

Energy-efficient and Fault-tolerant Resolution of Topographic Queries in Networked Sensor Systems*

Mitali Singh and Viktor K. Prasanna
Department of Computer Science
University of Southern California
Los Angeles, CA-90089, USA.
Email: {mitali, prasanna}@usc.edu

Abstract

This paper focuses on energy-efficient and fault-tolerant resolution of topographic queries in dense, uniformly deployed, two-dimensional sensor systems. Our approach is based on construction of the topographic map of user-defined features in the network. Once constructed, the map is used to resolve a large number of topographic queries efficiently. We present a distributed algorithm for construction and maintenance of the topographic map in presence of node failures and discuss resolution of topographic queries using the map. Our results show that our algorithm incurs 90% lesser time and 50% lower energy overheads on the average, and recovers more reliably from node failures in the network than the state-of-the-art.

1. Introduction

Sensor systems have large communication costs and are prone to frequent node failures. Thus, it is more difficult to maintain global state in these networks as compared to traditional computing systems. This has led to the belief that solutions that maintain global states are not suitable for sensor systems. Several research efforts have focused on designing localized algorithms that abstract the network as a “stateless” system [18]. However, we believe that maintaining global state in a distributed manner can be beneficial in several sensor applications. In this paper, we presented a *stateful* approach towards resolution of topographic queries in dense, uniformly deployed sensor systems. We demonstrated that our approach is energy efficient and fault tolerant to node failures in the network.

Topographic querying is the process of extracting data from a sensor network for understanding the graphic delin-

ation of features of interest in a terrain. Application areas where such information is particularly useful include contaminant monitoring, tracking soil moisture levels, water temperature estimation of river beds, etc. [9] [17]. A prototype implementation of a sensor network used for constructing topographic maps of soil moisture and temperature in vineyards is described in [3]. Topographic information about properties such as residual energy are useful for resource management, dynamic re-tasking and preventive maintenance of sensor networks. Boundary estimation and enumeration of feature regions is used in acoustic monitoring and tracking applications [14].

As opposed to geographic locations, topographic queries extract information about feature regions in the network. Resolution of such queries requires global (network-wide) collaboration among the sensor nodes (e.g., region identification and boundary estimation). Thus, techniques involving query-initiated explicit data aggregation and exfiltration [18] cannot be used to resolve these queries efficiently. We explore an alternate solution based on distributed construction and maintenance of the topographic map in the network for *features of interest that do not change very frequently*. Once the map is constructed, it can be used for efficiently routing and rapidly answering queries in the network. For example, a query to count the number of feature regions of interest need not trigger any activity other than obtaining the result from a distributed storage node (as discussed in Section 3). Processing of topographic information and responding to queries could be in most cases decoupled from the actual data gathering process, which can occur independently and at a lower frequency.

We compare our algorithm to the state-of-the-art [11] using the following metrics: (a) the time and energy for extraction and storage of the topographic information in the network, (b) the time and energy for resolving topographic queries using the stored information, (c) the time and energy overheads, and success rate in recovering from node failures

*This work is supported by the DARPA Power Aware Computing and Communication Program under contract no. F33615-02-2-4005 and in part by the NSF under grant number IIS-0330445.

in the network, (d) the correctness of the topographic information maintained, and (e) the accuracy of the query results. Network-level simulations are performed to compare the performance of the algorithms for several types of feature distributions and in various node failure scenarios. Our results show that on the average, our algorithm incurs 50% lower energy and 90% less time overheads in extracting and maintaining topographic information, and recovering from node failures in the network. Our algorithm is always able to recover from node failures, but the state-of-the-art fails in 5% of the cases due to undetected loops. Up to 75% error is observed in labeling feature regions using the state-of-the-art, whereas the error is less than 37% using our algorithm. Both algorithms succeed in obtaining results for 90% of the queries. The query results obtained by our algorithm are 100% correct, but those using the state-of-the-art are largely erroneous in several failure scenarios.

The rest of the paper is organized as follows. The next section defines our system model, the problem to be solved, and the definitions. Our algorithm is presented in Section 3, and compared with the state-of-the-art in Section 4. The simulation framework is described in Section 5, followed by the results in Section 6. We compare our work with other related research in Section 7, and conclude in Section 8.

2. Preliminaries

2.1. System Model

1. We assume that n sensor nodes are uniformly distributed over a two dimensional terrain. Each sensor node is uniquely identifiable by its geographical coordinates (required for gathering topographic information) and is equipped with a low power processor, a local clock and a wireless radio with fixed transmission range r .

2. The system is represented by a graph $G(V, E)$, where V is the set of vertices representing the sensor nodes and E is the set of edges. Edge $(u, v) \in E$ exists between two vertices u and v provided they lie within the communication (radio) range of each other. All edges in the graph are symmetric. The sensor nodes that share an edge are called neighbors. The node distribution is sufficiently dense to ensure that the graph is connected. However, we assume that the graph has a small (constant) degree and each sensor node maintains a list of all its neighbor IDs.

