Gradient-based Aggregation in Forest of Sensors (GrAFS)

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

***Abstract*—In several sensing applications the parameter being sensed exhibits a high spatial correlation. For example, if the temperature of a region is being monitored, there are distinct hot and cold spots. The area close to the hot spots is usually warmer than average, and there is a temperature gradient between the hot and cold spots. We exploit this correlation of sensor data to form a forest of logical trees in the sensor field, with the trees collectively spanning all the sensor nodes. The root of a tree corresponds to a sensor reporting the local peak value. The tree nodes represent the value gradient: each node's sensed value is smaller than that of its parent, and greater than that of its children. GrAFS provides a mechanism to maintain some information at the local peaks and share it with the sink of the sensor network. Using this information the sink can answer several queries. For queries that the sink cannot answer using locally available information, the sink knows the region of the sensor field that it needs to probe for the answer. Thus, queries can be answered in a time and/or bandwidth efficient manner. The GrAFS approach to data aggregation can easily adapt to changes in the spatial distribution of sensed values, and also cope with message losses and sensor node failures. Simulation experiments conducted using ns-3 quantify the performance of GrAFS.**

I. Introduction

Data aggregation has been extensively studied in the context of wireless sensor networks [1], [2], [3], [4], [5], [6], [7]. Stated simply, data from multiple sensors are aggregated to generate intermediate results of size much smaller than the combined sizes of all the aggregated samples. Data aggregation reduces network bandwidth usage. A reduction in the number of bits transmitted or received reduces energy consumption of nodes that typically have low energy reserves.

A simple approach to aggregation is to construct a spanning tree rooted at the sink. Each sensor node sends to its parent the aggregate of its own reading and the values obtained from the children. This approach is especially suitable for answering queries like maximum or minimum temperature in the sensor field. Regardless of the number of children a node has, the size of the aggregated data is always a constant.

However, the aggregation approach mentioned above provides the sink no insight into the distribution of values in the sensor field. Hence, it may be difficult to answer queries like the x^{th} percentile value in the sensor field. The amount of information made available to the sink has been minimized to maximize network efficiency.

The other extreme is to perform no aggregation at all. The sink receives readings from all the sensor nodes and can answer a variety of sophisticated queries. However, the communication overheads are proportional to the size of the network, leading to obvious scalability problems.

Several researchers have proposed novel approaches to aggregation (briefly described in Section V) that strike a judicious compromise between information communicated to the sink and bandwidth consumption. GrAFS, too, strives to achieve such a balance. If some queries cannot be answered by the sink based on local information, the sink can initiate a targeted search in the sensor network for relevant data in a speedy and bandwidth- and energy-efficient manner.

The aggregation approach described in this paper is especially useful if the sensed values have some spatial correlation. Several sensing scenarios fit the bill. For example, the temperature or the level of various gases and environmental pollutants in a region. If a sensor is reporting a very high temperature at time t, with high likelihood the neighboring sensors are also going to report high temperatures at or around time t.

GrAFS forms and maintains a forest of gradient-based trees. The roots of these trees are sensors that report the local peaks of the parameter being monitored. Within a tree, the path from the root to any node consists of nodes with monotonically decreasing values of the parameter being monitored.

In the following sections we describe our solution, prove its correctness, discuss its tolerance to message losses and node failures, and describe how it adapts to changes in the values reported by various sensor nodes. We also present results of simulation experiments that measure the performance and the overheads incurred by the solution.

II. System Model

We assume a sensor field covered by N sensor nodes, where N is not known to individual nodes. Each sensor node is equipped with a sensor, a wireless transceiver and a local clock. The local clock is used to establish a temporal order among events at the node. The wireless transceiver at a node has a limited communication range. So, a sensor node S_i can directly communicate (send and receive data) with a subset, Nbr_i, of sensor nodes referred to as the neighbors of S_i. S_i knows the membership of Nbr_i. There is a known *sink* node.

The underlying routing protocol can find a path from each sensor node to the sink.

We make no assumptions about the channel access protocol employed by the sensor nodes. Also, the range of values reported by the sensor nodes is not known in advance.

III. Basic Idea

Definition 1. *A sensor node is referred to as a local peak if its temperature is greater than or equal to the temperature of all its neighbors.*

Every message sent by a sensor node is timestamped with the value of its local clock. A node receiving multiple messages from the same node can determine the order in which the messages were sent.

A. Tree Construction

Solely for the purpose of illustration, we assume that the sensor network is being used to monitor the temperature of the sensor field and that each node has a temperature sensor. The proposed solution could also be used to monitor any phenomenon where the sensor values can be totally ordered, like humidity, illumination, atmospheric pressure, water salinity, chemical concentration, etc.

An underlying neighbor discovery protocol enables each node to determine the identity and temperature of its neighbors. Using this information a node can determine if it is a local peak. For each local peak a gradient-based tree rooted at the local peak is initially created, and subsequently maintained. In a gradient-based tree the temperature of a node is greater than or equal to the temperature of its children. If a parent-child relation exists between two nodes they must be neighbors of each other. As described in Section IV, each node, S_i, in a gradient-based tree knows the following:

1) the identity of the local peak whose tree S_i belongs to,
2) temperature of S_i's local peak,
3) S_i's parent in the tree,
4) S_i's children in the tree,
5) minimum temperature in the subtree rooted at S_i (temperature of a leaf node), and
6) number of nodes in the subtree rooted at S_i.

