

# Data Aggregation in Sensor Networks: No More a Slave to Routing

Ravi Prakash

Department of Computer Science  
University of Texas at Dallas  
Richardson, TX 75080  
Email: ravip@utdallas.edu

Ehsan Nourbakhsh

Department of Computer Science  
University of Texas at Dallas  
Richardson, TX 75080  
Email: ehsan@utdallas.edu

Kunal Sahu

Department of Computer Science  
University of Texas at Dallas  
Richardson, TX 75080  
Email: kunal@utdallas.edu

**Abstract**—Application-specific data aggregation can play a significant role in energy-efficient operation of wireless sensor networks. Existing aggregation techniques rely heavily on the routing protocol to build shortest paths to route node measurements to the base station and are limited in the types of supported queries. We propose an aggregation scheme that utilizes the inherent information gradients present in the network. The query is directed to the source of information, resulting in better load sharing in the network. We support a variety of queries ranging from simple maximum, minimum or average of the readings of sensor nodes to more complex quantile queries such as  $k$  highest values or  $k^{\text{th}}$  highest value through a generic query algorithm. The query algorithm shifts the computation to the querying agent, thus eliminating any in-network aggregation.

## I. INTRODUCTION

Several tree-based data aggregation protocols have been proposed to reduce the energy-consumption and communication overhead in wireless sensor networks. The non-leaf nodes in the tree act as aggregators by fusing the data received from their descendants and forwarding it towards their parent on the tree. These are general-purpose data aggregation schemes that overlook the characteristics of the application that requires this aggregation functionality. We feel that data aggregation is much more than just forming an optimum path to route the data from the sensor nodes to the sink and the particular application should be taken into consideration while designing an aggregation protocol. We propose a new approach to data aggregation by constructing a completely distributed data structure in the network based on the information gradients present in the network. This data structure helps answer application specific aggregation queries in an energy-efficient manner without having to rely on the underlying routing protocol.

## II. RELATED WORK

Recently a flurry of work [4], [5], [7], [9], [14], [15], [19], [23] has been performed in the field of in-network data aggregation especially for wireless sensor networks. For aggregate queries it is accepted that the computations are performed in-network whenever possible [14], [15]. With the in-network approach, sensor readings are accumulated into partial results and these results are combined as they propagate towards the base station. Each sensor node has to transmit only one message and relay a few others, thus saving

considerable energy. Existing aggregation protocols can be broadly classified into two categories:

**Tree-based:** In this approach (such as the TAG [14], TinyDB [15], [16], Cougar [7], and others [5], [9], [23]) a spanning tree is built, rooted at the querying agent. The query flows down the tree to all the nodes in the network and each query answer is generated by in-network aggregation along the tree, proceeding from the leaves to the root in a bottom-up fashion. One main advantage of this approach is that the aggregation is often simple and straightforward within the network. However most of these algorithms [7], [15], [23] tackle only simple aggregation problems such as max, min, sum or average. Also these tree-based protocols are prone to node and communication link failure which is relatively common in sensor network systems.

**Multi-path-based:** The multi-path approach [4], [19] permits arbitrary aggregation topologies beyond a normal tree structure. An advantage of the multi-path approach over the tree-based approach is lower communication error. Manjhi et al. [17] propose a *tributaries and delta* approach which combines the advantages of both the tree and multi-path approaches. Most of these works aim to reduce the communication error rate by using multiple paths available in a network, rather than adhering to a rigid tree topology. However they do not utilize the information gradients available in the network in a proactive manner.

Approximate aggregation schemes for more complex queries such as contours and wavelet histograms have also been proposed for the TinyDB system [10]. There have been other algorithms to solve the quantile queries such as median, consensus and range [23], [8], [3]. In data streams Greenwald and Khanna [8] have proposed an efficient approximation algorithm for computing quantiles. Manku and Motwani [18] have presented approximate algorithms for counting the frequent items in a network. However, they incur high communication cost dealing with simpler queries such as max, average or sum. We address this balance between answering simple as well as complex queries using the same global structure.

There has been significant work in the field of Information-Directed Routing in sensor networks [6], [12], [13], [21], which is slightly different from that of data aggregation. These schemes are based either on information-gradient [6], [12],

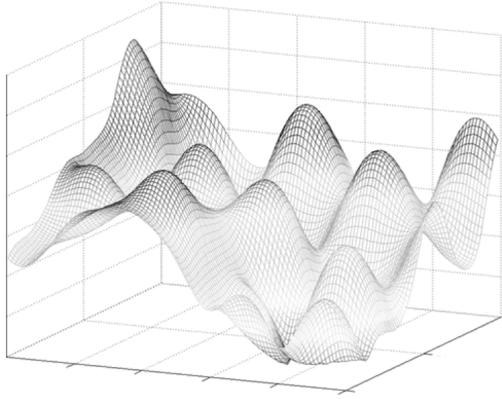


Fig. 1: Gradient-based Distribution

[13], flooding [11], or random walks [21]. The information-gradient approach uses sensors' measurements of the event to route queries to the information source present in the network. We focus on the information gradient to build a global tree structure which maintains information about all the sources present in the system and their relative positions. Any aggregate query is then handled through node traversals on the tree built on top of the information gradient, as explained later.