3. No global clock exists in the system. Sensor nodes communicate asynchronously over the wireless channel using message of size $O(\log n)$ bits. Two types of messages are supported: (a) a *broadcast* is a one-to-many message from the sender node to all its neighbors, and (b) a *unicast* is a one-to-one message destined to a specific neighbor.

4. The wireless channel is locally shared - a sensor node can receive a message provided only one of the nodes within

its radio range (neighbor) transmits at a time. A medium access mechanism is supported in the system for resolving channel contention when multiple sensor nodes access a shared channel at the same time. For the sake of analysis only, we assume that all messages are reliably transmitted to the neighbors in a small (constant) number of steps.

Time and energy analysis: As is customary in the analysis of asynchronous distributed systems [12], for analysis only, we assume that all sensor nodes operate at the same clock frequency and there exists a logical global clock in the network. One time step corresponds to a cycle of the logical clock. We assume that communication of a message between neighboring nodes takes one time step. The time complexity of an algorithm represents its overall execution time as measured by the logical clock. We define one unit of energy as the energy dissipated in transmission or reception of one message over the wireless channel. Communication of a single message between neighbors dissipates one unit of energy at the sender and each receiver node.

Node failure model: We study the fault tolerance of our algorithm under two widely different node failure models: the **isolated failure model** and the **geographically correlated failure model**. Similar models have been used for studying fault-tolerance in [6]. The isolated failure model captures randomly distributed, independent node failures. The geographically correlated failures model results in the failure of all nodes within a circle of specified radius. The justification for this model is that continuous usage or environmental effects within a geographic region can cause such correlated failure, either due to loss of connectivity or due to energy dissipation. The location of the centers of these circles is randomly distributed within the network.

2.2. Definitions

A *feature* is a user defined predicate on sensor data (e.g., “*is temperature greater than forty?*”). We assume that the number of features in a network is constant, and a sensor node can locally determine if it satisfies a feature. The sensor data used to determine whether it satisfies a feature is called the *feature value* (e.g., temperature reading).

Consider a partition of the network into a grid consisting of cells of size $d \times d$. Two sensor nodes are said to be *g-connected* (geographically contiguous) if they belong to the same or neighboring (north, south, west, east) cells. Here, d defines the resolution factor for region identification. Our assumption about dense deployment ensures that $d \leq r$, implying that the sensor nodes in neighboring cells are also wireless neighbors. $G_f = (V, E_f)$ denotes the *feature graph* for feature f . The vertices $v \in V$ represent the sensor nodes, and an edge $e(u, v) \in E_f$ exists iff u and v are g-connected and both satisfy feature f .

A *feature region* refers to a geographically contiguous area in the network consisting of sensor nodes that satisfy the same feature. Note that connected components in a feature graph represent feature regions. A *feature path* represents a sequence of geographically contiguous sensor nodes that satisfy the same feature. Consider the set of shortest feature paths between all pairs of sensor nodes in a feature region. The *diameter* of the feature region is defined as the length of the longest such path. Sensor nodes are said to be *weakly feature connected* if the number of distinct feature paths between them is small.

A *topographic map* represents the delineation of features of interest in a terrain by means of contour lines. In this paper, we use the term topographic map to represent a distributed data storage infrastructure that maintains topographic information (such as boundary and aggregates over feature values) about feature regions in the network.

The sensor node that injects a query in the network is called a *query node* for that query. We assume that a query can originate anywhere in the network.

2.3. Problem to be solved

Given a dense, uniformly-deployed, two-dimensional network of sensor nodes that store features of interest to end users, perform the following in a time and energy efficient, and fault-tolerant manner.

1. Construct and maintain a topographic map for user-defined features in the network: (a) Given a list of features, identify and label the feature regions in the network in a globally consistent manner (i.e. different regions do not have same label). (b) Extract and store the following topographic information in the network in a distributed manner: boundary and aggregates (sum/min/max) of feature values for each feature region, and global aggregates (such as number of regions, and sum/min/max of feature values) for each feature in the network.

2. Execute user-specified topographic queries: Using the stored topographic information, route and resolve the following topographic queries. For a user-specified feature, (a) count the number of feature regions in the network and (b) count the number of feature nodes in the network.

Note that the above set of queries is chosen for the sake of illustration only. Our algorithm can be easily customized to support several other topographic queries.

3. Our Algorithm

The key step involved in construction of the topographic map is identification and labeling of feature regions in the network, which is abstracted as the widely studied component labeling problem in classical image processing [1]. We exploit a *divide and conquer* approach for constructing the topographic map, described as follows.

