a data warehouse by robots or traversing vehicles using physical mobility as Data Mule [3]. Since the storage nodes only collect data from the sensors in their proximity and the data are transmitted through physical transportation instead of hop by hop relay of other sensor nodes, the problem of limited storage, communication capacity, and battery power is ameliorated.

Placing storage nodes is related to the sensor network applications. We believe that query is the most important application for sensor networks since, in essence, sensor networks are about providing information of the environment to the end users. A user query may take various forms, e.g., "how many nodes detect vehicle traversing events?" and "what is the average temperature of the sensing field?" In this scenario, each sensor, in addition to sensing the nearby environment, is also involved in routing data for two network services: the raw data transmission to storage nodes and the transmission for query diffusion and query reply. Two extremes, as mentioned before, would be transmitting all the data to the sink or storing them on each sensor node locally. On one hand, data solely stored in the sink is beneficial to the query reply incurring no transmission cost, but the data accumulation to the sink is very costly. On the other hand, storing data locally incurs zero cost for data accumulation, whereas the query cost becomes large because a query has to be diffused to the whole network and each sensor has to respond to the query by transmitting data to the sink. The storage nodes not only provide permanent storage as described previously but also serve as a buffer between the sink and the sensor nodes. The positioning of storage nodes, however, is extremely important in this

- *B. Sheng is with the Department of Computer Science, University of Massachusetts Boston, 100 Morrissey Boulevard, Boston, MA 02125. E-mail: shengbo@cs.umb.edu.*
- *Q. Li and W. Mao are with the Department of Computer Science, College of William and Mary, McGlothlin-Street Hall, Williamsburg, VA 23187-8795. E-mail: {liqun, wm}@cs.wm.edu.*

Manuscript received 27 Jan. 2009; revised 15 Aug. 2009; accepted 12 Oct. 2009; published online 24 May 2010.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-2009-01-0033. Digital Object Identifier no. 10.1109/TMC.2010.98.

communication model. A bad placement strategy may waste the storage resources and have an adverse effect on the performance. Therefore, a good algorithm for placing storage nodes is needed to strike a balance between these two extremes characterizing a trade-off between data accumulation and data query.

This paper considers the storage node placement problem in a sensor network. This problem can be very complex with respect to various optimization metrics, such as bandwidth minimization on each link and lifetime of the network. However, the bandwidth problem is alleviated because the storage nodes collect the data generated within their proximities and avoid the heavy load on each link incurred by the long-distance data communication. Lifetime of a network depends on the routing scheme and the network structure. Specifically, it is mainly affected by the most loaded nodes, which are likely the nodes close to the sink. The storage placement does not reduce the loads on those nodes since all the responses have to be sent through the nodes around the sink. Therefore, even though lifetime is an important metric, it is not a good metric to guide the storage placement design. Instead, we consider an approximation of lifetime: energy cost. We believe that energy cost is more crucial as it is the most important performance metric in sensor network design. Therefore, we aim to minimize the total energy cost in data accumulation and data query by judiciously placing the storage nodes.

We first examine the problem in the fixed tree model. We assume that the sensor network has organized into a tree rooted at the sink. The communication routes from sensors toward the sink are predefined by the tree. Our goal is to select some of the nodes in the tree as storage nodes, each of which is responsible for storing the raw data of its descendants that are not covered by other storage nodes. In many applications, for example, a sensor network along the highway, or a drainage or oil pipeline monitoring system, the network communication tree is fixed due to the constraints of the sensor deployment. Our results for the fixed tree model fit into those scenarios well. We also consider the dynamic tree model, where the (optimal) communication tree is constructed after the storage nodes are deployed. Specifically, each sensor selects a storage node in its proximity for its data storage with the goal to minimize the energy cost of the resulting communication tree. The sketch of our solution and preliminary results have been published in [4].

The remainder of the paper is organized as follows: In Section 2, we review the related work. In Section 3, we define several problems for storage node placement in the aforementioned models. In Sections 4 and 5, we present optimal algorithms for the problems defined within the fixed tree model. In Section 6, we give theoretical analysis of randomized deployment of storage nodes for both the fixed tree and dynamic tree models. In Section 7, we present our simulation results, which confirm the theoretical analysis. Finally, in Section 8, we make the conclusion and discuss future research.

2 Related Work

There have been a lot of prior research works on data querying models in sensor networks. In early models [5], [6], query is spread to every sensor by flooding messages. Sensors return data back to the sink in the reverse direction of query messages. Those methods, however, do not consider the in-network storage.

In the literature, several schemes have introduced an intermediate tier between the sink and sensors. LEACH [7] is a clustering-based routing protocol, where cluster heads can fuse the data collected from its neighbors to reduce communication cost to the sink. However, LEACH does not address storage problem. Data-centric storage schemes [8], [9], [10], [11], [12], as another category of the related work, store data on different places according to different data types. In [8], [9], and [11], the authors propose a data-centric storage scheme based on Geographic Hash Table, where the home site of data is obtained by applying a hash function on the data type. Another practical improvement is proposed in [12] by removing the requirement of point-to-point routing. Ahn and Krishnamachari [13] analyze the scaling behavior of data-centric query for both unstructured and structured (e.g., GHT) networks and derive some key scaling conditions. GEM [10] is another approach that supports data-centric storage and applies graph embedding technique to map data to sensor nodes. In general, the data-centric storage schemes assume some understanding about the collected data and extra cost is needed to forward data to the corresponding keeper nodes. In our paper, we do not assume any prior knowledge about the data: indeed in many applications, raw data may not be easily categorized into different types. To transmit the collected data to a remote location is also considered expensive because the total collected data may be in a very large quantity.

To facilitate data query, Ganesan et al. [14] propose a multiresolution data storage system, DIMENSIONS, where data are stored in a degrading lossy model, i.e., fresh data are stored completely while long-term data are stored lossily. In comparison, our scheme is more general without any assumption about the data correlation. PRESTO [15] is a recent research work on storage architecture for sensor networks. A proxy tier is introduced between sensor nodes and user terminals and proxy nodes can cache previous query responses. Compared to the storage nodes in this paper, proxy nodes in PRESTO have no resource constraints in terms of power, computation, storage, and communication. It is a more general storage architecture that does not take the characteristics of data generation or query into consideration. In the Cougar project [16], [17], a data dissemination tree is built with data sources as leaves. *View nodes* introduced in Cougar have similar functionalities as storage nodes in this paper. Our scheme focuses more on how to optimize the placement of storage nodes, while Cougar mainly focuses on how to implement data query with more details in a sensor network.

In addition, recent research work has shown the feasibility of storage nodes. Mathur et al. [18] investigate hardware supports to attach large-capacity flash memory to sensors. Their measurement study shows that access to large local storage is practical for sensors. Furthermore, storage nodes are also supported by recent research on software system. Zeinalipour-Yazti et al. propose MicroHash [19], an indexing structure to manage external flash memory of sensors in

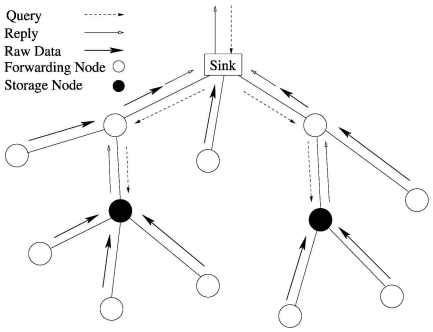

Fig. 1. Data access model with storage nodes.

order to efficiently look up the stored data. Mathur et al. [20] design Capsule system as an intermediate layer between flash memory and sensor applications.

In [21] and [22], the authors introduce an approach to analyzing communication networks based on stochastic geometry. They consider models built on Poisson processes and obtain formulas to express the average cost in function of the intensity parameters of Poisson processes. Baek et al. extend this work specifically to sensor networks in [23]. Our paper also analyzes random placement of storage nodes in Section 6. We will use similar means with [21], [22], and [23] to derive analytical formulas for the performance.

3 Problem Formulation

In this paper, we consider an application in which sensor networks provide real-time data services to users. A sensor network is given with one special sensor identified as the sink (or base station) and many normal sensors, each of which generates (or collects) data from its environment. Users specify the data they need by submitting queries to the sink and they are usually interested in the latest readings generated by the sensors.[1] To reply to queries, one typical solution is to let the sink have all the data. Then, any query can be satisfied directly by the sink. This requires each sensor to send its readings back to the sink immediately every time it generates new data. Generally, transferring all raw data could be very costly and is not always necessary. Alternatively, we allow sensors to hold their data and to be aware of the queries, then raw data can be processed to contain only the readings that users are interested in and the reduced-size reply, instead of the whole raw readings, can be transferred back to the sink. This scheme is illustrated in Fig. 1, where the black nodes, called *storage nodes*, are allowed to hold data. The sink diffuses queries to the storage nodes by broadcasting to the sensor network and these storage sensors reply to the queries by sending the processed data back. Compared to the previous solution, this approach reduces the raw data transfer cost (as indicated by the thick arrows in the figures) because some raw data transmissions are replaced by query reply (as indicated by the thin arrows). On the other hand, this scheme incurs an extra query diffusion cost (as indicated by the dashed arrows). In this paper, we are interested in strategically designing a data access model to minimize energy cost associated with raw data transfers, query diffusion, and query replies.

We first formally define two types of sensors (or nodes). ***1) Storage nodes***: This type of nodes have much larger storage capacity than regular sensors. In the data access model, they store all the data received from other nodes or generated by themselves. They do not send out anything until queries arrive. Based on the query description, they obtain the results needed from the raw data they are holding and then return the results back to the sink. Note that except enriched storage capacity, other resources on storage nodes are still constrained as regular sensors. The sink itself is considered as a storage node. ***2) Forwarding nodes***: This type of nodes are regular sensors and they always forward the data received from other nodes or generated by themselves along a path toward the sink. The outgoing data are kept intact and the forwarding operation continues until the data reach a storage node. The forwarding operation is independent of queries and there is no data processing at forwarding nodes.