The proposed algorithm is proactive and utilizes the information gradients present in the network to construct the tree even in the absence of any queries. We preserve the properties of our tree in cases of local reading changes through message exchanges between neighbors (which could be piggybacked on heartbeat messages). Thus any query arriving in the system is just directed to the appropriate node for the required information. Without transmissions from all the nodes in the sensor network and performing an in-network aggregation we use our global data structure to answer a variety of queries in a distributed manner.

### III. PROBLEM DESCRIPTION AND BASIC IDEA

#### A. Motivation

Several physical phenomena exhibit the diffusion property [1], [22] with distance, i.e. the observed magnitude,  $f(d)$ , is proportional to  $1/d^\alpha$ , where  $d$  is the distance of the observer from the location of the event and  $\alpha$  is the diffusion parameter. Some examples are fire surveillance, temperature monitoring and atmospheric conditions survey. We take advantage of these information gradients in our aggregation solution.

Figure 1 represents a gradient-based distribution of node readings in a sensor network field. There are multiple peaks and the readings of the surrounding nodes slowly taper off based on the diffusion laws. For simplicity we consider such a distribution for our data aggregation scheme.

#### B. System Model

We assume an asynchronous point-to-point communication model. Communication links between pairs of nodes are assumed to be bi-directional. The network can be visualized

as an un-directed graph  $G = (V, E)$ , where the set of nodes is represented by  $V$ , the vertices of the graph and  $E$  is the set of edges of the graph representing the bi-directional communication links. This graph is assumed to be connected.

All sensor nodes in the network have distinct identities and each node is aware of its identity. There is no notion of a shared memory or global clock between the nodes. We assume an upper bound of the time it takes for a message sent by one node to be received by a neighbor. Furthermore, each node is aware of its one-hop neighbors, and the underlying MAC protocol ensures reliable communication.

During tree formation, as described in the next section, we assume synchronous mode of communication between a node and its neighbors. We can implement synchrony for this phase of the algorithm by using a simple  $\alpha$ -synchronizer of [2].

#### C. Basic Idea

We propose a Gradient-based Data Aggregation algorithm that constructs a spanning tree in the network based on the actual sensor readings of the nodes. Instead of burdening the nodes located closer to the sink in a hierarchical tree aggregation scheme, we take into account the actual readings of the sensors while constructing the global tree.

The main idea is to build a global tree rooted at the sensor node with the maximum reading. We describe our algorithm in the context of an application that is interested in tracking the  $x$ -highest values in the network, where  $x$  could be an integer variable. In the case of the  $x$ -lowest values the algorithm can be easily modified to build the tree rooted at the node with the lowest reading.

The sink, which is interested in knowing the maximum value in the network, can directly contact the root node to retrieve the maximum value.

Readings of the sensor nodes may change with time. So we employ periodic updates between nodes and their one-hop neighbors to detect any local peaks in the field or a change in the identity of the node reporting the highest reading. We ensure that the highest valued node is always the root of this globally constructed tree and pointers are allocated to keep track of the secondary peaks. The only assumption we make is that the rate of change in node readings is small compared to the stabilization time of our algorithm.

The secondary peaks in the fields are tracked using watch pointers as explained in Section V. These pointers guide the query algorithm while answering queries such as the  $x$  highest values in the sensor network. As the identity of the highest valued node changes we invoke the root reversal algorithm that handles the update of the global tree structure and local watch pointers as explained in Section VI. Our query algorithm (Section VII) uses these data structures to access the sensor nodes in descending order of their readings - usually following the links from a node to its children, and sometimes following the watch pointers. All the proposed algorithms are completely distributed and do not require any central authorization service for their execution.

## IV. TREE FORMATION

The spanning tree is rooted at the sensor node with the global maximum value. Each node maintains pointers to its parent and children in the tree. Any future changes in node readings are handled through the algorithms explained in the later sections.