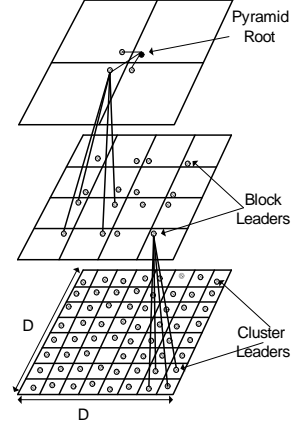


Figure 1. Pyramid-structure

I. Location-based clustering: A pyramid structure (see Figure 1) is overlaid on the network (independent of node distribution). Consider a network with dimensions $D \times D$, it is partitioned into four 2×2 **blocks** of size $\frac{D}{2} \times \frac{D}{2}$. This step is repeated recursively for each newly created block till the network is partitioned into blocks of size $c \times c$ called **clusters**. We choose $c = \frac{r}{\sqrt{5}}$ to ensure that sensor nodes in adjacent (north, south, west, east) clusters are wirelessly connected. In each cluster, the sensor node closest to the center of the cluster (identified by scanning the neighbor list) is chosen to be the **cluster-leader**. In each block, the cluster closest to the center of the network is called the **ideal-leader-cluster** of the block; the cluster-leader of this cluster is selected to be the **block-leader**. Each block is assigned a **level** based on its size. A block of size $(2^{k-1} \cdot c \times 2^{k-1} \cdot c)$, is called a block of level k , here $(1 \leq k \leq \log_2 D)$. A block-leader is assigned the same level as the largest block for which it is a block-leader. Note that the block-leader of a block of level k is also the block-leader of all the smaller blocks in which it lies (it belongs to the cluster closest to the center in the smaller blocks as well). The above implies that a block-leader of level k is also a block-leader at all levels less than k . The block-leader of the block representing the entire network is called the **pyramid-root** (smaller id is used to break ties between block-leaders equi-distant from center of the network for electing the pyramid root). Each block of level k contains three block-leaders of level $k - 1$ (not counting the block-leader), these are defined as the **subblock-leaders** of the block.

A sensor node can locally determine using its coordinates, whether it is the block-leader for a specific block. In the ideal scenario, every cluster has at least one sensor node, and a pyramid structure is constructed over the network in a distributed manner without involving any communication in the network. In more realistic scenarios, there may be **empty** clusters (without any functional nodes) in the network. If the ideal-leader-cluster is empty, no block-leader

is selected for the block. Missing block-leaders are treated as failed block-leaders. Detection and recovery of failed block-leaders is discussed later in the paper.

II. Map construction: This step identifies and labels feature regions in the network and computes the global and region-specific aggregates. All the **non-leader nodes** (sensor nodes that are not cluster-leaders or block-leaders) transmit their coordinates and feature values to the cluster-leader. The cluster-leader locally processes the received data to construct a *feature graph*. It performs depth-first-search on the feature graph to label connected components, which represent the feature regions in the cluster [1]. Global and region-specific aggregates are computed over the feature values in the cluster. Next, the feature graph is *reduced* as follows. A vertex is used to represent each feature region in the cluster, and an edge is added to the vertex for every sensor node in the feature region that lies on the cluster boundary. A feature region is said to be *contained* within a cluster (or block) if no sensor node in the feature region lies on the cluster boundary. The reduced graph and region-specific aggregates for *non-contained* feature regions is routed along with the global aggregates to the **next-level block-leader** (block-leader of the next level block). Initially, the block-leaders do not know the id of the next-level block-leader, and thus the packets are routed to the ideal-leader-cluster of the next-level block-leader using block routing (described later in the paper).

Each block leader locally processes the information received from the cluster-leaders to identify and label feature regions in the block (using same techniques as described above). It computes and routes the aggregates and the reduced graph representing non-contained feature regions to the next-level block-leader. The above steps are repeated recursively. On the k^{th} recursive iteration, each block-leader of level k processes information received from its subblock-leaders, and sends the aggregates and reduced graph for features in block to the next level block leader.

The smallest block containing a feature region is called the **parent-block** and its block-leader is called the **parent-leader** of the feature region. The parent-leader labels a locally contained feature region as $(x\ y\ k\ i)$, where $(x\ y)$ are the coordinates of the center of the block, k is the block level, and i is the smallest node id in the feature region. The parent-leader routes the label and region aggregates (constant information) to the subblock-leaders from which it received data for the region. On receiving the data, the block-leaders of the smaller blocks repeat the above steps recursively. In the final iteration, the cluster-leaders transmit the information to the non-leader nodes in the region.

At the end of this step, the topographic information is stored in the network in a **distributed, hierarchical, multi-resolution** manner. The pyramid-root stores the global aggregates (e.g., number of nodes, regions, max/min/sum of

feature values) for all the features in the network, whereas the block-leaders store the block aggregates (computed over the sensor nodes in the block). The cluster-leaders maintain feature lists representing feature values and coordinates of the sensor nodes in their cluster (region boundaries can be computed using this list). Each sensor node stores constant information – the label, region-specific aggregates and information about the parent-block for its feature region.

III. Resolution of topographic queries: The query nodes route the queries to the pyramid-root, which maintains the global aggregates such as the count of the number of sensor nodes and feature regions for each feature in the network. The pyramid-root locally computes and routes the responses to the query nodes. Resolution of several other topographic queries using the map is discussed in [16].

3.1. Packet losses and link failures

We use end-to-end application level acks to ensure that the functionality of our algorithm is not affected due to transient link failures or packet losses in the system. Since, the algorithm involves only one-to-one communication, acks can be implemented without large overheads.