Therefore, our goal is to design a centralized algorithm that can derive the best locations of the storage nodes to guide the deployment of such a hybrid sensor network. We make the following assumptions about the characteristics of data generation, query diffusion, and query replies: First, for data generation, we assume that each node generates r_d readings per time unit and the data size of each reading is s_d. Second, for query diffusion, we assume that r_q queries of the same type are submitted from users per time unit and the size of the query messages is s_q. Third, for query reply, we assume that the size of data needed to reply a query is a fraction α of that of the raw data. Specifically, we define a data reduction function f for query reply. For input x, which is the size of raw data generated by a set of nodes, function $f(x) = \alpha x$ for $\alpha \in (0, 1]$ returns the size of the processed data, which is needed to reply the query. We do not restrict the types of queries we impose on the sensor network in this paper, but we assume that we can obtain an empirical value for α through examining the historic queries. The parameter α characterizes many queries satisfied by a certain fraction of all the sensing data, e.g., a range query may be "return all the nodes that sense a temperature higher than 100 degree" and α can be estimated based on the data distribution information.

In this paper, the communication among all n nodes is based on a tree topology rooted at the sink. The tree is formed in the initial phase as follows: The sink first broadcasts a message with a hop counter. The nodes receiving the message will set the message sender as the parent node, increase the hop counter by 1, and broadcast it. If a node receives multiple messages, it will select the one with the minimum hop counter to broadcast and set the sender of the message as its parent. Data are transferred along the edges in this communication tree. To transmit one data unit, the energy cost of the sender and receiver is e_{tr} and e_{re}, respectively, and e_{tr} is also relevant to the distance between the sender and receiver. To simplify the problem, we set the length of each tree edge to one unit, which means that sensor nodes have a fixed transmission range and the energy cost of transferring data is only proportional to the data size. Our algorithms can be easily extended to nonuniform transmission ranges as long as

1. Our algorithms also apply to the queries to the historic data. For the ease of presentation, we assume that all queries correspond to the latest generated data.

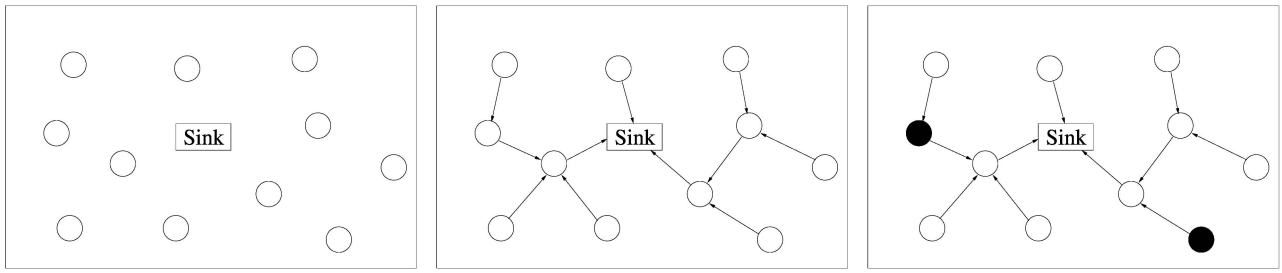

Fig. 2. Deploy storage nodes in the fixed tree model (storage nodes are black). (a) Step 1: Regular sensors are deployed in a field. (b) Step 2: A communication tree is constructed to relay data. (c) Step 3: Some regular sensors are upgraded to or replaced by storage nodes.

topology information is available. In our energy model, for simplicity of presentation, the receiving energy cost is assigned to the sender without changing the total energy cost. When sensor i sends one data unit to j, the energy cost of j is 0, and the energy consumed by i is

$$\begin{cases} e_{tr} + e_{re} & \text{if } j \text{ is } i\text{'s parent;} \\ e_{tr} + e_{re} \cdot c_i & \text{if } j \text{ is one of } i\text{'s children,} \end{cases}$$

where c_i is the number of i's children. In the rest of this paper, we normalize the energy cost by $(e_{tr} + e_{re})$ for easy presentation. Thus, transferring one data unit from i to its parent consumes one energy unit, and to broadcast one data unit to its children, i will consume b_i energy units, where $b_i = \frac{e_{tr}+e_{re}\cdot c_i}{e_{tr}+e_{re}}$.

Tree structure has been widely used in sensor networks. Although the communication tree may be broken due to link failure, this paper considers a common practice that only stable links are chosen to build the tree so that errors in transmission due to poor link quality can be reduced and the tree topology can be robust for a long time. Since the tree topology changes will be rare incurring small communication costs compared to the query and data collection cost, we will not consider the cost for tree topology information update in this paper. If packet loss happens, sensors may retransmit the packet till it gets through. We assume that the probability of retransmission is the same for all links. Thus, the energy cost for communication on each link is still proportional to the data size even though the overhead for a unit packet is larger than that of a perfect wireless link.

The tree structure is independent of the underlying low-layer communication protocol: like myriad routing protocols, tree structure is one of the routing schemes. Our algorithm only assumes that the energy cost is proportional to the transmitted data size, which can be realized in many communication protocols such as the duty cycle mechanism. Moreover, we assume that the applications considered in this paper can tolerate the delay caused by the low-layer communication, such as retransmission and duty cycle mechanism. We would also like to mention that the lower layer implementation, such as the duty cycle mechanism, is compatible with the functionality of the tree structure-based communication. Tree structure is constructed in the initialization phase in which duty cycle has not been started. Therefore, duty cycle does not affect the tree construction while the tree structure has to be considered for selecting the parameters for duty cycle.

Here, we present basic analysis of energy cost. Let i be a node in the communication tree and T_i be the subtree rooted at i. We use $|T_i|$ to denote the number of nodes in T_i. We define $e(i)$ to be the energy cost incurred at i per time unit, which includes the cost for raw data transfer from i to its parent if i is a forwarding node, the cost for query diffusion if i has storage nodes as its descendants, and the cost for query reply if i is a storage node or has a storage descendant. To define $e(i)$ mathematically, we need to consider several possible cases.

Case A. i is a forwarding node and there are no storage nodes in T_i. All raw data generated by the nodes in T_i have to be forwarded to the parent of i and there is no query diffusion cost. So $e(i) = |T_i| r_d s_d$.

Case B. i is a storage node and there are no other storage nodes in T_i. The latest readings of all raw data generated by the nodes in T_i are processed at node i and the reply size is be $\alpha|T_i|s_d$. Node i sends the reply to its parent when queries arrive. So $e(i) = r_q \alpha |T_i| s_d$.

Case C. i is a storage node and there is at least one other storage node in T_i. In addition to the cost for query reply as defined in Case B, i also incurs a cost for query diffusion that is implemented by broadcasting to its children. So $e(i) = r_q \alpha |T_i| s_d + b_i r_q s_q$.

Case D. i is a forwarding node and there is at least one storage node in T_i. This is the case where all three types of cost (for raw data transfer, query diffusion, and query reply) are present. Among the $|T_i| - 1$ descendants of i, let d_1 be the number of forwarding descendants without any storage nodes on their paths to i (the raw data generated at these d_1 nodes and at i itself will be forwarded from i to its parent without reduction) and d_2 be the number of storage descendant's or forwarding descendants with at least one storage node on their paths to i (the last readings of the raw data generated at these d_2 nodes would have been processed and reduced before reaching i). Obviously, $d_1 + d_2 = |T_i| - 1$. So,

$$e(i) = (d_1 + 1) r_d s_d + b_i r_q s_q + r_q \alpha d_2 s_d.$$

The communication tree can be formed before or after storage node deployment. Accordingly, we will consider two models in the rest of this paper. In the fixed tree model, as illustrated in Fig. 2, we first deploy regular sensors and construct a communication tree as usual. Based on the topology information, we select some of the regular sensors to be storage nodes. We can attach large flash memory to

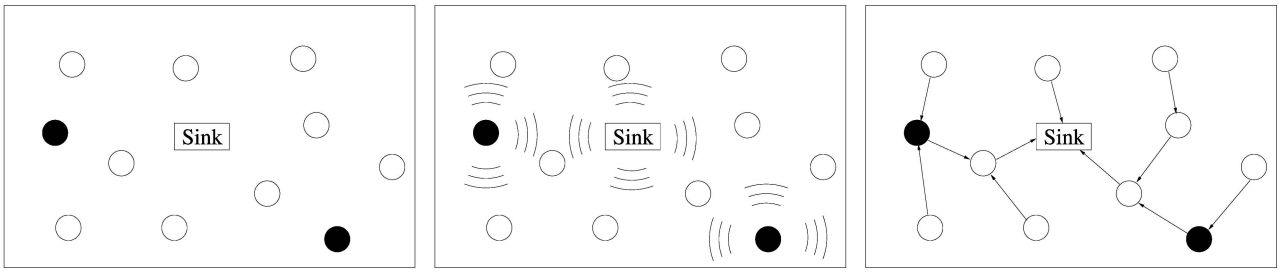

Fig. 3. Deploy storage nodes in the dynamic tree model (storage nodes are black). (a) Step 1: Regular sensors and storage nodes are deployed in a field. (b) Step 2: Each storage node (including the sink) broadcasts its location information. (c) Step 3: A communication tree is constructed based on the locations of storage nodes.

these selected sensors or replace them by more powerful storage nodes at the same locations. The other model, the dynamic tree model, is illustrated in Fig. 3. In this model, storage nodes are deployed before the communication tree is formed and their location information is broadcast to nearby regular sensors. After that, all sensors organize themselves into a communication tree according to the locations of the storage nodes. In both models, after tree construction and storage node deployment, each storage node needs to send a notification toward the sink. In this way, every sensor is aware of the existence of storage nodes among its descendants, and when a query arrives, it is able to determine whether to continue the diffusion or not.