Thus, a local peak knows the maximum temperature (own temperature), the minimum temperature and the total number of nodes in its gradient-based tree.

To construct a gradient-based tree a local peak lp_i initiates a flood message. A node *inherits* a flood message received from a neighbor only if its temperature is less than or equal to the temperature of that neighbor and *the node is not already part of a gradient-based tree*. On inheriting the flood message the node becomes a child of the neighbor that sent the message, and propagates the flood message to its other neighbors. Each node in the sensor field belongs to exactly one gradient-based tree.

B. Queries and Responses

Each local peak, lp_i, sends the following four-tuple to the sink: $< lp_i,\ high_i,\ low_i,\ num_i >$, where $high_i$, low_i and num_i are the maximum and minimum temperatures (henceforth, also referred to as the temperature range) and the number of nodes in lp_i's gradient-based tree, respectively. The local peak also registers itself with the sink. The sink stores the four-tuple reported by each local peak, lp_i, as a structure t_i.

A query about the maximum temperature in the sensor field can be immediately answered by the sink. It is the maximum of the *high* temperatures reported by the local peaks. The minimum temperature in the field is the minimum of the *low* temperatures reported by the local peaks. It is also possible to answer more complex queries, like the median temperature in the sensor field, at a cost that increases with the accuracy of the response.[1]

Let the query be to find the sensor with temperature that corresponds to the x-percentile in the sensor field. As described later, in Section IV, sink inspects the four-tuples in descending order of their low values. It determine the four-tuple, t_i, with corresponding low_i value such that at least $100 - x\%$ of the nodes have temperature greater than or equal to low_i. So, the $x - percentile$ value must be greater than or equal to low_i. Then, the sink inspects all the four-tuples, this time in ascending order of their $high$ values, to determine the four-tuple, t_j, with corresponding $high_j$ value such that at least $x\%$ of the nodes have temperature less than or equal to $high_j$. So, the $x - percentile$ value must be in the range $[low_i, high_j]$.

Now, as described in Section IV, the sink can limit its search for the $x - percentile$ value to only those gradient-based trees whose high-low range overlaps with the inverval $[low_i, high_j]$.

The gradient-based trees lend themselves to efficiently answering the following query: list all sensor nodes that have a reading in the range $[range\_low, range\_high]$. The sink can limit its search to the gradient-based trees whose temperature range overlaps with the queried range. Similarly, within each queried gradient-based tree, the search can be limited to only those subtrees whose temperature range overlaps with the queried range.

C. Reacting to Temperature Changes

If changes in temperature recorded by individual nodes do not change the temperature gradient, no tree management action needs to be performed other than the local peaks periodically sending their updated four-tuples to the sink. Sometimes a node may cease to be a local peak. In such a situation, zero or more new local peaks get formed. In GrAFS, when a node ceases to be a local peak it informs the sink about the disappearance of the tree rooted at the node. Also, all the nodes that were members of the departed tree become

[1]The cost of answering a query is expressed in terms of latency and the amount of communication between the sink and sensor nodes.

members either of new gradient-based trees or of other existing trees that grow at the expense of the departing tree.

At times a node that was part of a gradient-based tree t_i, but not a local peak, may become a new local peak while t_i's peak is still a local peak. The newly formed peak initiates the formation of a new gradient-based tree rooted at itself. All the newly formed gradient-based trees, as well as the trees that experience a change in membership communicate their new four-tuple information to the sink.[2]

D. Failure and Recovery

Link failures and node failures can be handled in similar ways because they both result in nodes being unable to communicate with each other. If a node is unable to communicate with a child in its gradient-based tree, the node recomputes its four-tuple to reflect the departure of subtree rooted at the unreachable child. Similarly, if a node is unable to communicate with its parent, the node either becomes a local peak and creates its own gradient-based tree, or joins an existing gradient-based tree. Similarly, when a node (re)joins the network and finds its value to be greater than all its neighbors it becomes a local peak and initiates the formation of its gradient-based tree.

IV. Solution Description

In this section we describe how gradient-based trees are constructed, how queries like finding the $x-percentile$ value are answered, and how the forest of gradient-based trees is adjusted following temperature changes in the sensor field.

A. Formation of Gradient-Based Trees

A local peak lp_i initiates the formation of a gradient-based tree by sending a JOIN($i,\ i,\ temp_i$) message to all its neighbors. The first parameter in the JOIN message is the identity of the local peak that initiated the diffusion of the JOIN message, while the second and third parameters are the identity and temperature of the node transmitting the message. A node y inherits a JOIN($i,\ x,\ temp_x$) message, received from a neighbor x, only if $temp_y \leq temp_x$ and if y is not already part of another gradient-based tree. If y inherits the JOIN($i,\ x,\ temp_x$) message, y becomes x's child and transmits the JOIN($i,\ y,\ temp_y$) message to all its neighbors. Node y always informs node x if it inherited x's JOIN() message or not. Thus, each node x in a gradient-based tree gets to know the identity of its parent and all its children (represented by the set $child_x$) within a finite time.