3.2. Detecting failures of sensor nodes

We observe that the failure of a non-leader sensor node in the network does not impact the functionality of the algorithm. A non-leader sensor node only participates in the initial step of sending its feature data to the cluster-leader, and the final step for receiving the region labels and aggregates. It does not maintain any other topographic information except the label, aggregates and parent-block for its feature region (stored by all the sensor nodes in the feature region). Thus, we focus only on the failure of leader sensor nodes.

The sensor nodes maintain updated neighbor lists, which are used and refreshed periodically by the lower level protocols (such as routing and medium access). We assume that in addition to maintaining ids and positions of one hop neighbors, the sensor nodes also store the leader level for one-hop neighbors in the lists.

1. **Local detection:** A sensor node locally detects failure of its one-hop neighbors without incurring any overheads, by monitoring its neighbor list. These include nodes in its own and (N S W E) adjacent clusters.

2. **Ack-based detection:** When a sensor node does not receive an ack for a message, it resends the information and waits for the ack. On failing to receive an ack after resending data several times, it assumes that the receiver node has failed or is not reachable.

3. **Failure notification messages:** When a cluster-leader detects failure of all the sensor nodes in the (N S W E) adjacent cluster, it sends failure notification messages (with id and level of the failed block-leader) to the subblock-leaders

of the blocks (for which one of the failed nodes was the leader) using **block routing** (discussed below). Explicit failure notification expedites detection of distant node failures, and unlike ack-based detection, it can be used to detect failures when there is no ongoing communication in the network. The communication overheads incurred are proportional to the size of the largest block for which the failed node is the block-leader.

Block routing: Several steps in our algorithm require information exchange between block-leaders and subblock-leaders (or cluster-leaders) and vice-versa. When the id (position) of the destined leader is known, packets are routed using GPSR [2] [10]. For scenarios, where the sender does not know the id of the destined leader, we use *block routing*. Using block routing a packet can be routed to any block-leader in the system, provided the sender knows k , the level of the block-leader, and (x, y) , a set coordinates that lie inside the block for which the destined node is the leader. This is achieved as described below.

Using k and (x, y) , the sender node locally computes (x_c, y_c) , the coordinates of the center of the destination block. The block size is $b \times b$, where $b = 2^{k-1} \times c$. $x_c = (x \div b) \times \frac{3}{2} \cdot b$ and $y_c = (y \div b) \times \frac{3}{2} \cdot b$, where, \div represents integer division. The packet is routed using GPSR with center of the block as destination coordinates. In addition to the destination address, the packet stores a field to mark that it is a block routing packet and a field to store the level of the block-leader. Each sensor nodes that receives the packet processes it using the following rules.

R1. If the receiver node lies outside the destination block and does not contain any neighbor that lies in the destination block, it processes the packet as a normal GPSR packet, continuing to route it towards (x_c, y_c) .

R2. If the receiver node lies outside the destination block and contain a neighbor that lies in the destination block, it routes the packet to the neighbor that lies in the block.

R3. If the receiver node lies inside the destination block, it checks whether it is the destined leader. If so, it processes the packet, else routes it using the following rule.

R4. If the receiver node lies inside the destination block, but it not the destined leader, it routes the packet to its (next level) block leader (using GPSR if block-leader position is known else using block routing). This step is repeated by the higher-level block-leaders until the packet reaches the destined block-leader. In the scenario, that an intermediate node was notified that the next-level block-leader has failed, the packet is buffered until a new block-leader is elected to replace the failed node.

To ensure efficient delivery of packets using block routing (especially when using R4), it is important to route packet to the center of the smallest block to which the des-

tinued block-leader belongs. For example, consider the scenario, where a packet destined to a level k block-leader, reaches the $k - 1$ block-leader in the same block, and the $k - 1$ block leader must use block routing to further route the packet. The $k - 1$ block leader does not route the packet to the center of the block of level k . Instead, it routes it to the center of the block of level $k - 1$ most likely to contain the level k leader for the block. The node selects the block closest to the center of the network as the most likely block, unless its previous block-leader belonged to a different block (this block is chosen then).

3.3. Recovering from sensor node failures

1. **Leader election to select a new block-leader:** If the cluster of the failed leader is not empty, its failure is locally detected by the other sensor nodes in the cluster. Each sensor node scans the neighbor list to identify the sensor node in the cluster that has the maximum remaining power. If several sensor nodes have same power, the one closest to the center of the cluster is selected as the new leader. Node ids are used to break ties between equidistant nodes.

Next, we consider the scenario where a block leader fails, and there is no other node in the cluster to take over its role. The cluster-leader of the cluster in the block that is closest to the ideal-leader-cluster of the block, is selected as the new leader (using the following steps). This ensures that the pyramid structure is as close to the ideal case scenario as possible. When more that one such leader exists, the one with higher remaining power is chosen.