### A. Data Structures

Each sensor node  $i$  maintains the following information:

- $Nbr_i[]$  : List of one-hop neighbors of  $i$ , obtained using any neighbor-discovery protocol.
- $SR_i$  : The reading of sensor node  $i$ . A sensor node could update its reading periodically.
- $GM_i$  : Tuple consisting of root node's id and reading of the global maximum heard by node  $i$ .
- $PPtr_i$  : Pointer to the parent node on the tree.
- $PR_i$  : Reading of the parent node.
- $CPtr_i[]$  : List of pointers to children nodes.
- $CR_i[]$  : List of readings of children nodes.
- $Dia_i$  : Estimate of the network diameter.
- $Cntr_i$  : Counter for detecting termination. We use this counter to keep track of repeated values of  $Dia_i$  and make sure that all the nodes in the network are aware of the global maximum.

### B. Algorithm

In an ideal scenario there would be a single peak and the readings of the other nodes would gradually decrease as we move farther away from the maximum valued node. However, in the presence of several peaks in the gradient distribution we cannot always guarantee this property of the constructed tree.

Every node that is a local peak initiates a flood of its own value, claiming that it is the global maximum. In every round, upon receiving these flood messages from neighbors, each sensor node takes a maximum of all the received values and sets its parent pointer to the node from whom it received the highest value. In such a fashion the flood of the actual global maximum spreads farther into the network and a tree, rooted at the node with the global maximum value, is constructed. The floods initiated by other local/secondary peaks are suppressed.

Now to determinate the termination of our algorithm we introduce the additional parameters  $Dia$  and  $Cntr$ . Since we do not assume any prior knowledge of the diameter, hence we add an extra parameter  $Dia$  to each flood message being sent in this phase. This indicates how far the global maximum has propagated in the network from the initiator of that value. For such a node having the global maximum value, say node  $g$ , it will receive increased values of  $Dia_g$  in every other round. These increased values would represent echoes of  $g$ 's flood message from successively farther nodes in the network. Therefore on receiving three successive identical values of  $Dia_g$ , which is captured in the  $Cntr_g$  parameter, the node  $g$  can safely deduce that the tree has been built completely and its message has been echoed from the farthest node from itself [20]. This signals the completion of the algorithm and

thereafter the root node initiates a terminate message down the tree. This message carries the identity of the root node and its latest reading so that all nodes in the network are aware of the root node's identity.

### C. Illustration

We illustrate the tree formation algorithm with the example shown in Figure 2. The numbers next to vertices indicate node readings. Each node independently executes the tree formation algorithm and the whole protocol works in a distributed fashion without any central coordination.

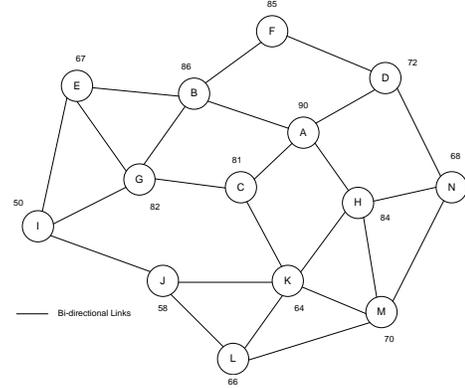


Fig. 2: Network Topology

The diffusion initiated by the global maximum value node (A in this scenario) spreads through the network and more nodes become aware of its identity and value. A node becomes the child of a neighbor from which it receives the global maximum value with minimum hop count.

As a result of this algorithm we shall obtain a final tree shown in Figure 3 rooted at node A.

## V. WATCH POINTER

### A. Definition

Due to the several local maxima in the gradient-based distribution there are bound to be some local/secondary peaks in the network: the reading of a node may be greater than that of its parent in the global tree. Hence, we introduce the concept of watch pointers to track such secondary peaks in order to answer queries.

Each watch pointer (WPtr) contains the following information:

- Node : Node ID of the secondary peak.
- Reading : Sensor reading of the secondary peak.
- NextHop : Node ID of the next hop to reach the secondary peak.

The watch pointers are created following the initiation of watch messages from the secondary peaks. These watch messages are forwarded upwards on the tree towards the root until the closest higher ancestor is reached i.e., a node whose reading is greater than or equal to that of the secondary

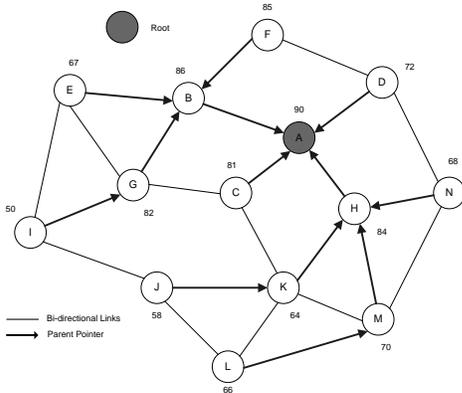


Fig. 3: Resultant Tree

peak. Each node that encounters this message sets up a watch pointer, towards the child from which the watch message was received, for the initiating secondary peak. We can visualize the secondary peak to be a logical child of this closest higher ancestor. The path from the initiating node to the closest higher ancestor is termed as the *Watch Path* for that secondary peak.