Having propagated the JOIN($i,\ x,\ temp_x$) message, node x waits to receive a four-tuple of the form $< node\_id,\ temp_y,\ temp\_low_y,\ number >$ from each child node y.[3] A leaf node z, with temperature $temp_z$ sends the following four tuple to its parent: $< z,\ temp_z,\ temp_z,\ 1 >$. Once node x has received the four-tuples from all its children its generates a four tuple of the form $< x,\ temp_x,\ temp\_low_x,\ num_x >$ where $temp\_low_x$ and num_x are computed as follows:

$$temp\_low_x = \min_{y \in child_x} (temp\_low_y)$$

$$num_x = 1 + \sum_{y \in child_x} (num_y)$$

Thus, $temp_x$ and $temp\_low_x$ represent the highest and the lowest temperatures, respectively, recorded by nodes in the gradient-based sub-tree rooted at node x. The total number of nodes in the sub-tree rooted at x is equal to num_x. Node x sends the four tuple to its parent. A local peak communicates the four-tuple it computes to the sink.

B. Query Processing

Having received the four-tuples from the local peaks, the sink adds the num fields in these tuples to determine the total number of sensor nodes. The sink manages all the four-tuples of the type t_i using the following two heaps:

- Descending_low_heap: a max-heap consisting of all the four-tuples reported to the sink where the *low* field of a four-tuple is used to determine its position in the heap. So, the root of the heap corresponds to the four-tuple with the highest *low* value.
- Ascending_high_heap: a min-heap consisting of all the four-tuples reported to the sink where the *high* field of a four-tuple is used to determine its position in the heap. So, the root of the heap corresponds to the four-tuple with the lowest *high* value.

1) Percentile Queries: The sink first executes Algorithm 1. This algorithm scans the four-tuples for the gradient-based trees in descending order of their low value. We explain the execution of Algorithm 1 with the help of an example shown in Figure 1. The goal is to find the $70 - percentile$ value. Each horizontal line in the figure represents a gradient-based tree. The name of the tree appears above the line and number of nodes in the tree is shown in parentheses after the name. The numbers on the left and right of the line are the low and $high$ values (in the unit of choice) of the tree, respectively. There are a total of nine gradient-based trees and one hundred nodes.

Algorithm 1 first processes the four-tuple for tree T1 which has ten elements because T1 has the highest *low* value among all trees. Thus, the sink knows there are at least ten sensor nodes with temperature at least 55 units. Then, it processes the four-tuple for tree T2 because it has the next highest *low* value of 45 units. As T2 has fifteen nodes, the sink concludes that at least twenty-five $(10 + 15)$ sensor nodes out of one hundred have temperature greater than or equal to 45 units.

[2] If two neighboring nodes are reporting temperatures that are close to each other, small changes in their temperature and/or minor inaccuracy of the temperature sensors may trigger rapid creation or disappearance of local peak(s). This may trigger corresponding tree formation/modification messages. Such *gradient-tree flapping* can be damped by using appropriate thresholds on the temperature differences required to trigger tree formation or tree modification.

[3] The way gradient-based trees are constructed, the maximum temperature of a sub-tree rooted at node y will be equal to $temp_y$.

Then, the four tuple corresponding to tree T3 is processed. It has ten nodes with the *low* value equal to 43 units. This means at least thirty-five out of one hundred sensor nodes have temperature greater than or equal to 43 units. Hence, the $70 - percentile$ value must be at least 43 units.

Next, the sink executes Algorithm 2 to determine a value $high\_end$ which is the upper-bound on the $x - percentile$ value. Its operation is described in the context of the same set of nine gradient-based trees mentioned above, this time arranged in ascending order of their high values, as shown in Figure 2. The first four-tuple processed by the sink corresponds to tree T9 with seven nodes and the *high* temperature of 25 degrees. This means at least seven nodes have temperature no more than 25 degrees. The sink then processes the four-tuple for tree T7 with the second-lowest $high$ value of 34 and consisting of ten nodes. Thus, at least seventeen $(7 + 10)$ sensor nodes have temperature no more than 34 units. The sink continues processing the four-tuples until it can determine that at least seventy out of the one hundred nodes have temperature no more than a certain value. This happens after the sink has processed the four-tuples for trees T8, T4, T3, T5, and T6 as well (in that order). At this point the sink knows that at least seventy-five nodes have temperature no more than 53 units.

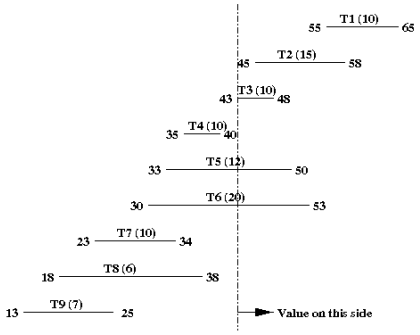

Fig. 1. Four-tuples arranged in descending order of low value.