All the subblock-leaders in the block are informed of the failure of the block-leader, using failure-notification messages. Leader election takes place in rounds involving communication between the subblock-leaders in the block. In round i , subblock-leaders of level i communicate with each other. One of the subblock-leaders (the one with the maximum energy) is elected to be the new block-leader, and the algorithm terminates. If there are no subblock-leaders at level i (also implying that the block of level $i - 1$ containing the failed block-leader is empty, else there would be at least one subblock-leader of level i), no communication takes place in the i^{th} round and the $(i + 1)^{th}$ round is executed. The algorithm terminates after k rounds in the worst case, where k denotes the level of the failed block-leader. If the block is not empty, a new leader is selected before $i \leq k$ rounds, and the new leader belongs to the smallest non-empty block to which the failed node belongs.

Let us assume that a new leader is elected in the i^{th} round. A constant number of messages are exchanged between the subblock-leaders of level i , which are $O(2^i)$ hops away. Messages exchange takes $O(2^i)$ time steps and dissipates $O(2^i)$ units of energy. Let T_i define the start time for the i^{th} round. For the correct implementation of our algorithm we require that $T_i \geq T_{i-1} + O(2^{i-1})$. This

ensures that the i^{th} round does not begin before leader election is complete at the $(i - 1)^{\text{th}}$ round if there are any $i - 1$ subblock-leaders in the block. Thus, $T_i \geq O(\sum_{j=1}^{i-1} 2^j) = O(2^i)$, we choose $T_i = O(2^i)$. The overall time and energy taken for leader election is $O(2^i)$.

2. Recovering the information maintained at the block-leader: When a new block-leader is elected, it sends a new leader packet to its subblock-leaders and the sensor nodes in its cluster. On receiving the message, the sub block leaders (and the cluster nodes) resend the topographic information and queries to the new block-leader. The block-leader can compute the lost information by using this information.

Consider the failure of a block-leader of level k , which is replaced by a subblock-leader of level i in the block ($1 < i < k$). This implies that there are no sensor nodes in the block of level $i - 1$, to which the failed leader belongs. The new block leader received topographic information from the subblock-leaders of level $i \leq j < k$ in the block. A subblock-leader of level j is $O(2^j)$ hops away from the failed leader. It sends $O(2^j)$ packets to the new leader [15] that are routed in $O(2^j)$ time steps dissipating 4^j units of energy. Total time taken is $\sum_{j=i}^{k-1} O(2^j) = O(2^k - 2^i)$ and overall energy dissipation is $\sum_{j=i}^{k-1} O(4^j) = O(4^k - 4^i)$.

Worst-case analysis: The largest overheads are incurred when the pyramid root fails, which is of level $O(\log_2 D)$. We assume a constant number of sensor nodes in each cluster, and thus $O(D) = O(\sqrt{n})$. Let us assume that a k level subblock-leader is elected as the new block-leader, where ($1 \leq k < \log_2 \sqrt{n}$). Leader election requires $O(2^k)$ time steps and dissipates $O(2^k)$ units of energy. Recovery of information from subblock-leaders takes $O(\sqrt{n} - 2^k)$ time steps and $O(n - 4^k)$ units of energy. Total time taken is $O(\sqrt{n})$ and energy dissipation is $O(n - 4^k + 2^k) = O(n)$. Note that larger energy is dissipated when a lower level block-leader is elected as the new leader.

4. State-of-the-art

We consider the state-of-the-art **feature extraction algorithm** described in [11], which organizes the network into a forest of trees. Each tree represents a feature region in the network, and is rooted at the minimum id node in the feature region. The **root node** maintains aggregates computed over the feature values stored in the feature region. Tree construction dissipates $O(n^2)$ energy units and requires $O(n)$ time steps, while map construction dissipates $O(n \log_2 n)$ energy units and $O(\sqrt{n})$ time steps, assuming no node failures in the network (discussed in [15]).

Consider the failure of a *root node*, the recovery involves construction of a new tree rooted at the new minimum id node in the feature region. The recovery overheads are significantly larger than those involved in our algorithm (analyzed in Section 3). While recovery from root node fail-

ure requires reconstruction of the tree, simpler, less costly heuristics can be used to recover from failures of non-root nodes. No such techniques have been discussed in [11]. We use the localized tree maintenance heuristics described in [18] for locally recovering from node failures, and evaluate their performance using simulations in Section 6.

Next, we discuss resolution of the queries discussed in Section 2, using the information maintained by the state-of-the-art. Since the location of the root nodes is not known to the sensor nodes that are not in the same feature region, the queries are flooded in the network. The root nodes with the queried feature route the relevant region-specific aggregate relevant to the query node. The query node computes the global aggregates by processing the region-specific aggregates received. Assuming no node failures, this takes $\Omega(n)$ time steps and dissipates $\Omega(n)$ energy units, while query resolution using our algorithm takes $O(\sqrt{n})$ time steps and $O(\sqrt{n})$ energy units [16].

5. Simulation Framework

Our high-level analysis is based on the model described in Section 2, which makes several simplifying assumptions (such as collision-free communication and symmetric wireless links). We perform network-level simulations to evaluate the performance of the algorithms in a more realistic manner. The core of our framework is the **Global Mobile Systems Simulation Library** [7], which provides support for several widely-used radio models and networking protocols for wired and wireless networks.