Within the fixed tree model, we will consider two problems of storage node placement. Given an undirected tree T with nodes labeled as $1, 2, \ldots, n$, the length of each edge is 1. Let $e(i)$ be the energy cost of node i in one time unit as defined above. The objective is to place storage sensors (and hence, forwarding sensors) on nodes in T such that the total energy cost $\sum_{i \in T} e(i)$ is minimized. In the case when there is no limit on the number of storage nodes that can be used to minimize the energy cost, the problem is denoted by UNLIMITED. In the case when there are a limited number of storage nodes, say, k, to use, the problem is denoted by LIMITED. For the dynamic tree model, the storage node placement problem becomes how to place k given storage nodes to form a communication tree with the minimum total energy cost. This problem is NP-hard since it is a general case of the minimum k-median problem. We have proposed a 10-approximation algorithm for the dynamic tree model in our previous work [24].

The above problems defined with the fixed tree and dynamic tree models aim to find the optimal locations for storage nodes in a deterministic way. In reality, however, the storage nodes may not be deployed in a precise way. Instead, their deployment may be random with a certain density λ. This paper also evaluates the performance of random deployment of storage nodes in fixed and dynamic trees. Finally, Table 1 lists the notations, which will be used in the rest of the paper.

4 Unlimited Number of Storage Nodes

We first present a linear-time algorithm for the problem UNLIMITED, where an unlimited number of storage nodes are available to use to minimize the energy cost of a communication tree. Recall that $e(i)$ is the energy cost at node i. Let T_i be the subtree rooted at i. Then, $E(i)$ is the energy cost of nodes in T_i, defined to be $E(i) = \sum_{i \in T_i} e(i)$.

Our algorithm relies on the following lemma:

Lemma 1. *Given a node i and its subtree T_i, if $\alpha r_q \geq r_d$, then i must be a forwarding node to minimize $E(i)$. Otherwise, i must be a storage node to minimize $E(i)$.*

Proof. We compare the energy cost of two trees, which are identical in every aspect except that the first tree's root is a forwarding node and the second tree's root is a storage node. Let E_1 and E_2 be the energy cost of these two trees, respectively. Comparing the energy cost of individual nodes, one by one, in the two trees, we observe that any two nonroot nodes in the same position of the trees must have the same energy cost. The only difference is the energy cost of the roots. Let e_1 and e_2 be the energy cost of the roots in the two trees, respectively. Therefore, $E_1 - E_2 = e_1 - e_2$. To prove the lemma, it suffices to prove that

$$e_1 - e_2 \begin{cases} \leq 0 & \text{if } \alpha r_q \geq r_d; \\ > 0 & \text{if } \alpha r_q < r_d. \end{cases}$$

We consider two cases. First, if both roots have no storage descendants, then according to the four-case definition of energy cost given in the previous section (Cases A and B, specifically), we have

$$\begin{aligned} e_1 - e_2 &= |T_i| r_d s_d - r_q \alpha |T_i| s_d \\ &= |T_i| s_d (r_d - \alpha r_q) \begin{cases} \leq 0 & \text{if } \alpha r_q \geq r_d; \\ > 0 & \text{if } \alpha r_q < r_d. \end{cases} \end{aligned}$$

TABLE 1
Summary of Notations

r_d / s_d: rate of data generation / size of each data
r_q / s_q: rate of user queries / size of each query message
α: data reduction rate (query reply size / raw data size)
n: total number of sensors
k: total number of storage nodes (for LIMITED problem)
T_i: the subtree rooted at node i
d_i: the depth of node i
c_i: the number of node i's children
$e(i)$: energy cost of node i
$E(i)$: energy cost of all the nodes in T_i
λ / λ_s: density of regular sensors / storage nodes
e_{tr} / e_{re}: energy cost for transmitting / receiving a unit data

Second, if both roots have at least one storage descendant, then according to the four-case definition of energy cost given in the previous section (Cases D and C, specifically), we have

$$\begin{aligned} e_1 - e_2 &= ((d_1+1)r_d s_d + b_i r_q s_q + r_q \alpha d_2 s_d) \\ &\quad - (r_q \alpha |T_i| s_d + b_i r_q s_q) \\ &= (d_1+1) s_d (r_d - \alpha r_q) \begin{cases} \le 0 & \text{if } \alpha r_q \ge r_d; \\ > 0 & \text{if } \alpha r_q < r_d. \end{cases} \end{aligned}$$

In the first tree with a forwarding root, recall that d_1 is the number of forwarding descendants of the root without any storage nodes on their paths to the root and that d_2 is the number of storage descendants plus the number of forwarding descendants with at least one storage node on their paths to the root. Also recall that $d_1 + d_2 = |T_i| - 1$. □

From the above lemma, we can conclude that if $\alpha r_q \ge r_d$, then every node (except for the root/sink, which is always a storage node) in the sensor network must be a forwarding node to minimize the energy cost. However, if $\alpha r_q < r_d$, things are a little tricky. Although the root of the tree, say, i, must be a storage node, it may not be true that every node in the sensor network must be a storage node to minimize the energy cost. One would think that in order for the tree to incur a minimum energy cost, all of its subtrees should incur a minimum energy cost. However, since $\alpha r_q < r_d$, these optimal subtrees all have storage nodes as their roots. This means that the energy cost of root i will have to include the cost for query diffusion $b_i r_q s_q$ since it has storage children, i.e., $e(i) = r_q \alpha |T_i| s_d + b_i r_q s_q$. The cost for query diffusion, however, can be eliminated if all subtrees of i have only forwarding nodes, i.e., $e(i) = r_q \alpha |T_i| s_d$. (See Cases C and B in the four-case definition of $e(i)$ in the previous section.) Thus, the minimum energy cost of the tree rooted at i should be derived from the better of these two scenarios.

For a tree T_i rooted at i, let C_i be the set of children of i. Let $E^*(i)$ be the minimum (optimal) energy cost of T_i. If C_i is empty, i.e., i is a leaf, then i must be a storage node to achieve its minimum energy cost. So $E^*(i) = r_q \alpha s_d$. If C_i is not empty, then for any $j \in C_i$, let $E_f(j)$ be the energy cost of T_j when all nodes in T_j are forwarding nodes. So

$$E^*(i) = \min \Bigg\{ r_q \alpha |T_i| s_d + b_i r_q s_q + \sum_{j \in C_i} E^*(j) \, , \; r_q \alpha |T_i| s_d + \sum_{j \in C_i} E_f(j) \Bigg\}.$$

Algorithm 1 given in pseudocode finds the optimal placement of storage nodes in two cases: 1) $\alpha r_q \ge r_d$ and 2) $\alpha r_q < r_d$, where the first case is trivial and the second case is solved by dynamic programming that works from the bottom to the top of the tree. Assume that n nodes in the tree T are labeled using the postorder.[2] A table $E^*[1..n]$ is used to hold the minimum energy cost of all subtrees rooted at node $i = 1, \ldots, n$. At the end of the computation, $E^*[n]$ will hold the minimum energy cost of T (which is rooted at n according to the postorder labeling). We also maintain a second table $E_f[1..n]$, which records the energy cost of all subtrees when all nodes in each subtree are forwarding nodes. In the algorithm, lines 5-9 compute the E^* and E_f entries for all leaves and lines 10-19 compute the E^* and E_f entries for the remaining nodes following our postorder.

Algorithm 1. Place Unlimited Storage Nodes

```
1:  make the root a storage node
2:  if αr_q ≥ r_d then
3:    make all nonroot nodes forwarding nodes and return
4:  end if
5:  for all leaves i do
6:      make i a storage node
7:      E*[i] = r_q α s_d
8:      E_f[i] = r_d s_d
9:    end for
10:   for all remaining nodes i, in postorder, do
11:     make i a storage node
12:     cost1 = r_q α|T_i| s_d + b_i r_q s_q + Σ_{j∈C_i} E*[j]
13:     cost2 = r_q α|T_i| s_d + Σ_{j∈C_i} E_f[j]
14:     E*[i] = min{cost1, cost2}
15:     E_f[i] = |T_i| r_d s_d + Σ_{j∈C_i} E_f[j]
16:     if cost1 ≥ cost2 then
17:         change each descendant of i that is a storage
            node to a forwarding node
18:     end if
19:   end for
```

There are only $O(n)$ entries to compute in tables E^* and E_f, and to compute each entry that corresponds to a node, only its children will have to be considered. Furthermore, each node starts as a storage node. Once it is changed to a forwarding node by the algorithm, it will never be changed back. Therefore, the time complexity of Algorithm 1 is $O(n)$, where n is the number of nodes.

Summarizing the discussion above, we have the following theorem:

Theorem 1. *If $\alpha r_q \ge r_d$, then the optimal tree with the minimum energy cost contains only forwarding nodes (except for the root). If $\alpha r_q < r_d$, then the optimal tree can be constructed by a dynamic programming algorithm in $O(n)$ time.*

We also observe that every node starts as a storage node and that once it is changed to a forwarding node, all its descendants are changed to forwarding nodes as well. Thus, it is impossible for a forwarding node to have a storage descendant. Likewise, it is impossible for a storage node to have a forwarding ancestor. We then have the following corollary:

Corollary 1. *In the optimal tree, if i is a forwarding node, all its descendants are forwarding nodes. If i is a storage node, all its ancestors are storage nodes.*

In summary, this UNLIMITED problem refers to the scenario that the deployment budget is sufficient to upgrade every sensor to be a storage node. However, simply making all sensors storage nodes may not be the best strategy. The appropriate deployment still depends on

2. The postorder used in this paper is slightly different from the textbook definition of postorder in that our postorder requires all leaves to be listed first.

the characteristics of query and data generation. Intuitively, if there are a large volume of queries for a certain set of data and the reduction function yields a large α, it would be better to transfer these data to the sink. On the other hand, if queries are infrequent and the reply size is much less than the raw data, it would be more efficient to hold the raw data locally. According to Theorem 1 and Corollary 1, the optimal deployment of storage nodes has a special property. For any path from a leave to the root, there is a clear boundary that distinguishes forwarding nodes from storage nodes. The nodes below the boundary layer to leaves are forwarding nodes and the nodes above the boundary toward the sink are storage nodes.