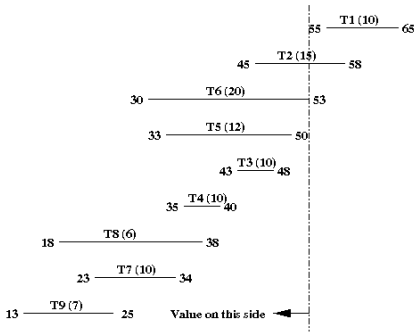

Fig. 2. Four-tuples arranged in ascending order of high value.

Let us first consider the simple case, as in the example above, where $low\_end \leq high\_end$.[4] If $low\_end < high\_end$ then the sensor with the x-percentile value is located in a gradient-based trees whose temperature range overlaps with the range $< low\_end,\ high\_end >$. For the example discussed above, the low_end and $high\_end$ are 43 and 53 units respectively. So, the search for $70 - percentile$ value should be limited to trees T2, T3, T5 and T6. This eliminates 43 nodes right away, without any additional communication.

Algorithm 1: **Find_low_end(x)**: finds lower end of x-percentile value

Require: Descending_low_heap
low_count = 0
low_end = maxval
low_tree = NULL;
while low_count $< \frac{100-x}{100}N$ **do**
low_tree = delete_max(Descending_low_heap)
low_count $+=$ low_tree.num;
low_end $+=$ low_tree.low;
end while
return low_end, low_count, low_tree

Algorithm 2: **Find_high_end(x)**: find the higher end of x-percentile value

Require: Ascending_high_heap
high_count = 0
high_end = minval
high_tree = NULL
while high_count $< \frac{x}{100}N$ **do**
high_tree = delete_min(Ascending_max_heap)
high_count $+=$ high_tree.num
high_end $+=$ high_tree.high
end while
return high_end, high_count, high_tree

To reduce the difference between low_end and $high\_end$ and converge towards the precise x-percentile value the sink executes one or both of the following steps:

1) Send a probe to the local peak of low_tree. For the example described above using Figures 1 and 2 the low_tree is T3. The local peak of low_tree sends a four-tuple of the form $<$ $low\_tree.lp$ $low\_tree.lp.temperature$, $low\_tree.lp.temperature$, $1 >$ corresponding to itself, and four-tuples corresponding to the gradient-based trees rooted at each of its children. Then, the four tuple for low_tree is replaced with the four tuples returned by the peak of low_tree in *Descending_min_heap* and Algorithm 1 is executed on the modified *Descending_min_heap* to increase the value of low_end.

[4]If both are equal then their value corresponds to the $x - percentile$ value. Also, in this case Algorithm 2 returns a pointer to a gradient-based tree whose local peak has temperature equal to the $x - percentile$ value.

2) Send a probe to the local peak of $high\_tree$. For the example described above, the $high\_tree$ is T6. The local peak of $high\_tree$ sends a four-tuple of the form $< high\_tree.lp$, $high\_tree.lp.temperature$, $high\_tree.lp.temperature$, $1 >$ corresponding to itself, and a four-tuple corresponding to the gradient-based trees rooted at each of its children. Then, the four tuple for $high\_tree$ is replaced with the four tuples returned by the peak of $high\_tree$ in *Ascending_max_heap*, and Algorithm 2 is executed on the modified $Ascending\_max\_heap$ to decrease the value of $high\_end$.

These two steps progressively reduce the size of the interval [low_end, $high\_end$] while limiting the search to only one gradient-based tree at a time.

Now, let us consider the possibility that $low\_end > high\_end$. If $low\_count = \frac{(100-x)}{100}N$ and $high\_count = \frac{x}{100}N$ and no gradient-based tree has a temperature range that overlaps with the open interval $(high\_end,\ low\_end)$ then $high\_end$ corresponds to the x-percentile value and $high\_tree.lp$ is the sensor node with the x-percentile value. All other cases where $low\_end > high\_end$ cannot occur. We refer to such cases as *infeasible* cases.

Lemma 1. *Algorithms 1 and 2 will never yield infeasible cases.*

Proof: We prove the lemma by contradiction. Let us assume that $low\_count \geq \frac{(100-x)}{100}N \wedge high\_count \geq \frac{x}{100}N$ and $\neg(low\_count = \frac{(100-x)}{100}N \wedge high\_count = \frac{x}{100}N)$.