Wireless channel: Radio specifications of the mote nodes [4] are used for configuring the radios layer parameters in our simulations. The radio range of the sensor nodes is fixed to 10m. Radio transmit and receive power are assumed to be 15 mW and 13.5 mW respectively. Signal propagation is simulated using the two ray path loss model. Packet reception model is based on SNR. The 802.11 protocol is used for resolving wireless contention.

Data communication: We implemented GPSR [2] [10] routing in GlomoSim for network-level routing of packets. Block routing is implemented in the application layer.

Asynchronous local clocks: All sensor nodes are equipped with local clocks which have the same frequency but are not time synchronized. The local clocks are generated by adding random offsets to the global clock in GloMoSim.

Network topology: Network topologies are generated using uniform node distribution over a two-dimensional terrain. We simulated networks of sizes 40m×40m with 120 nodes, 80m×80m with 400 nodes, and 160m×160m with 1370 nodes. The number of nodes is chosen such that the average node degree in the network is 20.

Feature distribution: Figure 2 illustrates the feature distributions used in our simulations. **gdw** illustrates the hydraulic conductivity in soil measured by a sensor system

deployed over a 30 acre test plot in Palmdale, CA [9]. **random** depicts the feature distribution without any spatial correlation. **snake** represents large diameter feature regions.

Node failures: We simulate scenarios with 10% (0.1), 20% (0.2) and 30% (0.3) of the sensor nodes failing in the system, with the following failure patterns: random (Ran), cluster (Clust), and radii (Rad) (see Figure 3). The random pattern represents the isolated failure model, any sensor node in the network fails randomly in the network. The cluster model is representative of the geographically correlated failure model, a cluster is chosen randomly in the network. All sensor nodes in the cluster fail at the same time. The radii model is similar to the cluster, but all the sensor nodes within circles of radii $5m$ fail together. The center of the circles is chosen randomly in the network. The failure timings are distributed randomly over the total duration.

Time and energy: The total time represents the communication time of the algorithm as measured by the GlomoSim clock. The overall energy dissipation is the sum of transmission and reception energy dissipated by the sensor nodes. Computation time and energy are negligible in comparison to communication [15].

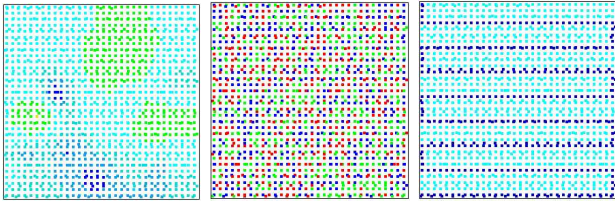


Figure 2. (a) gdw (b) random (c) snake

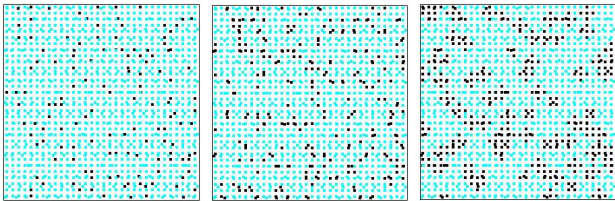


Figure 3. Failures (a) 0.1Ran (b) 0.2Clust (c) 0.3Rad

6. Simulation Results

I. Node failures during construction: The first set of experiments simulate failures during map and tree construction. We observe that our algorithm is always able to recover from node failures (possibly with some inaccuracies in results), but the state-of-the-art fails in 5% of the test cases. This happens when localized tree repair techniques create loops in the tree. The sensor nodes in the loop communicate with each other infinitely, exhausting all the resources, and resulting in large congestion in the network.

Energy Dissipation: Figure 4 shows the energy dissipation for the two algorithms. The increase in energy in failure cases (as compared to the no failure case) represents the failure detection and recovery overheads. We observe that the overall energy dissipation and the recovery overheads for map construction are 50% less (on the average) than the state-of-the-art. The energy increase is proportional to the number of failures for both algorithms, but the rate of increase is higher in the state-of-the-art. The map construction energy is lower for cluster failures than random failures, which is explained as follows. The recovery overheads increase with the number of leader node failures. Consider k random node failures in the network, in the worst case scenario, each failed node is a cluster leader. Given a total of k node failures in the network, the number of leader failures in the cluster failures is k/n_c , where n_c is the number of sensor nodes per cluster. Since $n_c \geq 1$, $k/n_c \leq k$. Radii failures dissipate the largest energy as the node failures in the interior of the circles are not locally detected, but only after several futile communication rounds. In some cases, the construction energy of the tree is less than the map, specifically for radii failures in small-sized networks. This happens when the failures result in partition of a larger region into two smaller regions. This reduces the tree construction energy, which is proportional to the region size.