5 Limited Number of Storage Nodes

In the problem UNLIMITED discussed in the previous section, we assume that we have enough storage nodes for the need to minimize the energy cost of the network. In reality, however, storage nodes may come with a hardware cost. Considering a limited budget for deploying a sensor network, there might be only a small portion of sensors as storage nodes. This is why we have also defined the problem LIMITED, which is similar to UNLIMITED except that we have only k storage nodes to deploy. Since the root (sink) is always a storage node, we assume that $k \geq 1$ and that $k-1$ is the maximum number of storage nodes that may appear as descendants of the root. Furthermore, from the discussion in the previous section, if $\alpha r_q \geq r_d$, the optimal tree has no storage nodes at all except the root. In this case, we just do not deploy any of the $k-1$ storage nodes and we get an optimal tree. Our discussion in this section on LIMITED is for the case of $\alpha r_q < r_d$. Since the number of storage nodes is limited, where to place them becomes a crucial problem. A bad placement strategy may hardly improve the performance. Basically, there is a trade-off between two trends. On the one hand, if storage nodes are close to the sink, i.e., at a high level in the tree structure, they can process more raw data, thus reduce the reply size from storage nodes to the sink. However, the sensor network spends much energy in transferring the raw data from low-level forwarding nodes to the storage nodes. On the other hand, if the storage nodes are far away from the sink, the raw data from their descendants can be processed earlier along the path toward the sink. However, storage nodes may cover only a few regular sensors, which leads to much raw data transferred to the sink without being processed. Besides this trade-off, the benefits from a storage node also depend on the locations of other storage nodes. Therefore, in this section, we propose the optimal placement strategy in order to maximize the benefits from deploying k storage nodes.

5.1 Arbitrary Trees

Assume that a communication tree T is given with up to k storage nodes already optimally deployed. By definition, the energy cost of T is $\sum_{i \in T} e(i)$. However, we are going to use a different and unique method to calculate this cost, which works from the bottom of the tree toward the root. Starting from the leaf nodes and following the postorder until the root is eventually reached, for each node i, we compute the energy cost already incurred within the subtree T_i rooted at i, which is $E(i)$ by our notation, plus the energy cost contributed by the nodes in T_i to their ancestors, which includes both raw data transmission cost and query reply cost according to the four-case definition of the energy cost of an individual node. Specifically, if i is a forwarding node, it contributes a raw data transmission cost of $r_d s_d$ to each of its forwarding ancestors that lie between i and i's closest storage ancestor (due to Cases A and D) and a query reply cost of $r_q \alpha s_d$ to each of the other ancestors (due to Cases B and D). If i is a storage node, however, it contributes a query reply cost of $r_q \alpha s_d$ to each of its ancestors (due to Cases C and D). Fig. 4 depicts the two scenarios. The top path from node i to the root (sink) is when i is a forwarding node and the bottom path from i to the root is when i is a storage node. Above each node (except i) is the contribution from i to the energy cost incurred at the node.

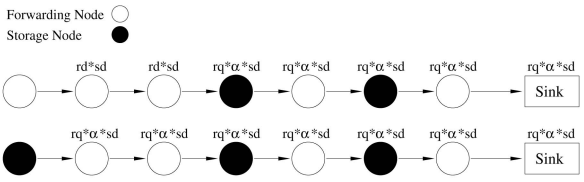

Fig. 4. Computing the contribution to the energy cost of all ancestors.

Let l be the number of forwarding nodes between i and its closest storage ancestor, not including i. Let m be the upper bound on the number of storage nodes in T_i. Then, we use $E_i(m,l)$ for the energy cost that includes $E(i)$ and the amount contributed by the nodes in T_i to the energy cost of their ancestors. Note that $0 \leq m \leq k$ and $0 \leq l \leq n-2$. In the case that i is a storage node or i's parent is a storage node, l becomes 0. Furthermore, if $m = 0$, no storage node is used in T_i, and if $m \geq 1$, at least one but no more than m storage nodes are used in T_i. Therefore, $E_n(k,0)$ is the minimum energy cost of T with up to k storage nodes to deploy, assuming that n is the label for the root.

When traversing the nodes in postorder in the tree starting from the leaves, let i be the current node being traversed. Let d_i be the depth of i in the tree, which is the number of edges on the path from i to the root n. We can define $E_i(m,l)$ recursively. For notational simplicity, we first define $Q_0(m)$ and $Q_1(m)$ as follows:

$$Q_0^i(m) = \begin{cases} 0 & \text{if } m = 0; \\ b_i r_q s_q & \text{if } m \geq 1. \end{cases}, \quad Q_1^i(m) = Q_0^i(m-1).$$

If i is a leaf node, $E_i(m,l)$ includes the energy cost of i and the precalculated amount contributed by i to all of its d_i ancestors. Specifically, if i is a forwarding node, its own energy cost is $r_d s_d$ and its contribution to the energy cost of its ancestors is $l r_d s_d + (d_i - l) r_q \alpha s_d$. If i is a storage node, its own energy cost is $r_q \alpha s_d$ and its contribution to the energy cost of its ancestors is $d_i r_q \alpha s_d$. Therefore,

$$E_i(m,l) = \begin{cases} (l+1) r_d s_d + (d_i - l) r_q \alpha s_d & \text{if } m = 0; \\ (d_i + 1) r_q \alpha s_d & \text{if } m \geq 1. \end{cases}$$

If i is a forwarding nonleaf node with a child set of C_i, the upper bound m must be divided among all of

its children. Let $P(m)$ be the set of all permutations $p = (m_j^p | j \in C_i)$, where $\sum_{j \in C_i} m_j^p = m$ and m_j^p denotes the upper bound on the number of storage nodes for subtree T_j in permutation p. Then, $E_i(m, l)$ is defined to be the sum of the amount from all of its subtrees,

$$\min_{\forall p \in P(m)} \left\{ \sum_{j \in C_i} E_j(m_j^p, l+1) \right\},$$

the energy cost of i, $r_d s_d + Q_0^i(m)$, and the precalculated amount of energy cost contributed by i to its ancestors, $l r_d s_d + (d_i - l) r_q \alpha s_d$. So,

$$E_i(m, l) = \min_{\forall p \in P(m)} \left\{ \sum_{j \in C_i} E_j(m_j^p, l+1) \right\} + (l+1) r_d s_d + (d_i - l) r_q \alpha s_d + Q_0^i(m).$$

If i is a storage nonleaf and nonroot node, the upper bound $m - 1$ must be divided among all of its children. Let $P(m-1)$ be the set of all permutations $p = (m_j^p | j \in C_i)$, where $\sum_{j \in C_i} m_j^p = m - 1$ and m_j^p denotes the upper bound on the number of storage nodes for subtree T_j in permutation p. Then, $E_i(m, l)$ is defined to be the sum of the amount from all of its subtrees,

$$\min_{\forall p \in P(m-1)} \left\{ \sum_{j \in C_i} E_j(m_j^p, 0 \right\},$$

the energy cost of i, $r_q \alpha s_d + Q_1^i(m)$, and the precalculated amount of energy cost contributed by i to its ancestors, $d_i r_q \alpha s_d$. So,

$$E_i(m, l) = \min_{\forall p \in P(m-1)} \left\{ \sum_{j \in C_i} E_j(m_j^p, 0) \right\} + (d_i + 1) r_q \alpha s_d + Q_1^i(m).$$

Algorithm 2 given here maintains a two-dimensional $(k+1) \times (n-1)$ table, $E_i[m, l]$, at each node i, where $0 \le m \le k$ and $0 \le l \le n-2$. Assume that a postorder traversal is done beforehand and that the depth of each node is computed beforehand. Both the postorder and the depths can be obtained in $O(n)$ time. In the algorithm, lines 1-6 compute the E_i tables for all leaves i, lines 7-12 compute the E_i tables for the remaining nonroot nodes i, and lines 13 and 14 compute the entry $E_n[k, 0]$ for the root n. After all the tables are constructed, the minimum energy cost of the tree with up to k storage nodes can be found in the entry $E_n[k, 0]$. Note that instead of constructing a table for the storage root n, we compute only the needed entry for n.