The set, L, of at least low_count sensor nodes have temperature no less than low_end, and the set, H, of at least $high\_count$ sensor nodes have temperature no greater than $high\_end$. As $low\_end > high\_end$ we can assert that $L \cap H = \emptyset$. As $\neg(low\_count = \frac{(100-x)}{100}N \wedge high\_count = \frac{x}{100}N)$, this implies that $\mid L \cup H \mid > N$ which is impossible. ■

2) Range Queries: For ease of description, we consider the sink to be the parent of all local peaks. Let the query be to list all the sensor nodes that have a reading in the range [$range\_low$, $range\_high$]. The sink forwards this query to only those local peaks whose gradient-based trees' temperature range, [low, $high$], overlaps with [$range\_low$, $range\_high$], and then waits for responses from them. Each node that receives this query first checks if the queried range overlaps with the temperature range of its subtree. If not, the node sends a NULL response to its parent. Otherwise (when there is an overlap), the node forwards the query to only those children that have reported a temperature range that overlaps with the temperature range in the query, and waits for responses from them. A leaf node cannot propagate the query further, and sends a response to its parent. The response contains the attributes of the leaf node (including its identity) and its temperature if the temperature is within the queried range. If the leaf node's recorded temperature does not lie within the queried range, it sends a NULL response. When a node receives responses from all the children to whom it had forwarded the query, it aggregates their responses along with its own response (if its own temperature lies within the queried range) and sends the aggregated response to its parent. Note that at most one NULL response is sent to a parent (even if multiple children sent NULL responses), and that too only if no node in the subtree has temperature within the queried range.

Such range queries can be used to form temperature contour lines and contour maps.

C. Reacting to Temperature Changes

Let us consider a node x that now has a child y with a higher temperature that its own. This may be either due to x cooling down, y warming up, or both. We refer to this occurrence as a *gradient disruption*.

As described earlier, x knows the four-tuple corresponding to the sub-tree rooted at y. Due to the gradient disruption, the sub-tree rooted at y can no longer be part of the gradient-based tree, T_i to which x belongs. Let $< x,\ temp_x,\ temp_{low\_x},\ num_x >$ be the four-tuple corresponding to the sub-tree rooted at x prior to the gradient-disruption. The following actions are performed by the various nodes:

Action 1: Node x performs the following actions:

- **Step 1:** Node x recomputes its four-tuple, not considering the sub-tree rooted at y, and conveys the updated four-tuple to its parent. Each node on the path between x and the root of x's gradient-based tree, including the root, recomputes its four-tuple on receiving the update from its child. Finally, the root of T_i communicates its updated four-tuple to the sink. If node x happens to be the local peak of a gradient-based tree prior to the gradient disruption, the four tuple it sends to the sink is $< x,\ NULL,\ NULL,\ 0 >$ indicating the disappearance of the tree rooted at x.
- **Step 2:** If x was a local peak prior to the gradient disruption, x sends a RELINQUISH($x,\ temp_x$) message to all its children. If x was not a local peak prior to gradient disruption, it only sends the RELINQUISH($local\_peak\_of\_x,\ temp_x$) message to y. Node y on receiving the RELINQUISH($\alpha,\ temp_x$) message from x, where α is the identity of the local peak, leaves the gradient-based tree rooted at α, and propagates the message to its children. As the message propagates through the tree, nodes leave the gradient-based tree rooted at α. They are now temporarily orphaned and need to join another gradient-based tree.

Action 2: When a node of α's tree receives the RELINQUISH message it forwards the message to all its neighbors that belong to other gradient-based trees, too.

Action 3: A node z that has become a local peak diffuses a JOIN message through the network, and the message propagates along the temperature gradient. Nodes that inherit the JOIN message become part of the gradient-based tree rooted at z. Ultimately, z sends the four-tuple corresponding to its gradient-based tree to the sink, thus registering a new tree with the sink.

Action 4: When a node nbr that belongs to an adjacent gradient-based tree receives the RELINQUISH(α, x) message, nbr propagates a JOIN message on behalf of its own local peak with the goal of absorbing those nodes previously in α's tree that are downstream from it with respect to the temperature gradient. Once the absorption is done, the updated four-tuple is sent to the sink by the local peak of the absorbing tree.

Depending on the underlying channel access and routing protocol, it is possible that a node receives a JOIN message initiated by a new local peak before its receives the RELINQUISH message initiated on behalf of its old local peak. *To avoid any confusion such out-of-order delivery of message may cause, the JOIN message can be made to carry information to the effect that it is a causal successor of the RELINQUISH message and should be processed only after the receiver has processed the RELINQUISH message. Also, if for any reason the RELINQUISH message was never received by a node, the subsequent JOIN message carries information about the RELINQUISH message.* So, the node on receiving the JOIN message can join the new tree.

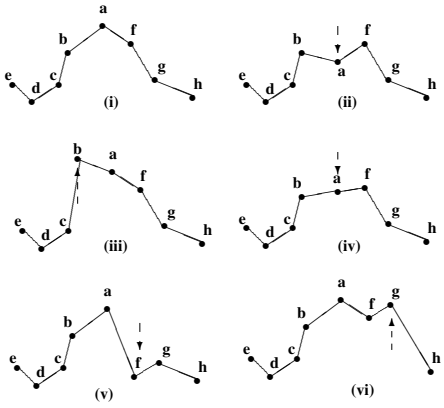

Fig. 3. Reaction to temperature changes

Using Figure 3, we illustrate how the proposed solution reacts to temperature changes. Initially (Figure 3(i)), node a is the local peak and nodes $b - d$ and $f - h$ are members of the gradient-based tree rooted at a. The other five topologies reflect gradient-disruptions with respect to Figure 3(i).