With changes in local readings or readings of surrounding nodes a node which was a secondary peak may cease to be a secondary peak. The message type that deletes any stale watch pointers in the network is referred to as the *forget* message. Once a node receives a forget message for a certain secondary peak, it checks whether it has allocated a watch pointer for that peak. If so, it deletes that entry and forwards the *forget* message to its parent in the tree. The message flow stops when it encounters a node that has no history of this watch pointer.

### B. Initiation

Every sensor node in the field periodically senses the environment and updates its own value. In the event of a major change in its own value it initiates the *WP Initiation Algorithm*. The first step involves exchange of update messages between the node experiencing the change and its one-hop neighborhood. The condition for being a secondary peak is that the node's value should exceed that of its parent and all its children on the tree. In that case it considers itself as a secondary peak and initiates a watch message to its parent. The message consists of its node-id, value and watch path (which is initially set to the secondary peak node-id itself).

The only condition to be checked for any considerable change in a nodes reading is whether it becomes or no longer is the closest higher ancestor of some secondary peak. When a node, say  $z$ , notices that its reading has exceeded the value of  $WPtr[x].Reading$  it implies that  $z$  which was on the watch path for the secondary peak but not the closest higher ancestor, has become the closest higher ancestor of node  $x$ . Hence it initiates a forget message on behalf of  $x$  and sends it upwards on the tree so that all the stale watch pointers from  $z$  to the previous closest higher ancestor of  $x$  are de-allocated. Similarly if  $z$ 's reading falls below  $WPtr[x].Reading$ ,  $z$  no longer remains

the closest higher ancestor of  $x$ . Thus  $z$  initiates a watch message towards the root on behalf of  $x$  in search of the closest higher ancestor for the secondary peak at  $x$ . These initiations of watch or forget messages are not performed by the secondary peaks, rather they are triggered due to local changes in nodes readings which are on the watch path.

### C. Processing Watch & Forget Messages

For every watch message received for secondary peak  $y$ , a node  $x$  allocates a watch pointer with Node set to  $y$ , Reading set to  $SR_y$  and NextHop set to the immediate sender of the message. The watch message flows upwards on the tree until the closest higher ancestor is reached. The node  $x$  also appends its node-id at the end of the watch path contained in the watch message before forwarding it to its parent. Note that the closest higher ancestor for some peaks could be the root of the tree. Thus a stream of watch pointers is assigned from closest higher ancestor to the secondary peak for every secondary peak in the system.

### D. Illustration

Consider the network topology and resultant global tree structure shown in Figures 2 and 3. Notice that in the global tree structure shown in Figure 3 every node in the tree had a value greater than all its descendents. Also, there were no secondary peaks and, consequently, no allocation of watch pointers.

Now let us consider the following changes in distribution: readings of nodes G, J, and M rose from 82, 58, and 70 to 88, 86 and, 88, respectively. These new reading would makes these three nodes have greater values than their parents and all their children, thus making them secondary peaks in the system. The watch messages initiated by these nodes travel all the way up the tree to the root node A and watch pointers are allocated on the watch path for all these secondary peaks.

Figure 4 represents the state of the system for the shown readings of the sensor nodes. For any further changes in relative readings, either watch or forget messages are initiated in the system and watch pointers are changed accordingly. Since the WP Initiation and WP Handler algorithms are executing at each node, the watch pointer allocation and de-allocation occurs in a completely distributed fashion without the need for any coordination between the sensor nodes. The threshold that triggers any initiation owing to change in local reading is not determined by the algorithm. This could be a user-defined parameter depending on how frequently one requires the set-up of watch pointers to happen in the network and the error in the query algorithm is bounded by this threshold value.

## VI. ROOT REVERSAL

### A. Definition

Let us consider a change in the identity of the global maximum-valued node. In such a case we need to re-structure the global tree data structure so that the new global maximum is the root of the tree. Along with the change in the root node

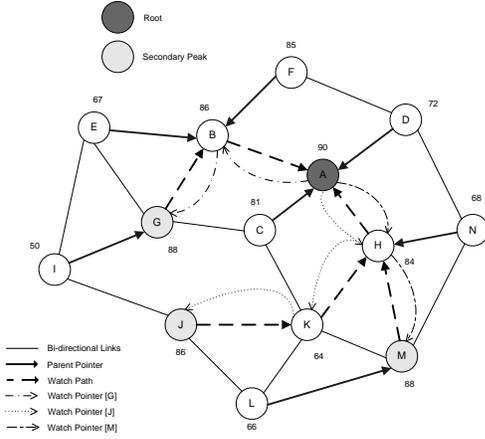


Fig. 4: Tree with Secondary Peaks

identity, we would need to change the parent, child and watch pointers of a few nodes in the network.