Execution time: On the average, map construction takes 10% of the time required for tree construction using the state-of-the-art (see Figure 5). The time for tree construction is similar for most failure cases and problem sizes except when one or more root nodes in the network fail (spikes in the graph), and the tree is reconstructed. The time for locally recovering from a node failure only depends on the node density, which is the same in all the cases. Failure detection and recovery time in map construction is the largest for the radii failures. The node failures in the interior of the circles are detected using ack based detection, which incurs large time overheads (proportional to the network size in the worst case). The overall time and recovery time overheads also depend on the time of failures. If the failed node has already sent its information to the leader, the recovery process is executed in parallel to the main construction algorithm. This results in lower overheads as compared to the scenario, where the map construction algorithm must stall until the recovery (this explains why the figure shows no regular trend based on number or type of failures).

Maximum network queue size: The maximum network queue size (maximum over all nodes and total execution time) is an indicator of the maximum congestion in the network. Our algorithm shows largest congestion for the random feature distribution as it involves the largest number of boundary crossings. We observe this to be twice that of tree construction on the average, as shown in Figure 6. For

our algorithm, the largest congestion (and energy dissipation) takes place at the pyramid root. The tree construction algorithm performs badly for large diameter feature regions (such as the snake), and results in several sensor nodes with significantly larger queue size than the pyramid root in our algorithm. Moreover, several large spikes are observed for tree construction in Figure 6, which are due to formation of transient loops in the tree during repair. Transient loops refers to loops that are detected and broken off early (without resulting in infinite communication).

Accuracy in labeling feature regions: Next, we examine the accuracy of the two algorithms in correctly labeling the feature regions. Figure 7 illustrates the percentage of sensor nodes in the network that are assigned wrong labels by the two algorithms. Our algorithm shows 5% error on the average and 37% error in the worst case. Feature regions are wrongly labeled in map construction, when the block routing scheme fails to route the final labels to subblock-leaders (when failures result in large holes in a block).

The labeling error observed is significantly higher for the state-of-the-art: 30% on the average and 68% in the worst case. Larger error rate is observed for the snake and random feature distribution as compared to the *gdw* feature distribution. In the state-of-the-art, label information for a feature region is propagated among the sensor nodes in the feature region using unreliable broadcasts. Some of the label packets are lost due to interference. In most cases, every small area in a feature region contains several sensor nodes, and each of them broadcast the label. Since the label is broadcast several times in the area, the algorithm is not affected by the loss of a few packets. However, such is not the case in when the feature regions are weakly feature connected. It is possible that there is a single feature path connecting two parts of the feature region. When the label propagates along this path, a single packet loss can prevent it from being flooded in the entire feature region. Consequently, the feature region is partitioned into two smaller feature regions, only one of which is correctly labeled. Depending on the size of the wrongly labeled feature region, the error rate can be significantly high. The correctness of our algorithm (unlike the state-of-the-art) is not affected by packet losses and is not highly sensitive to the feature distribution.

Accuracy of stored topographic information: Lastly, we execute queries and compare the results to examine the accuracy of the topographic information maintained by the two algorithms. The queries are executed after (map or tree) construction is over. The results are illustrated in Figure 8. In the current implementations, we do not support instantaneous updates. Consider the scenario during map construction, where a cluster-leader detects the failure of a sensor node in its cluster after it has sent the data to the leader. The sensor node updates its local information but does not

resend the updated data to its leader. Consequently, the topographic information in the network is not always up-to-date. To accommodate deviation in results due to failures, we define a valid zone for each query (range of query result assuming no node failure and maximum node failures). A query is resolved correctly if the result is in the valid zone.

While 100% queries are correctly resolved using our algorithm, less than 10% queries are correctly resolved using the state-of-the-art. In several cases, the incorrect results deviate *by factor of two* from the correct values. Note that a single error in maintaining the tree links can result in large inaccuracies in the topographic aggregates maintained by the root. Consider the scenario, where a sensor node in the tree changes its parent. If the old parent is not correctly informed, the topographic aggregate of the child is used twice in computing the global aggregate. Depending on the values of the aggregates, the results can be largely erroneous.

II. Node failures during query resolution: This set of simulations is performed to compare the fault tolerance of the two algorithms during query resolution. No node failures are simulated in the (tree/map) construction step, but only during query resolution. 30 randomly placed topographic queries are executed in the network at time interval of 10sec. We observe that the recovery overheads show a similar trend as during construction. The query results obtained using the state-of-the-art are less erroneous on the average than those obtained when failures were simulated during tree construction. This is because once the tree is constructed, the query results are only affected by the failure of the root nodes. Since the location of failures is chosen randomly, and less than 1% of the sensor nodes are root nodes in our simulations, the probability of a root node failure is very low. Note that in the state-of-the-art, the energy dissipation in query resolution decreases with increase in number of node failures in the network (see Figure 9). This is because decrease in the number of functional nodes reduces the number of broadcasts in flooding the query.

7. Related Work

Several research efforts have focused on resolving spatial and/or aggregate queries in sensor networks using dynamically constructed, “well-structured” spanning trees [18]. Such techniques cannot be used to resolve topographic queries, which are routed towards feature regions as opposed to geographic locations. Identification and labeling of feature regions is a critical step in resolution topographic queries, which requires non-trivial computation and involves network-wide collaboration and state maintenance. [8] briefly discusses an in-network algorithm for contour mapping in sensor networks, but provides no details for the steps involved in identification and merging of isobars.