In Figure 3(ii) the temperature of a decreases and it is no longer the local peak. Node a informs the sink about the disappearance of the gradient-based tree for which it was the root. It also propagates the RELINQUISH message to all the other nodes formerly in its tree. After b and f, the new local peaks, have processed and forwarded the RELINQUISH message they initiate their respective JOIN messages. Nodes c and d join the gradient-based tree rooted at b, while g and h join the gradient-based tree rooted at f. Both the new local peaks send their JOIN messages to a which joins one of the trees. So, no node stays orphaned.

In Figure 3(iii) node b's temperature increases to a value greater than a's temperature. As node a was a local peak prior to the gradient disruption, it propagates the RELINQUISH message through its gradient-based tree. Subsequently, node b, the new local peak, initiates the JOIN message which diffuses to *all* other nodes that were previously members of node a's tree. The situation in Figure 3(iv) is similar as the old local peak has cooled down to a temperature lower than that of a neighbor, f, which is the new local peak.

Figures 3(v) and (vi) show the situation where the old local-peak, a, continues to be a local peak. Due to gradient inversion, either due to the cooling of f (part v) or the heating of g (part vi), node g becomes a new local peak. As the gradient inversion involves nodes f and g, node f informs a about the change in its tree composition. Node f also propagates the RELINQUISH message down the subtree rooted at g. Consequently, nodes g and h leave a's tree. Also, node g initiates the JOIN message which makes node h a member of g's tree, but does not propagate beyond node f on the left hand side of the figure.

Now, let us consider the situation depicted in Figure 3(vi) as the initial condition. The local peak g cools down to a temperature below that of node f and the new configuration is shown in Figure 3(i). Here we have a local peak disappearing, with no new local peak being formed to replace it. As described above, the RELINQUISH message generated by the departing local peak g reaches f and h. If f was not part of g's gradient-based tree ($i.e.$, it belonged to a's tree), then as described in Action 4 of the solution, f propagates the JOIN message on behalf of its root a and node g and h get absorbed into the gradient-based tree rooted at a. If f was previously a part of g's tree, on receiving the RELINQUISH message from g it not only leaves g's tree but also propagates the RELINQUISH message to node a, as described in Action 2. Node a propagates the JOIN message, as stated in Action 4, and absorbs f, g and h into its tree, adjusts its four-tuple and sends the updated four-tuple to the sink.

Next we discuss the correctness of the proposed solution for reacting to gradient disruption. But, let us first define the correctness criterion.

Definition 2. *Safe state: Each sensor node y belongs to the gradient-based tree of exactly one local peak x and the temperature of nodes increases monotonically along the path from y to x.*

Node y's transition out of the safe state can be due to two reasons:

1) Node y considers node x to be its local peak, but x has ceased to be a local peak.
2) Temperatures of nodes on the path from y to x are not monotonically increasing.

Lemma 2. *If a node transitions out of the safe state it comes to know of this transition within finite time.*

Proof: Let node y's transition out of the safe state be due to the fact that its local peak x has ceased to be a local peak. In such a situation x initiates a RELINQUISH message. Assuming that the underlying protocol stack ensures reliable inter-neighbor communication within finite time and that nodes do not fail, the RELINQUISH message propagates through the

entire gradient-based tree rooted at x. Hence, y receives the RELINQUISH message within a finite time of x ceasing to be a local peak and realizes that it is an *orphan* node, i.e., it is no longer in a safe state.

Let node y's transition out of the safe state be due to gradient-disruption on the path from y to its local peak. As described in Action 1-Step 2 above, the upstream node involved in gradient disruption initiates a RELINQUISH message on behalf of the local-peak of y's gradient-based tree. As this RELINQUISH message is sent down the gradient-based tree and is propagated by every recipient in the tree, y receives the RELINQUISH message in finite time. ■

Before we proceed with the rest of the proof, let us define two sets:

Definition 3. *NT_LP: set of local peaks of gradient-based trees that border the region containing the orphaned nodes (here the abbreviation NT is used to denote Neighboring Trees).*

Definition 4. *New_LP: set of new local peaks formed as a result of gradient disruption. This set could possibly be empty.*

Lemma 3. *A sensor node returns to the safe state within a finite time of transitioning out of the safe state provided no additional gradient-disruptions occur.*

Proof: For each gradient-based tree T that borders the orphaned region there exists at least one sensor node, s, that borders the orphaned region. Node s propagates JOIN() message into the orphaned region. Each new local peak $x \in New\_LP$ also propagates a JOIN() message into the orphaned region. Each new local peak x enters the safe state on account of being the root of a gradient-based tree created by itself. For all other orphaned nodes, we prove the lemma by contradiction.

Let us assume a non-local peak orphaned node y is unable to return to the safe state because y never receives a JOIN() message from a neighbor with a higher temperature. This implies that on every path from $z \in NT\_LP \cup New\_LP$ to y composed only of non-local peak nodes (excluding z) there is at least one sensor node $s_{barrier}$ with a higher temperature than its predecessor. It can be shown that the set of $s_{barrier}$ nodes form a perimeter around y such that all the nodes on the perimeter have temperature greater than their neighbor(s) outside the perimeter. For ease of understanding we propose the following analogy from terrain mapping: this perimeter is analogous to a ridge in a mountainous region.