The Root Reversal algorithm handles the problem of change in the identity of the global maximum node. This algorithm has two components: RR Initiation and RR Handler. The old root of the tree is the initiator of the algorithm as described in the next subsection. All the changes required in the parent, child and watch pointers are centered on the path between the old root and the new root in the system, termed as the reverse path. This reverse path is derived from the watch message that flows from the new root (which is just a secondary peak before the execution of root reversal algorithm) to the old root of the tree.

The following changes are involved in the root reversal algorithm along the reverse path:

- Parent Pointers : The direction of the parent pointers of nodes lying on the reverse path needs to be reversed. Since each node points towards the root of the system, we just replace the old parent pointer value with the immediate neighbor of a node on the reverse path.
- Child Pointers : While the parent pointers are being reversed we also update the children pointers along the reverse path.
- Watch Pointers : The watch pointers on the reverse path point from the old root towards the secondary peaks. However with the change in the identity of the root node these pointers now become incorrect as they are pointing in the wrong direction. Thus the watch pointers need to be revised along the reverse path, too, so that they are pointing from the new root node towards the secondary peaks.

### B. Illustration

We consider the same example as before in Figure 4 with all the watch pointers allocations for the secondary peaks. Without loss of generality let us suppose that the reading of node  $M$  rises from 88 to 96, which exceeds that of the root node  $A$ . When the watch message initiated from node

$M$  reaches the root, node  $A$  realizes that it is no longer the global maximum node in the network and thus executes the RR Initiation Algorithm.

Node  $A$  copies the watch path from the watch message as the reverse path for this algorithm. It prepares a reverse message which contains the new root node-id ( $M$ ), the reading of the new root node, the reverse path and two separate lists which handle the allocation or de-allocation of the watch pointers along the path. These lists are named as the Assign List (containing the node-id list for which watch pointer should be allocated) and the Remove List (containing the node-id list for which watch pointer should be removed).

Node  $A$  sets its parent pointer to the immediate sender of the watch message, which happens to be node  $H$ . Also the watch pointers at node  $A$  whose next hops lie on the reverse path happen to be  $J$  and  $M$  and those entries are added into the Assign List. The set of local watch pointers, assign list entries and remove list entries at each node on the reverse path is shown in Figure 5.

The next node on the reverse path is node  $H$  and that executes the RR Handler Algorithm upon the reception of the reverse message from node  $A$ .  $H$  allocates a watch pointer for the secondary peak  $A$  and lets that entry in the assign list as it is not the closest higher ancestor of node  $A$ . For the entries in the remove list it performs the origin check. For the entry  $M$ ,  $H$  does not happen to be the origin as the node ids are different and also the next hop from  $H$  to reach  $M$  is indeed  $M$  itself which lies on the reverse path. So it de-allocates the watch pointer entry for  $M$  locally and does not remove that entry from the remove list.

However for the entry  $J$  in the remove list, the next hop to reach  $J$  from  $H$  is  $K$  which is not on the reverse path from  $A$  to  $M$ . Hence  $H$  happens to be the proxy origin for  $J$  in this case. So, instead of deleting the watch pointer, it is moved from the remove list to the assign list. Similarly as the reverse message reaches its final destination  $M$ , the node allocates local watch pointers for the entries in assign list, i.e.  $A$  and  $J$ . The only entry remaining in the remove list is its own id  $M$ , which should ideally be moved to the assign list and it is the origin for that watch pointer allocation. However that becomes immaterial because  $M$  is the new root node and it terminates the RR Handler Algorithm after setting its parent pointer to null value.

Following the execution of the root reversal algorithm, the resultant tree is shown in Figure 6. Notice that the watch path for secondary peak  $G$  stays the same because no portion of it was lying on the reverse path. The same does not hold true for the secondary peak at  $J$ , and thus a part of the path is changed from  $H \rightarrow A$  to  $H \rightarrow M$  because of the change in the position of root node. We also set up the watch pointers for the old root node  $A$  (now just another secondary peak) and erase the watch pointers for the new root node  $M$  in a single pass along the shortest path from  $A$  to  $M$ .