Algorithm 2. Place Limited Storage Nodes

1: **for** all leaves i **do**
2: **for** $m = 0$ to k **do**
3: **for** $l = 0$ to $n - 2$ **do**
4: **if** $m = 0$ **then**
5: $E_i[m, l] = (l+1) r_d s_d + (d_i - l) r_q \alpha s_d$
6: **if** $m \ge 1$ **then** $E_i[m, l] = (d_i + 1) r_q \alpha s_d$
7: **for** all remaining nonroot nodes i, in postorder, **do**
8: **for** $m = 0$ to k **do**
9: **for** $l = 0$ to $n - 2$ **do**
10: $min1 = \min_{\forall p \in P(m)} \{\sum_{j \in C_i} E_j[m_j^p, l+1]\} + (l+1) r_d s_d + (d_i - l) r_q \alpha s_d + Q_0^i(m)$
11: $min2 = \min_{\forall p \in P(m-1)} \{\sum_{j \in C_i} E_j[m_j^p, 0]\} + (d_i + 1) r_q \alpha s_d + Q_1^i(m)$
12: $E_j[m, l] = \min\{min1, min2\}$
13: $E_n[k, 0] = \min_{\forall p \in P(k-1)} \{\sum_{j \in C_n} E_j[m_j^p, 0]\} + r_q \alpha s_d + Q_1^i(m)$
14: return $E_n[k, 0]$

Assume that every node in the tree has at most c children. To partition an upper bound m into up to c upper bounds with the sum equal to m, there are at most $\binom{m+c-1}{m} = \binom{m+c-1}{c-1} \le \binom{k+c-1}{c-1}$ permutations. The algorithm constructs $O(n)$ tables and each table consists of $O(kn)$ entries. To compute each entry, the time is

$$O\left(\binom{k+c-1}{c-1} c \right) = O((k+c-1)^{c-1} c/(c-1)!) = O((\max\{k, c\})^{c-1}).$$

Thus, the time complexity of the algorithm is $O(kn^2(\max\{k, c\})^{c-1})$. We summarize the discussion above in the following theorem:

Theorem 2. *Given a communication tree with n nodes and at most c children for each parent, let k be the maximum number of storage nodes that may be deployed in the tree. Then, the optimal tree with the minimum energy cost can be constructed by a dynamic programming algorithm in $O(kn^2(\max\{k, c\})^{c-1})$ time.*

5.2 Regular Trees

Now we consider a special case of LIMITED, where the given network is a regular tree with exactly c children for each nonleaf node and all leaves at the same level. For such a c-ary regular tree, we can modify Algorithm 2 to achieve a faster time complexity by making use of the regularity of the tree structure.

First, any subtree in a regular tree is also a regular tree and nodes at the same level have the subtrees with the same topology. This suggests that instead of keeping a table for each node as in Algorithm 2, we may keep just one table for each level. We name the levels from bottom to top, with all leaves at level 0, all parents of the leaves at level 1, and finally, the root at level $\lfloor \log_c n \rfloor$. For each level h, we define a two-dimensional table $E_h[m, l]$ for $0 \le m \le k$ and $0 \le l \le \lfloor \log_c n \rfloor - 1$, which returns the energy cost incurred within the subtree rooted at level h plus the contribution from the nodes in the subtree to their ancestors. As used previously, m is still the maximum number of storage nodes to use in the subtree and l is the number of forwarding nodes between the root of the subtree and the storage ancestor closest to the root of the subtree.

We first define $Q_0(m)$ and $Q_1(m)$ as follows:

$$Q_0(m) = \begin{cases} 0 & \text{if } m = 0; \\ \dfrac{e_{tr} + c \cdot e_{re}}{e_{tr} + e_{re}} r_q s_q & \text{if } m \ge 1. \end{cases} \quad Q_1(m) = Q_0(m-1).$$

Let $H = \lfloor \log_c n \rfloor$. Algorithm 3 first computes the table $E_0[m, l]$ for all leaves at level 0, for $0 \le m \le k$ and $0 \le l \le H - 1$ in lines 2-5. Then it works its way up, level by level, until level $H - 1$ in lines 6-11. The root, which is at level H,

is treated in lines 12 and 13, differently from the other nodes since it must be a storage node. By using one table for each level, our algorithm will construct $\lfloor \log_c n \rfloor$ tables. This will result in savings in both space and time, compared with our algorithm for arbitrary trees, which needs to construct n tables. The modified Algorithm 3 for regular trees has a time complexity of $O(k(\log n)^2(\max\{k,c\})^{c-1})$.

Algorithm 3. c-ary Regular Tree
1: $H = \lfloor \log_c n \rfloor$
2: **for** $m = 0$ to k **do**
3: **for** $l = 0$ to $H-1$ **do**
4: **if** $m = 0$ **then** $E_0[m,l] = (l+1)r_d s_d + (H-l)r_q \alpha s_d$
5: **if** $m \geq 1$ **then** $E_0[m,l] = (H+1)r_q \alpha s_d$
6: **for** $h = 1$ to $H-1$ **do**
7: **for** $m = 0$ to k **do**
8: **for** $l = 0$ to $H-1$ **do**
9: $min1 = \min_{\forall p \in P(m)}\{\sum_{j=1}^{c} E_h[m_j^p, l+1]\}$
$+ (l+1)r_d s_d + (H-h-l)r_q \alpha s_d + Q_0(m)$
10: $min2 = \min_{\forall p \in P(m-1)}\{\sum_{j=1}^{c} E_h[m_j^p, 0]\}$
$+ (H-h+1)r_q \alpha s_d + Q_1(m)$
11: $E_h[m,l] = \min\{min1, min2\}$
12: $E_H[k,0] = \min_{\forall p \in P(k-1)}\{\sum_{j=1}^{c} E_{H-1}[m_j^p, 0]\}$
$+ r_q \alpha s_d + Q_1(m)$
13: return $E_H[k,0]$

The result of the algorithm can be summarized in the following theorem:

Theorem 3. *Given a c-ary regular tree with n nodes, let k be the maximum number of storage nodes that may be deployed. Then, the optimal tree with the minimum energy cost can be constructed by a dynamic programming algorithm in $O(k(\log n)^2(\max\{k,c\})^{c-1})$ time.*

6 Stochastic Analysis

The algorithms in the previous sections aim to find the optimal locations for storage nodes. In practice, however, sensors including the storage nodes may not be deployed in a deterministic way. Instead, random deployment of sensors is commonly seen in applications. In this section, we analyze the performance of random deployment of storage nodes in both fixed tree and dynamic tree models.

6.1 Fixed Tree Model

Assume that the forwarding nodes and storage nodes are randomly distributed to the field with density λ and λ_s, respectively. For simplicity, we consider a disk network field, where the sink is placed at the center and R is the radius. In the fixed tree model, the network builds a communication tree in which each node finds the shortest path to the sink by following the tree edges. Each forwarding node sends its data to the first ancestor storage node on the path to the sink. As our simulation and other previous research show, the radius (r_i) of the area covered by the nodes that are i hops or less from the sink is proportional to i. Let this ratio be $c' = \frac{r_i}{i}$, Thus, we can estimate the number of nodes whose distances to the sink are between $(t-1)c'$ and tc', i.e., the nodes with depth t. Let $num(t)$ represent the total number of the nodes whose depth values are t,

$$num(t) = \lambda\pi(t^2c'^2 - (t-1)^2c'^2) = \lambda\pi(2t-1)c'^2.$$

For a node with depth t, let $s(t)$ be the expected hop distance to its closest storage ancestor. The cost caused by this node is $r_d s_d \cdot s(t) + r_q \alpha s_d \cdot (t - s(t))$. The probability that an individual node is a storage node is $p = \frac{\lambda_s}{\lambda}$. Therefore,

$$\begin{aligned} s(t) &= p \cdot 0 + p(1-p) \cdot 1 + p(1-p)^2 \cdot 2 + \cdots \\ &\quad + p(1-p)^{t-1} \cdot (t-1) + (1-p)^t \cdot t \\ &= \left(\frac{1}{p} - 1\right)(1 - (1-p)^t). \end{aligned}$$

The total energy cost in this model can be expressed as

$$E = \sum_{t=1}^{\frac{R}{c'}} num(t)(r_d s_d s(t) + r_q \alpha s_d (t - s(t))) + E_q,$$

where E_q is the cost of query diffusion. The value of c' is related to the communication range and node density. We can obtain the value from simulation.

Query messages are diffused from the sink to every storage node. For each storage node, it incurs an extra query diffusion cost along the path to its closest storage ancestor. If we assume that there is no overlap among the paths connecting each storage node and its closest storage ancestor, the total query diffusion cost E_q can be formulated as

$$E_q = \sum_{t=1}^{\frac{R}{c'}} num'(t) r_q s_q s(t), \quad (1)$$

where $num'(t) = \lambda_s \pi(2t-1)c'^2$ is the number of storage nodes whose depth values are t and recall that $s(t)$ is the expected distance to the closest storage ancestor.

6.2 Dynamic Tree Model

The fixed tree model assumes that the communication tree does not change according to the placement of the storage nodes. In the dynamic tree model, after the storage nodes have been positioned, each sensor node chooses the best storage node for storage with respect to the minimal communication cost for data forwarding and query diffusion and reply. The storage node placement in this model is more complicated than that in the fixed tree model because we need to consider the interplay between the storage node placement and the selection of the storage node for each sensor. These two steps affect each other dynamically.

In the optimal solution, a storage node should send query reply to the sink by following the shortest path because the data from a storage node cannot be further reduced according to the definition of data reduction function. A forwarding node has to choose a storage node for data storage to minimize the total communication cost for its data including raw data transfer to the storage node and query reply from the storage node to the sink. Therefore, the closest storage node may not be the best choice if it is further away from the sink yielding more cost on query reply. Assume that the sink is located at the origin. Let $\vec{x_i}$ represent the location of sensor i and $fd(\vec{x_i})$ indicate the location of the forwarding destination (storage node) assigned to sensor i. If i is a storage node, then $fd(\vec{x_i}) = \vec{x_i}$. The energy cost of sending raw data from $\vec{x_i}$ to $fd(\vec{x_i})$ is $r_d s_d |\vec{x_i} - fd(\vec{x_i})|$. The query reply cost for the data from forwarding node i is $r_q \alpha s_d |fd(\vec{x_i})|$. In total,

the cost generated by a single node i in a time unit is $r_d s_d|\vec{x_i} - fd(\vec{x_i})| + r_q \alpha s_d |fd(\vec{x_i})|$. To find the optimal solution, we need to minimize the cost for each sensor.