For y not to receive any JOIN message it would also require that there be no local peak either on the perimeter or within it. However, this is impossible. Hence, the assumption that y never receives a JOIN() message is contradicted. ■

D. Failure and Recovery

We consider two kinds of failures:

1) Node failure, and
2) Link failure.

Node failure occurs when a sensor node ceases to function, either due to depletion of its energy supply or due to some hardware failure. Link failure occurs when an operational node is unable to communicate with one or more neighbors over an extended period of time. The underlying neighbor discovery protocol may declare failure of a link with a neighbor if no heartbeat message is received from that neighbor for a predefined period of time, or if an attempt to propagate a tree construction/maintenance message to the neighbor fails inspite of retransmissions at the MAC sublayer.

From the perspective of a given node S_i, link failure and node failure are indistinguishable: S_i may be unable to communicate with a neighbor S_j either due to the failure of the wireless link between them or due to the failure of S_j. The first reaction to an inability to communicate with a neighbor S_j is to remove S_j from Nbr_i. Subsequent recovery of a failed node S_j, reestablishment of the link between S_i and S_j, or deployment of a new node S_j will result in S_i adding S_j to Nbr_i. In the following paragraphs we explain all actions of S_i when there is a change to Nbr_i. We would like to stress that *the response to node or link failures is very similar to the response to gradient disruption.*

S_j **is removed from** Nbr_i**:** Node S_j could be a child or parent of S_i in a gradient-based tree, or there could be no parent-child relation between them.

- If S_j was S_i's child, S_i updates its four-tuple to reflect the removal of the subtree rooted at S_j. Then, S_i sends the revised four-tuple to its parent. Each node on the path between S_i and the root of S_i's gradient-based tree (including the root) recomputes its four-tuple on receiving the updated four-tuple from a child. Finally, the root sends its updated four-tuple to the sink.
- If S_j was the parent of S_i in some gradient-based tree then S_i acts as if it received a RELINQUISH() message from S_j, and diffuses the RELINQUISH() message through its subtree. If S_i realizes that it has become a local peak due to the departure of S_j from its neighborhood, it initiates the formation of a gradient-based tree rooted locally.
- If S_j is neither the parent, nor a child of S_i then S_i does nothing else.

S_j **is added to** Nbr_i**:** If S_i was a local peak, but has a lower temperature than S_j, S_i diffuses a RELINQUISH(S_i) message through its gradient-based tree orphaning all the nodes in its tree. Subsequently, these nodes need to join existing or newly formed gradient-based trees, as described earlier. If S_j is not part of any gradient-based tree and its temperature is lower then or equal to the temperature of S_i, S_i sends a JOIN message to S_j to absorb S_j into its tree. If S_i is not a local peak and S_j has a higher temperature than S_i, S_i does not need to do anything. If a newly found neighbor S_j is a local peak, it will form its own gradient-based tree and register itself with the sink. If the newly found neighbor is not a local peak it will join an existing tree because its temperature is less than or equal to at least one of its neighbors.

V. Related Work

The problem of scalable and efficient in-network aggregation in wireless sensor networks has been extensively researched since it was introduced by Estrin et. al. in [8]. While some works such as [1], [2] avoid explicit structures, many other works mainly focus on efficient data compression and aggregation by considering construction and maintenance of a framework. Also some works such as COUGAR [9] and TinyDB [10] have studied use of embedded database systems in sensor networks. Two types of the popular structure-based approaches are the tree based and cluster based aggregation techniques.

Cluster-based protocols form clusters of sensors around cluster-heads. Nodes report data to cluster-head. The cluster-head in turn reports either to other cluster-heads or to the base station. LEACH protocol presented in [11] forms cluster around randomly selected cluster-heads. These cluster-heads aggregate data and directly send it to the base station. Heinzelman et. al. [12] enhance LEACH by using the base station for cluster-head assignment. Zhao et. al. [13] further refine the cluster-head selection process in the LEACH protocol to improve the results. These approaches require significant energy consumption for the setup stage, and may require periodic re-structure to improve cluster quality. LEACH-based protocols also assume the distances from cluster-head to the base station and from the nodes to cluster-heads are only one hop each. Such assumptions may not be valid in some networks.

The alternative method is to form a tree and use the tree as the aggregation backbone. Madden et. al. in [3] and [4] use a shortest path tree to form a Tiny AGgregation service (TAG) framework. TAG allows parents to notify their children about the waiting time for data gathering, and the sleeping cycles can be adjusted accordingly. Intanagonwiwat et. al. in [5] propose a Greedy Incremental Tree (GIT) to establish an energy efficient path based on Directed Diffusion [14]. These tree based protocols construct and maintain a static tree without knowledge of the underlying application and data correlation.