NODE #	BEFORE			AFTER		
	WPtr	ASSIGN LIST	REMOVE LIST	WPtr	ASSIGN LIST	REMOVE LIST
A	[G,J,M]	$\perp$	$\perp$	[G]	[A]	[J,M]
H	[J,M]	[A]	[J,M]	[J,A]	[A,J]	[M]
M	$\perp$	[A,J]	[M]	[A,J]	$\perp$	$\perp$

Fig. 5: Execution Trace of Root Reversal Algorithm

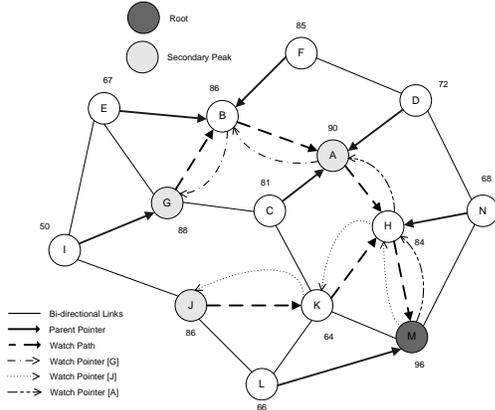


Fig. 6: Final Resultant Tree

### C. Optimization

We observe that this change in the global tree structure following the execution of the root reversal algorithm could result in longer paths from some of the nodes in the network to the new root node. All the nodes not lying on the reverse path do not undergo any changes in their structure, and hence are always on the shortest path to the old root. After all the changes those nodes still pass through the old root to reach the new root, even though there might be shorter paths from those nodes to the new root node. For example in figure 6, the path from node  $G$  to root node  $M$  passes through the old root node  $A$  and consists of four hops. However there exists a shorter path from  $G$  to  $M$  through  $C$  and  $K$  consisting of only three hops. Even with the longer paths all the previously mentioned properties of the global tree structure are satisfied and one does not encounter any error during the query algorithm.

Running the tree formation algorithm in the case of a root node identity change can solve this problem. But that requires a flood by every local peak in the network, resulting in significant communication overhead. There is a tradeoff between running the root reversal versus the tree formation algorithm. The former generates fewer messages and takes less time to converge. However, the user can periodically run the tree formation algorithm to optimize all the paths and data structures in the sensor network.

## VII. QUERY

Most of the earlier aggregation methods either target simple queries such as max, min, sum, average or tackle more complex queries such as the  $k^{th}$  highest or lowest value,

median, top/bottom  $k$  values. We aim to solve a combination of these queries using the same tree structure and the various pointers maintained at each node. Since we have presented our entire approach considering the maximum or top  $k$  values, i.e., the interest is in retrieving the readings in descending order; we shall continue the same trend for the query algorithm as well.

The querying agent can easily determine the identity of the root node by enquiring any node in the network as all the nodes are aware of the global maximum valued node. For every change in the identity of the root node, the new root node informs all the nodes in the network about this change within a finite amount of time. This is extremely important for the functioning of our query algorithm because we run this algorithm on the querying agent which at first contacts the root node and subsequently contacts other nodes in the network depending on the output of the algorithm, described in detail in the next subsection. For contacting a particular node in the network we may choose to utilize the routing protocol to guarantee the shortest path to any node.

### A. Algorithm

To understand how querying proceeds let us visualize the gradient-based distribution of the sensor network as a mountainous region submerged entirely in water. As we slowly drain out the water, the highest peak (which happens to be the root of our tree structure) would emerge first. Thereafter, the nodes present on the sides of the peak would emerge from the water, which is nothing but the descendants of the root node on the tree. Subsequently, if there are some secondary peaks they shall become visible as more and more water drains out. We already have stored the information about these secondary peaks and their locations through the watch pointers, so including these logical children through watch pointers we can include the secondary peaks into our consideration. This is sort of a distributed max heap traversal where we are traversing down multiple slopes at the same time and outputting the nodes in descending order of their readings. This forms the crux of our algorithm and we terminate the algorithm as all the water is drained out and all the nodes become visible.

We maintain a max heap data structure at the querying agent and initialize it with the root node. Please note that we have taken some liberties while using the term *heap* in this discussion. This is not a traditional heap structure; rather we use it to illustrate the traversal of nodes in a fashion which is similar to a max heap. Every Delete Max operation on the

OP #	MAX HEAP	OUTPUT	READING
1	[ <b>M</b> ]	M	96
2	[ <b>L,H,A,J</b> ]	A	90
3	[L, H, <b>B,C,D,G</b> , J]	G	88
4	[L, H, B, C, D, <b>I</b> , J]	B	86
5	[L, H, <b>E,F</b> , C, D, I, J]	J	86
6	[L, H, E, F, C, D, I, <b>K</b> ]	F	85
7	[L, H, E, C, D, I, <b>K</b> ]	H	84
8	[L, <b>N</b> , E, C, D, I, <b>K</b> ]	C	81
9	[L, N, E, D, I, <b>K</b> ]	D	72
10	[L, N, E, I, <b>K</b> ]	N	68
11	[L, E, I, <b>K</b> ]	E	67
12	[L, I, <b>K</b> ]	L	66
13	[I, <b>K</b> ]	K	64
14	[I]	I	50

Fig. 7: Execution of Query Algorithm

heap returns the node id along with its reading in decreasing order.