The total energy cost in this model is

$$E = \sum_i (r_d s_d|\vec{x_i} - fd(\vec{x_i})| + r_q \alpha s_d |fd(\vec{x_i})|) + E_q, \quad (2)$$

where E_q is the cost of query diffusion. We find that E_q in this dynamic tree model is the same as that in the fixed tree model. Because in both models, each storage node is connected to the sink by the shortest path. Therefore, we can also use (1) to estimate E_q. In the following of this section, we will analyze the rest part of (2), which is denoted by E'.

First, let us consider a scenario where a sensor at location $\vec{x}$ forwards its raw data to a storage node at location $\vec{y}$. We define a function $F(\vec{x}, \vec{y})$ as the energy cost caused by this sensor, $F(\vec{x}, \vec{y}) = r_d s_d |\vec{x} - \vec{y}| + r_q \alpha s_d |\vec{y}|$. There are requirements for this scenario to be valid. Essentially, for the sensor at $\vec{x}$, no other storage node could yield less cost than the one at $\vec{y}$. Next, we will derive this condition. We define an area $G(\vec{x}, U) = \{\vec{y} | F(\vec{x}, \vec{y}) \leq U\}$, that is, if a sensor at $\vec{x}$ selects any storage node in that area, the energy cost for the data of that sensor would be no more than U. Thus, $G(\vec{x}, F(\vec{x}, \vec{y})) \cap S = \phi$ is the requirement for the above scenario. Considering the Poisson deployment of sensors, the minimum reply cost can be expressed as

$$E' = \lambda \iint P(\vec{y} \in S) F(\vec{x}, \vec{y}) P(G(\vec{x}, F(\vec{x}, \vec{y})) \cap S = \phi) dy dx,$$

where S is the set of all storage nodes including the sink. $P(\vec{y} \in S)$ is the probability that there is a storage node at location $\vec{y}$,

$$P(\vec{y} \in S) = \begin{cases} 1 & \text{if } \vec{y} \text{ is the origin;} \\ \lambda_s, & \text{otherwise.} \end{cases}$$

$P(G(\vec{x}, F(\vec{x}, \vec{y})) \cap S = \phi)$ represents the probability of $G(\vec{x}, F(\vec{x}, \vec{y})) \cap S = \phi$. As we described before, this condition means that there is no other storage node in the area $G(\vec{x}, F(\vec{x}, \vec{y}))$. According to the Poisson process of deploying storage nodes with the density λ_s,

$$P(G(\vec{x}, F(\vec{x}, \vec{y})) \cap S = \phi) = \begin{cases} e^{-\lambda_s |G(\vec{x}, F(\vec{x}, \vec{y}))|} & \text{if } F(\vec{x}, \vec{y}) \leq F(\vec{x}, 0); \\ 0, & \text{otherwise.} \end{cases}$$

Unlike other nodes, the sink is deterministically fixed in the network. So if area G covers the sink, there is no need to compute the probability. The forwarding node will definitely send data directly to the sink.

However, $|G|$ in the formula above, called the Cartesian Oval, cannot be expressed in a closed form. To approximate the energy cost, we make each forwarding node simply choose the closest storage node for data storage. The network field is then divided into Voronoi cells induced by storage nodes. The energy cost of this topology is very close to the optimal case, especially when $\lambda_s \ll \lambda$.

Assume that there is a forwarding node i at location $\vec{x}$, the probability that i sends data through a storage node at location $\vec{y}$ becomes

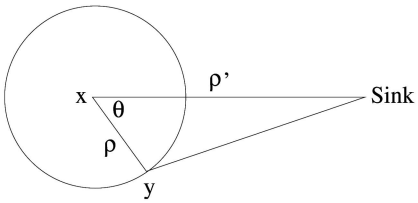

Fig. 5. A forwarding node at location $\vec{x}$ sends data via a storage node at $\vec{y}$.

$$P(\vec{x} \to \vec{y}) = \begin{cases} 0 & \text{if } |\vec{x} - \vec{y}| \geq |\vec{x}|; \\ e^{-\lambda_s \pi |\vec{x}|^2} & \text{if } \vec{y} \text{ is the origin;} \\ \lambda_s e^{-\lambda_s \pi |\vec{x} - \vec{y}|^2}, & \text{otherwise.} \end{cases}$$

Thus,

$$\begin{aligned} E' &= \lambda \iint F(\vec{x}, \vec{y}) P(\vec{x} \to \vec{y}) dy dx \\ &= \lambda \int F(\vec{x}, 0) e^{-\lambda_s \pi |\vec{x}|^2} dx \\ &\quad + \lambda \iint_{|\vec{x} - \vec{y}| < |\vec{x}|} F(\vec{x}, \vec{y}) \lambda_s e^{-\lambda_s \pi |\vec{x} - \vec{y}|^2} dy dx. \end{aligned} \quad (3)$$

In the first term, $F(\vec{x}, 0) = r_d s_d |\vec{x}|$. Therefore,

$$\begin{aligned} \lambda \int F(\vec{x}, 0) e^{-\lambda_s \pi |\vec{x}|^2} dx &= \lambda r_d s_d \int |\vec{x}| e^{-\lambda_s \pi |\vec{x}|^2} dx \\ &= 2\pi \lambda r_d s_d \int_0^R \rho^2 e^{-\lambda_s \pi \rho^2} d\rho. \end{aligned} \quad (4)$$

For the second term in (3), Fig. 5 shows the variables after coordination conversions, where $\rho = |\vec{x} - \vec{y}|$ and $\rho' = |\vec{x}|$. $F(\vec{x}, \vec{y})$ can be expressed by $r_d s_d \rho + r_q \alpha s_d \sqrt{\rho'^2 + \rho^2 - 2\cos\theta \rho' \rho}$. Thus, the second term becomes

$$\begin{aligned} &\lambda \iint_{|\vec{x} - \vec{y}| < |\vec{x}|} F(\vec{x}, \vec{y}) \lambda_s e^{-\lambda_s \pi |\vec{x} - \vec{y}|^2} \, dy dx \\ &= 4\pi^2 \lambda \lambda_s r_d s_d \int_0^R \int_0^{\rho'} e^{-\lambda_s \pi \rho^2} \rho^2 \rho' d\rho d\rho' \\ &+ 2\pi \lambda \lambda_s r_q \alpha s_d \int_0^R \int_0^{2\pi} \int_0^{\rho'} e^{-\lambda_s \pi \rho^2} \\ &\rho \rho' \sqrt{\rho'^2 + \rho^2 - 2\cos\theta \rho' \rho} \, d\rho d\theta d\rho'. \end{aligned} \quad (5)$$

Combining (4) and (5), the total energy cost except query diffusion is

$$\begin{aligned} E' &= 2\pi \lambda r_d s_d \int_0^R \rho^2 e^{-\lambda_s \pi \rho^2} d\rho \\ &+ 4\pi^2 \lambda \lambda_s r_d s_d \int_0^R \int_0^{\rho'} e^{-\lambda_s \pi \rho^2} \rho^2 \rho' d\rho d\rho' \\ &+ 2\pi \lambda \lambda_s r_q \alpha s_d \int_0^R \int_0^{2\pi} \int_0^{\rho'} e^{-\lambda_s \pi \rho^2} \\ &\rho \rho' \sqrt{\rho'^2 + \rho^2 - 2\cos\theta \rho' \rho} \, d\rho d\theta d\rho'. \end{aligned}$$

We can further approximate E' by examining its two components separately. Let E_{fs} be the cost of transferring raw data between forwarding nodes and their closest storage nodes. Let E_{ss} be the cost to relay reply from storage nodes to the sink. For a forwarding node i, the expected distance to the closest storage node is $\frac{1}{2\sqrt{\lambda_s}}$. Thus,

$$E_{fs} = \lambda\pi R^2 r_d s_d \frac{1}{2\sqrt{\lambda_s}}, \quad (6)$$

where $\lambda\pi R^2$ is the total number of forwarding nodes and $r_d s_d \frac{1}{2\sqrt{\lambda_s}}$ is the energy cost of transferring raw data from an individual forwarding node to its closest storage node. In addition, $E_{ss} = r_q \alpha s_d \sum_{i\in S}(D_i + 1)L_i$, where D_i is the total number of descendants and L_i is the distance to the sink. Since each forwarding node chooses the closest storage node for data storage, the number of forwarding nodes that each storage node is responsible for is approximately the same. If we replace L_i by the mean value L', then $E_{ss} = r_q \alpha s_d L' \sum_{i\in S}(D_i + 1)$, where $\sum_{i\in S}(D_i + 1)$ represents the number of nodes that send data via storage nodes to the sink. Let N' be the number of forwarding nodes that send data directly to the sink, $\sum_{i\in S}(D_i + 1) = \lambda\pi R^2 - N'$. N' can be derived as

$$N' = \lambda \int e^{-\lambda_s \pi |\vec{x}|^2} dx = \lambda \int_0^{2\pi} \int_0^R e^{-\lambda_s \pi \rho^2} \rho d\rho d\theta$$
$$= 2\pi\lambda \int_0^R \rho e^{-\lambda_s \pi \rho^2} d\rho = \frac{\lambda}{\lambda_s}(1 - e^{-\lambda_s \pi R^2}).$$

And L' can be simply approximated as

$$L' = \frac{\lambda_s \int_0^R 2\pi r \cdot r dr}{\lambda_s \pi R^2} = \frac{2}{3}R.$$

Therefore,

$$E_{ss} = r_q \alpha s_d \left(\frac{2}{3} R \left(\lambda\pi R^2 - \frac{\lambda}{\lambda_s}(1 - e^{-\lambda_s \pi R^2})\right)\right). \quad (7)$$

Combining (6) and (7),

$$E' = \lambda\pi R^2 r_d s_d \frac{1}{2\sqrt{\lambda_s}} + r_q \alpha s_d \left(\frac{2}{3} R \left(\lambda\pi R^2 - \frac{\lambda}{\lambda_s}(1 - e^{-\lambda_s \pi R^2})\right)\right).$$

7 Performance Evaluation

Our evaluation is based on simulation. In this section, we will present the performance results of the proposed algorithms.