Dynamic Convoy Tree-Based Collaboration (DCTC) is introduced in [15] to reduce the overhead of tree migration and re-structure. DCTC assumes distance to the event center is known and assigns the closest node as the root of the local tree. Such improvements may not be efficient in case of gradual movements, as high volume of message exchanges will be required. Trees avoid double counting as there is one route between each node and the sink, but the tree structure is vulnerable to node failures. Our work focuses on constructing a forest of gradient-based tree structures that dynamically change based on the sensor readings. The root of the local tree will act as the cluster-head and reports the locally aggregated data to the sink node. The forest is able to quickly recover from node failures.

Our work is also similar in goals to extended query types as discussed by Shrivastava et. al. in [6]. They propose *q-digest* data structure for data aggregation. Each node will merge and compress q-digest data recieved from its children in the routing tree, then passes it to its parent. If the the sensor readings are integers in $[1, \sigma]$ range, then by using message size m and *q-digest* they can answer quantile queries within an error of $O(log(\sigma)/m)$.

Fasolo et. al. in [7] provide a survey of existing in-network techniques and protocols. They provide a summary of the protocols and the advantages and disadvantages of each protocol or data presentation. Some open issues and the direction for these open issues are also provided in this survey.

VI. Simulation Experiments

We conducted simulation experiments to evaluate the performance of GrAFS using the ns-3 simulator [16]. One hundred sensor nodes were uniformly distributed in a $900m \times 900m$ area. The communication and interference ranges of each node were set to $100m$ and $200m$, respectively. We simulated the IEEE 802.11 Phy+MAC protocol with a peak data rate of 54 Mbps for wireless communication. We chose to conduct the experiments using ns-3 as we found TOSSIM [17] to be wanting in simulating all the features of GrAFS. The choice of IEEE 802.11 protocol was made because we envision the sensor nodes in a large scale network to be significantly farther apart than those in a WPAN running the IEEE 802.15.4/ZigBee stack.

Each node transmitted a *heart beat* message every five seconds as a MAC broadcast. The heart beat message carried the identity and temperature of the transmitter. This aided in neighbor discovery and in determining if a node was a local peak. The temperature of each node varied over time and was normally distributed with a mean of 70 $units$, and a variance of 20 $unit^2$. The time between changes in the temperature of a node was normally distributed with a mean of 180 $seconds$, and a variance of 60 $second^2$.

References

[1] K. Fan, S. Liu, and P. Sinha, "Scalable Data Aggregation for Dynamic Events in Sensor Networks," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*. ACM, 2006, pp. 181–194.

[2] K. Fan and P. Sinha, "Distributed Online Data Aggregation for Large Scale Sensor Networks," in *5th IEEE International Conference on Mobile Ad-Hoc and Sensor Systems MASS*, 2008, pp. 153–162.

[3] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks," *ACM SIGOPS Operating Systems Review* , vol. 36, no. SI, pp. 131–146, 2002.

[4] S. Madden, R. Szewczyk, M. Franklin, and D. Culler, "Supporting Aggregate Queries Over Ad-Hoc Wireless Sensor Networks," in *Proceedings Fourth IEEE Workshop on Mobile Computing Systems and Applications*, 2002, pp. 49–58.

[5] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann, "Impact of Network Density on Data Aggregation in Wireless Sensor Networks," in *Proceedings of 22nd International Conference on Distributed Computing Systems*, 2002, pp. 457–458.

[6] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, "Medians and Beyond: New Aggregation Techniques for Sensor Networks," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*. ACM, 2004, pp. 239–249.

[7] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-Network Aggregation Techniques for Wireless Sensor Networks: A Survey," *IEEE Wireless Communications*, vol. 14, no. 2, pp. 70–87, 2007.

[8] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks," in *Proceedings of the 5th Annual ACM/IEEE International conference on Mobile Computing and Networking*. ACM, 1999, pp. 263–270.

[9] Y. Yao and J. Gehrke, "The COUGAR Approach to In-Network Query Processing in Sensor Networks," *SIGMOD Record*, vol. 31, no. 3, pp. 9–18, 2002.

[10] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: An Acquisitional Query Processing System for Sensor Networks," *ACM Transactions on Database Systems*, vol. 30, no. 1, pp. 122–173, 2005.

[11] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks," in *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, 2000, p. 10 pp. vol.2.

[12] ——, "An Application-Specific Protocol Architecture for Wireless Microsensor Networks," *IEEE Transactions on Wireless Communications*, vol. 1, no. 4, pp. 660–670, 2002.

[13] L. Zhao, X. Hong, and Q. Liang, "Energy-Efficient Self-Organization for Wireless Sensor Networks: A Fully Distributed Approach," in *IEEE Global Telecommunications Conference (GLOBECOM '04)*, vol. 5, 2004, pp. 2728–2732 Vol.5.

[14] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed Diffusion for Wireless Sensor Networking," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 2–16, 2003.

[15] W. Zhang and G. Cao, "DCTC: Dynamic Convoy Tree-Based Collaboration for Target Tracking in Sensor Networks," *IEEE Transactions on Wireless Communications*, vol. 3, no. 5, pp. 1689–1701, 2004.

[16] "The ns-3 network simulator," http://www.nsnam.org, July 2009.

[17] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications," in *Proceedings of First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, 2003.