The querying agent only contacts a particular sensor node in the network when its node-id is output as a result of the delete max operation on the heap. Remember that the agent can use the routing protocol to guarantee the shortest path to reach the nodes and the algorithm is not dependent on that aspect. The information required from every node is its watch pointers (identities of all its logical children), and its child pointers. In the case of a secondary peak since we want the nodes in descending order, we need to traverse on both sides of the secondary peak. So, we include its parent pointer along with all its children, neighboring ones on the tree actual as well as logical ones. We repeat the same set of steps until all the nodes in the networks are marked as visited.

### B. Illustration

We run the query algorithm on the example network shown in Figure 6 and explain how the nodes are being outputted in the right order. The secondary peaks are at nodes *A*, *G* and *J*, with node *M* being the root of the tree.

We start the execution of the algorithm at the root node *M*. Since *M* is the only node present in the heap, the delete max operation outputs *M* as the node with the highest value. Then the querying agent contacts node *M* for its latest reading (96). Node *M* has watch pointer allocated for *A* and *J*, which happen to be its logical children. Hence, nodes *A* and *J* are added to the heap. Also *M* has children pointers to neighboring nodes *H* and *L*. So they are also added to the heap.

The delete max operation outputs node *A*. From *A* we follow the child pointers to reach nodes *B*, *C*, and *D*, and follow the watch pointer to reach *G*. We notice that node *G* happens to be the third highest value in the network and our algorithm outputs on the next execution of delete max. We repeat the same operation until all the nodes are visited and the execution trace for the example is shown in Figure 7. It consists of the contents of the heap, modifications to it at every

step and which node is output after each round of execution. Also, the bold letters represent the nodes inserted into the heap at the beginning of each round.

## VIII. ANALYSIS

In this section we analyze the performance of our algorithms in terms of number of messages exchanged and the time taken to run to completion.

### A. Theoretical Evaluation

Let the maximum degree of a vertex in the graph representing the network topology be  $u$  and let the diameter of the network be  $d$ . In the worst case, the tree formation algorithm would require every node to transmit  $u$  messages in each round (one for each neighbor). The number of rounds required for the tree to be completely formed would be directly proportional to the diameter of the network; hence it would take  $O(d)$  rounds to build the global data structure. The total number of messages to be transmitted by each node during the tree formation phase would be  $O(u \times d)$  in the worst case.

The number of messages transmitted during the watch pointer and root reversal algorithm depend entirely on the distribution of the nodes and the rate of change in their local measurements. The user always has the option of choosing an optimum threshold for triggering these changes to maintain the balance between accuracy and communication overhead. For the query algorithm we realize that in order to output all the nodes in the network we visit each node only once. Hence we spend constant time at each node and thus the time complexity for running the query algorithm completely in the network of  $n$  nodes is  $O(n)$ . However the running time for the max query is  $O(d)$  as we need to traverse the entire diameter of the network to reach the maximum valued root node. Similarly to retrieve the top  $k$  values or the  $k$ th highest value we need to visit only  $k$  nodes, thus the time taken is  $O(k \cdot d)$ . However, this worst case scenario is uncommon for natural phenomena distributions wherein we have the information hotspots relatively close to each other in a bounded area.

The key advantage of our scheme, as compared to other approaches, is that the global data structure helps us answer a variety of queries in a reasonable amount of time. Ranging for simple queries such as max, average and sum to more complex quantile ones such as median, top  $k$  values or the bottom  $k^{th}$  value we can answer these queries using the same tree structure and pointer allocations with minor changes to the query algorithm running at the base station. This is useful in the case of already deployed networks where it is difficult to change the implementation running on the sensor nodes to perform in-network aggregation for a variety of queries. This aggregation scheme also shifts the burden of running the query algorithm to the sink, thus saving battery power and computational resources of the sensor nodes.

Almost all the messages (apart from those exchanged in root reversal) are short in size and can easily be piggybacked on the heartbeat messages between neighboring nodes. This could drastically reduce the communication overhead incurred

during the maintenance of the tree structure. Messages transmitted after the tree formation phase arise due to changes in local measurements.

## IX. CONCLUSION

We have presented a distributed data aggregation scheme for sensor networks which utilizes the information gradients present in the network. Our approach uses these naturally available information gradients to build a tree structure that is maintained at each sensor node through minimal messages exchanged between one-hop neighbors. We also provide algorithms that handle the local changes in nodes readings and preserve the desired properties of the tree structure. The query algorithm eliminates any in-network aggregation and is extremely versatile in terms of supported queries through traversal of the tree structure. We have also shown how these algorithms function in a completely distributed fashion without any central authority or synchronization between the sensor nodes.