7.1 Simulation Settings

In our simulation, we consider a network of sensors deployed on a disk of radius 5 with the sink placed at the center. One thousand sensor nodes ($n = 1{,}000$) are deployed to the field randomly following two-dimensional spatial Poisson process. Node transmission range is set to 0.65. After all nodes are deployed, a routing tree rooted at the sink is constructed by flooding a message from the sink to all the nodes in the network. As we mentioned in Section 3, the message carries the number of hops it travels so that each node chooses among its neighbors the node that has the minimum number of hops to be its parent. This tree topology is needed in the simulation of the fixed tree model. This step, however, can be skipped for the dynamic tree model.

In the rest of this section, we will present and compare the following algorithms:

- **FT-DD**: It represents the fixed tree model with deterministic deployment. In FT-DD, the storage nodes are deployed by following the dynamic programming algorithm according to the known tree topology.
- **FT-RD**: It represents the fixed tree model with random deployment. In FT-RD, we randomly select a certain number of nodes in the network to be storage nodes.
- **DT-RD**: It represents the dynamic tree model with random deployment. In this algorithm, the storage nodes are randomly deployed. After that, each forwarding node selects the best storage node to deliver data and each storage node replies to query by following the shortest path to the sink.
- **ST-RD**: It represents semidynamic tree model with random deployment, which is the enhanced version of FT-RD with a local adjustment. When a sensor i is upgraded to storage node in a tree structure, its siblings' children will try to set i as their parents if i is within their communication range.
- **Greedy**: It represents a greedy algorithm where the most heavily loaded sensors will be upgraded to storage nodes. Usually, those sensors close to the sink will become storage nodes in this algorithm.

This list does not include the existing in-network storage approaches such as data-centric storage because the problem settings are different for a comparison. Approximately, the performance of data-centric storage is close to the random deployment in dynamic tree model in this paper since the locations of storage sites are determined by hash values.

In addition, we use relative energy cost as performance metrics. We measure the cost when no storage node except the sink is deployed as the baseline. Let the energy cost in this no storage scenario be E_f. And let the energy cost after the storage nodes are deployed be E. The relative energy cost is defined as $\frac{E}{E_f}$, represented as a percentage. In the rest of this section, we use "energy cost" for "relative energy cost."

Due to the randomness of our simulation environment, results from the same parameter setting might vary a lot. Therefore, for a certain set of parameters, we conduct 100 independent trials and the average energy cost is used in the following analysis and comparison. Unless otherwise stated, we set the following parameters in our simulations: $r_d = r_q = s_d = s_q = 1$ and $\alpha = 0.5$. We evaluate the energy cost by varying the number of storage nodes k and the data reduction rate α. Note that the energy cost is also related to r_d, s_d, r_q, and s_q. However, for comparison purpose, changing r_q, s_q, r_d, s_d will be equivalent to changing α. To simplify the description, we fix r_q, s_q, r_d, s_d and only vary α to examine different characteristics of data and queries.

7.2 Random Deployment

Fig. 6 shows the energy cost of random deployment in the fixed tree model. We compare our theoretical estimation with simulation results. As we can see from the figure, the theoretical estimation and the simulation match well. We have examined the simulation carefully and found that many storage nodes are placed at the leaf nodes or have very few descendants. Therefore, the data reduction for those descendants is negligible and less energy is saved compared to the case that each node sends all the data to the sink.

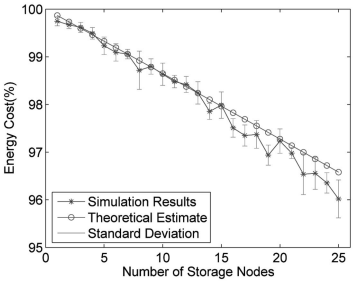

Fig. 6. FT-RD: Energy cost with varying number of storage nodes (k).

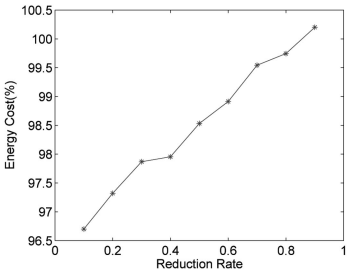

Fig. 7. FT-RD: Tthe impact of data reduction rate (α).

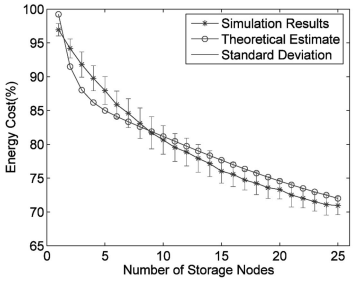

Fig. 8. DT-RD: Energy cost with varying number of storage nodes (k).

In Fig. 7, we show the energy cost with respect to different data reduction rates α. We fix the number of storage nodes ($k = 10$) and change the data reduction rate α from 0.1 to 0.9. In this fixed tree model, decreasing data reduction rate cannot improve the performance too much. Even when α is set to 0.1, we still have more than 96 percent energy cost with 10 storage nodes. The reason is that data accumulation to the storage nodes from the forwarding nodes consumes most of the energy with respect to the query diffusion and reply. Moreover, when α is 0.9, the energy cost is even worse than the original cost because the incurred query diffusion cost becomes larger than the benefits obtained.

The energy cost of random deployment in the dynamic tree model is shown in Fig. 8. In this model, the location of each storage node is broadcast to forwarding nodes so that they can choose the proper storage nodes to deliver data for the energy concern. In this way, we take full advantage of every storage node and maximize their contributions to the whole network. As shown in Fig. 8, random deployment performs much better in this dynamic tree model. The energy cost decreases very fast with increasing number of storage nodes, e.g., with 10 storage nodes (1 percent of total nodes), we can save energy by approximately 20 percent.

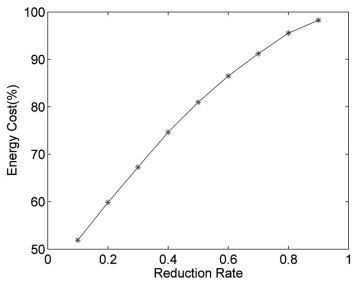

Fig. 9. DT-RD: The impact of data reduction rate (α).

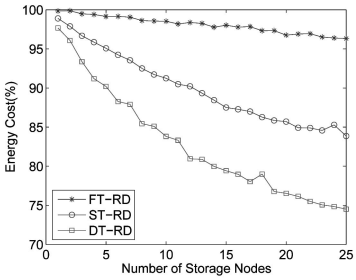

Fig. 10. Comparison of energy cost with varying number of storage nodes (k).

Fig. 9 illustrates the impact of data reduction rate to the energy cost in the dynamic tree model. This time, α becomes an important parameter because every storage node is in charge of many forwarding nodes in the dynamic tree model. A small decrease of α will reduce energy cost greatly. We also observe that our theoretical estimation matches well with the simulation results. Although our stochastic analysis uses some expected values and approximations, the maximum difference between the two curves is less than 5 percent.

As shown above, with random deployment, the dynamic tree model has a significant performance improvement over the fixed tree model. However, the locations of storage nodes need to be broadcast to all other nodes and the new tree is completely different from the originally constructed tree one. We consider a semidynamic tree model in which local adjustments are applied to the originally constructed tree. For each storage node i, all the forwarding nodes within the transmission range of i that have a depth no less than i's depth select i as parent. As a result, each storage node gains more descendants and accepts more raw data storage. Fig. 10 compares the energy cost of random deployment in three models (fixed tree, dynamic tree, and semidynamic tree), as well as deterministic deployment in the fixed tree model. As shown in Fig. 10, DT-RD achieves the best performance, while FT-RD has the worst performance. Local adjustment in ST-RD improves the performance of the fixed tree model. In FT-RD, each storage node has no control about how many descendants it can have. Many storage nodes are deployed with few descendants, which explains why FT-RD delivers the worst performance. ST-RD allows each storage node to have some restrained flexibility in choosing its descendants, and has a better performance than FT-RD. DT-RD has more flexibility in choosing descendants, and we see a much improved performance.

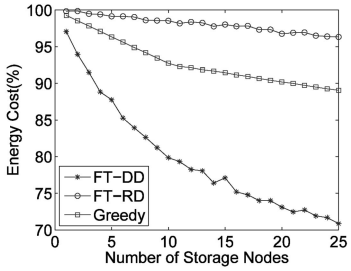

Fig. 11. Comparison of FT-DD, FT-RD, and Greedy: Energy cost with varying number of storage nodes (k).

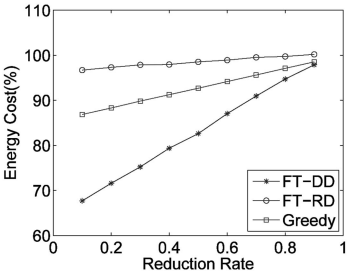

Fig. 12. Comparison of FT-DD, FT-RD, and Greedy: The impact of data reduction rate (α).

TABLE 2
Depth Distribution (D: Depth, #: Number of Sensors)

D	1	2	3	4	5	6	7	8	9	10	11
#	10	32	48	74	108	128	176	189	164	65	6

7.3 Deterministic Deployment

Figs. 11 and 12 illustrate the performance of deterministic deployment in the fixed tree model. In the simulation, the locations of the storage nodes are obtained by Algorithm 2. Compared to the random deployment (FT-RD) and greedy algorithm, deterministic deployment significantly improves the performance by precisely computing the optimal positions to put the storage nodes.