In the current work, we have assumed that the rate of change in the nodes measurements is comparatively slower than the convergence time of our algorithms. We also have not taken into account the effect of node and communication links failure in our study. This can be solved by running the tree formation algorithm and setting up the entire structure in the presence of failures in the system. The refreshed structure would reflect the current state of the system and thus yield the appropriate results. Since our approach is proactive, we incur some additional overhead while setting up the pointers and information gradients. However, once the tree structure is stabilized multiple queries including different types can be quickly answered without any messages being transmitted by the sensor nodes and any in-network computation being performed.

As presented in this paper, the aggregation scheme only provides the identity of the sensor nodes for the queries. This work can be easily extended to preserve the spatial information of the nodes which would facilitate the user to know the exact locations of the information sources. This could be extremely useful for tracking and navigation applications. The mobility of the sensor nodes inside the network is also not considered in this paper; hence one can extend this scheme to handle different mobility patterns in sensor networks. With the movement of nodes the information gradients would change resulting in a number of existing links being broken and newer links being formed. This would require further changes in the global structure to reflect the current state of the system.

## REFERENCES

- [1] D. R. Askeland. The science and engineering of materials. 1994.
- [2] Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, 1985.
- [3] Haowen Chan, Adrian Perrig, Bartosz Przydatek, and Dawn Song. Sia: Secure information aggregation in sensor networks. *J. Comput. Secur.*, 15(1):69–102, 2007.
- [4] Jeffrey Considine, Feifei Li, George Kollios, and John Byers. Approximate aggregation techniques for sensor databases. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 449, Washington, DC, USA, 2004. IEEE Computer Society.
- [5] Amol Deshpande, Carlos Guestrin, Samuel R. Madden, Joseph M. Hellerstein, and Wei Hong. Model-driven data acquisition in sensor networks. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 588–599. VLDB Endowment, 2004.
- [6] Jaded Faruque. Rugged: Routing on fingerprint gradients in sensor networks. In *in Proceedings of the IEEE International Conference on Pervasive Services (ICPS), Novi Sad, Serbia and*, 2004.
- [7] Johannes Gehrke and Samuel Madden. Query processing in sensor networks. *IEEE Pervasive Computing*, 3(1):46–55, 2004.
- [8] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 58–66, New York, NY, USA, 2001. ACM.
- [9] Michael B. Greenwald and Sanjeev Khanna. Power-conserving computation of order-statistics over sensor networks. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 275–285, New York, NY, USA, 2004. ACM.
- [10] Joseph M. Hellerstein, Wei Hong, Samuel Madden, and Kyle Stanek. Beyond average: Towards sophisticated sensing with queries. In *In IPSN*, pages 63–79, 2003.
- [11] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 56–67, New York, NY, USA, 2000. ACM.
- [12] Huijia Lin, Maohua Lu, Nikola Milosavljevic, Jie Gao, and Leonidas J. Guibas. Composable information gradients in wireless sensor networks. In *IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks*, pages 121–132, Washington, DC, USA, 2008. IEEE Computer Society.
- [13] Juan Liu, Feng Zhao, and Dragan Petrovic. Information-directed routing in ad hoc sensor networks. In *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 88–97, New York, NY, USA, 2003. ACM.
- [14] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 131–146, New York, NY, USA, 2002. ACM.
- [15] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 491–502, New York, NY, USA, 2003. ACM.
- [16] Samuel Madden, Robert Szewczyk, Michael J. Franklin, and David Culler. Supporting aggregate queries over ad-hoc wireless sensor networks. In *WMCSA '02: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, page 49, Washington, DC, USA, 2002. IEEE Computer Society.
- [17] Amit Manjhi, Suman Nath, and Phillip B. Gibbons. Tributaries and deltas: efficient and robust aggregation in sensor network streams. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 287–298, New York, NY, USA, 2005. ACM.
- [18] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 346–357. VLDB Endowment, 2002.
- [19] Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary Anderson. Synopsis diffusion for robust aggregation in sensor networks. *ACM Trans. Sen. Netw.*, 4(2):1–40, 2008.
- [20] David Peleg. Time-optimal leader election in general networks. *J. Parallel Distrib. Comput.*, 8(1):96–99, 1990.
- [21] Narayanan Sadagopan, Bhaskar Krishnamachari, and Ahmed Helmy. Active query forwarding in sensor networks. 3:91–113, 2005.
- [22] J. F. Shackelford. Intro to materials science for engineers. 2000.
- [23] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and beyond: new aggregation techniques for sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 239–249, New York, NY, USA, 2004. ACM.