In Fig. 11, we fix the reduction rate $\alpha = 0.5$, and vary the number of storage nodes from 2 to 25. With a few storage nodes, the energy cost is sharply reduced in Fig. 11. When k becomes larger, the slope gradually becomes flatter. As shown in the figure, we can save about 20 percent energy cost with 10 storage nodes, and 30 percent with 25 storage nodes. In addition, Fig. 12 shows the energy cost with varying reduction rate, when the number of storage nodes is fixed as 10. As we can see, the energy cost is nearly linear to the reduction rate α.

Intuitively, the performance in the fixed tree model depends on which level of the tree the storage nodes are deployed at. When storage nodes are close the leaves, which often happens in FT-RD, the benefit of data reduction is limited. On the other hand, when storage nodes are deployed close to the sink as in the greedy algorithm, a large amount of raw data have to traverse a long path to reach the storage nodes still yielding high energy cost. The locations derived from our algorithm usually reside in the middle levels. Here is an example of the result. Table 2 shows the depth distribution in a particular tree structure. The depth of the sink is 0. When we deploy 10 storage nodes in such a tree, the derived depths of storage nodes are 3, 4, 4, 4, 4, 5, 5, 6, 6, and 6.

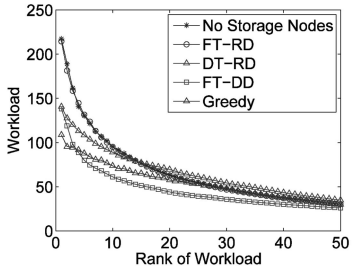

Fig. 13. Comparison of load balancing: Workload is measured as the size of the messages sent by each sensor per unit time.

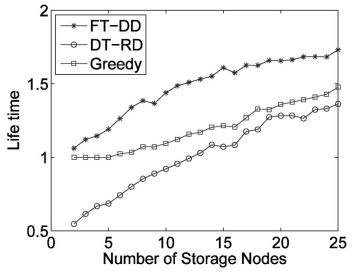

Fig. 14. Comparison of lifetime with varying number of storage nodes (k): values are normalized by the lifetime without storage nodes.

7.4 Network Life

Finally, load distribution and network lifetime are shown in Figs. 13 and 14. We show the workloads of the most heavily loaded 50 nodes in Fig. 13. In Fig. 14, we define lifetime as the time that 2 percent nodes are depleted of energy. In our setting, it means that 20 sensors are out of operation. According to our simulation, this scenario will usually cause disconnection in the sensor network. As we can see in Fig. 13, FT-RD almost has no improvement on load balancing and lifetime. In contrast, FT-DD lengthens the lifetime a lot with a small number of storage nodes, although the objective of our algorithm is to minimize the total energy cost. For example, with 15 storage nodes, the lifetime is increased by more than 60 percent. DT-RD does not perform well with only a few storage nodes because the sensors connecting storage nodes and the sink carry a lot of workloads for both raw data transmission and reply collection. The greedy algorithm is superior to DT-RD by specifically reducing the energy cost of the most heavily loaded nodes.

8 Conclusion

This paper considers the storage node placement problem in a sensor network. Introducing storage nodes into the sensor network alleviates the communication burden of sending all the raw data to a central place for data archiving and facilitates the data collection by transporting data from a limited number of storage nodes. In this paper, we examine how to place storage nodes to save energy for data collection and data query. We consider two models: fixed tree model and dynamic tree model. In the first model, we give exact solution on how to place storage nodes to minimize total energy cost. Using stochastic analysis, we also give the

performance estimation for both models under the assumption that the storage nodes are deployed randomly.

ACKNOWLEDGMENTS

This project was supported in part by US National Science Foundation grant nos. CNS-0721443, CNS-0831904, and CAREER Award CNS-0747108.

REFERENCES

[1] P. Gupta and P.R. Kumar, "The Capacity of Wireless Networks," *IEEE Trans. Information Theory,* vol. 46, no. 2, pp. 388-404, Mar. 2000.
[2] E.J. Duarte-Melo and M. Liu, "Data-Gathering Wireless Sensor Networks: Organization and Capacity," *Computer Networks (COMNET),* vol. 43, no. 4, pp. 519-537, Nov. 2003.
[3] R.C. Shah, S. Roy, S. Jain, and W. Brunette, "Data MULEs: Modeling a Three-Tier Architecture for Sparse Sensor Networks," *Proc. First IEEE Int'l Workshop Sensor Network Protocols and Applications (SPNA),* May 2003.
[4] B. Sheng, Q. Li, and W. Mao, "Data Storage Placement in Sensor Networks," *Proc. ACM MobiHoc,* pp. 344-355, 2006.
[5] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong, "TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks," *SIGOPS Opererating Systems Rev.,* vol. 36, no. SI pp. 131-146, 2002.
[6] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong, "The Design of an Acquisitional Query Processor for Sensor Networks," *Proc. ACM SIGMOD,* pp. 491-502, 2003.
[7] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-Efficient Communication Protocols for Wireless Microsensor Networks," *Proc. Int'l Conf. System Sciences,* Jan. 2000.
[8] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin, "Data-Centric Storage in Sensornets," *SIGCOMM Computer Comm. Rev.,* vol. 33, no. 1, pp. 137-142, 2003.
[9] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu, "Data-Centric Storage in Sensornets with GHT, a Geographic Hash Table," *Mobile Networks and Applications,* vol. 8, no. 4, pp. 427-442, 2003.
[10] J. Newsome and D. Song, "GEM: Graph Embedding for Routing and Data-Centric Storage in Sensor Networks without Geographic Information," *Proc. First Int'l Conf. Embedded Networked Sensor Systems,* pp. 76-88, 2003.
[11] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker, "Geographic Routing Made Practical," *Proc. Second USENIX Symp. Networked Systems Design and Implementation (NSDI '05),* May 2005.
[12] C.T. Ee, S. Ratnasamy, and S. Shenker, "Practical Data-Centric Storage," *Proc. Third USENIX Symp. Networked Systems Design and Implementation (NSDI '06),* May 2006.
[13] J. Ahn and B. Krishnamachari, "Fundamental Scaling Laws for Energy-Efficient Storage and Querying in Wireless Sensor Networks," *Proc. Seventh ACM Int'l Symp. Mobile Ad Hoc Networking and Computing,* pp. 334-343, 2006.
[14] D. Ganesan, B. Greenstein, D. Estrin, J. Heidemann, and R. Govindan, "Multiresolution Storage and Search in Sensor Networks," *ACM Trans. Storage,* vol. 1, no. 3, pp. 277-315, 2005.
[15] M. Li, D. Ganesan, and P. Shenoy, "PRESTO: Feedback-Driven Data Management in Sensor Networks," *Proc. Third USENIX Symp. Networked Systems Design and Implementation (NSDI '06),* May 2006.
[16] A. Demers, J. Gehrke, R. Rajaraman, N. Trigoni, and Y. Yao, "The Cougar Project: A Work-in-Progress Report," *SIGMOD Record,* vol. 32, no. 4, pp. 53-59, 2003.
[17] J. Gehrke and S. Madden, "Query Processing in Sensor Networks," *IEEE Pervasive Computing,* vol. 3, no. 1, pp. 46-55, Jan. 2004.
[18] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy, "Ultra-Low Power Data Storage for Sensor Networks," *Proc. Fifth Int'l Conf. Information Processing in Sensor Networks,* pp. 374-381, 2006.
[19] D. Zeinalipour-Yazti, S. Lin, V. Kalogeraki, D. Gunopulos, and W.A. Najjar, "MicroHash: An Efficient Index Structure for Flash-Based Sensor Devices," *Proc. Fourth USENIX Conf. File and Storage Technologies,* 2005.
[20] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy, "CAPSULE: An Energy-Optimized Object Storage System for Memory-Constrained Sensor Devices," *Proc. Fourth Int'l Conf. Embedded Networked Sensor Systems,* 2006.
[21] F. Baccelli, M. Klein, M. Lebourges, and S. Zuyev, "Stochastic Geometry and Architecture of Communication Networks," *J. Telecomm. Systems,* vol. 7, pp. 209-227, 1997.
[22] F. Baccelli and S. Zuyev, "Poisson-Voronoi Spanning Trees with Applications to the Optimization of Communication Networks," *Operations Research,* vol. 47, no. 4, pp. 619-631, 1999.
[23] S.J. Baek, G. de Veciana, and X. Su, "Minimizing Energy Consumption in Large-Scale Sensor Networks through Distributed Data Compression and Hierarchical Aggregation," *IEEE J. Selected Areas Comm.,* special issue on fundamental performance limits of wireless sensor networks, vol. 22, no. 6, pp. 1130-1140, Aug. 2004.
[24] B. Sheng, C.C. Tan, Q. Li, and W. Mao, "An Approximation Algorithm for Data Storage Placement in Sensor Networks," *Proc. Second Int'l Conf. Wireless Algorithms, Systems and Applications (WASA '07),* Aug. 2007.

Bo Sheng received the PhD degree in computer science from the College of William and Mary in 2010. He is an assistant professor in the Department of Computer Science at the University of Massachusetts Boston. His research interests include wireless networks and embedded systems with an emphasis on the efficiency and security issues. He is a member of the IEEE.

Qun Li received the PhD degree in computer science from Dartmouth College. He is an associate professor in the Department of Computer Science at the College of William and Mary. His research interests include wireless networks, sensor networks, RFID, and pervasive computing systems. He received the US National Science Foundation CAREER Award in 2008. He is a member of the IEEE.

Weizhen Mao received the PhD degree in computer science from Princeton University under the supervision of Andrew C.-C. Yao. She is currently an associate professor in the Department of Computer Science at the College of William and Mary. Her current research interests include the design and analysis of online and approximation algorithms for problems arising from application areas, such as multicore job scheduling and wireless and sensor networks.