The work most closely related to ours is, which addresses the feature extraction problem in sensor networks.

We compare it with ours in Section 4 and 6. We use the term *fault-tolerance* to refer to the capability of the system to recover from node and link failures. We do not take faulty sensor readings into consideration, which can be recovered using the techniques described in [11]. Boundary estimation has been discussed from an information-theoretic perspective in [14]. Our approach to query-resolution is not “stateless”, but maintains global state in the system. [13] describes a state-centric programming model for sensor networks, which will facilitate specification and implementation of our algorithms. The pyramid structure used by our algorithm is similar to that used by the DIMENSIONS [5] data handling architecture. However, DIMENSIONS focuses on computation, storage and retrieval of spatial and temporal data aggregates and not on identification of feature regions, or extraction and storage of topographic data.

Our prior work: We discuss an energy efficient algorithm for construction of the topographic map in [15]. The performance of the algorithm is evaluated using high-level analysis and simulations without modeling node and link failures or collisions. In this paper, we present a fault-tolerant algorithm for construction and maintenance of the topographic map and analyze its performance and correctness in presence of node failures in the network. We perform network-level simulations to evaluate our algorithm in a more realistic manner, and present results for various node failure patterns and feature distributions in the network. [16] describes several other types of topographic queries that can be resolved efficiently using the map.

8. Conclusion

We discussed a distributed algorithm for energy-efficient and fault-tolerant construction and maintenance of topographic map in sensor networks. Once constructed, the map is used to resolve several types of topographic queries efficiently. We demonstrated that our algorithm is significantly more time and energy efficient, and fault-tolerant as compared to the state-of-the-art algorithm.

We thank our reviewers for their feedback, which has helped us in refining the directions for our future work. In this paper, we evaluated the performance of our algorithm in dense, uniform networks. While the functionality of our algorithm is not impacted by node topologies (provided the network remains connected), its performance may deteriorate for non-uniform and sparse node distributions. As part of our future work, we will compare our algorithm to the state-of-the-art for other node topologies. Also, the current implementation of the algorithm is not energy balanced. The leader nodes dissipate more energy than non-leader nodes. As part of our future work, we will develop a leader rotation scheme for more energy-balanced implementation of our algorithm. We will also discuss resolution of other topographic queries using the map.

References

- [1] H. M. Alnuweiri and V. K. Prasanna. Parallel architectures and algorithms for image component labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(10):1014–1034, 1992.
- [2] P. Bose, P. Morin, I. Stojmenovic, and J. Urruti. Routing with guaranteed delivery in ad hoc wireless networks. In *Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, August 1999.
- [3] J. Burrell, T. Brooke, and T. Beckwith. Vineyard computing: sensor networks in agricultural production. In *Pervasive Computing Magazine, IEEE*, pages 38–45, March 2004.
- [4] The CrossBow Corporation, <http://www.xbow.com>.
- [5] D. Ganesan, D. Estrin, and J. Heidemann. DIMENSIONS: Why do we need a new data handling architecture for sensor networks? In *HotNets Workshop*, October 2002.
- [6] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(4)(11-25), October 2001.
- [7] The Global Mobile Information Systems Simulation Library, <http://pcl.cs.ucla.edu/projects/glomosisim/>.
- [8] J. M. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: Towards sophisticated sensing with queries. In *Information Processing in Sensor Networks*, April 2003.
- [9] J. Kim, J. Saez, N. Busek, Y. Park, J. E. Haux, D. Estrin, and T. Harmon. Network sensing of nitrate in support of groundwater quality protection. In *Annual Research Review Poster, Center for Embedded Networks Sensing (CENS)*, 2004.
- [10] B. Karp and H. T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Int’l Conference on Mobile Computing and Networking*, August 2000.
- [11] B. Krishnamachari and S. S. Iyengar. Efficient and fault-tolerant feature extraction in sensor networks. In *Information Processing in Sensor Networks*, April 2003.
- [12] L. Lamport and N. Lynch. *Distributed Computing: Models and methods*. Formal Models and Semantics, Handbook of Theoretical computer Science, 1990.
- [13] J. Liu, M. Chu, J. Liu, J. Reich, and F. Zhao. State-centric programming for sensor-actuator network systems. In *IEEE Pervasive Computing*, 2003.
- [14] R. Nowak and U. Mitra. Boundary estimation in sensor networks: Theory and methods. In *Information Processing in Sensor Networks*, April 2003.
- [15] M. Singh, A. Bakshi, and V. K. Prasanna. Constructing topographic maps in networked sensor systems. In *Workshop on Algorithms for Wireless and Mobile Networks*, August 2004.
- [16] M. Singh and V. K. Prasanna. Supporting topographic queries in a class of networked sensor systems. In *PerSeNS, Int’l Conference on Pervasive Computing*, March 2005.
- [17] The klamath river flow studies, http://www.krisweb.com/krisweb_kt/klamflow.htm.
- [18] J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In *SNPA, Int’l Conference on Communications*, May 2003.

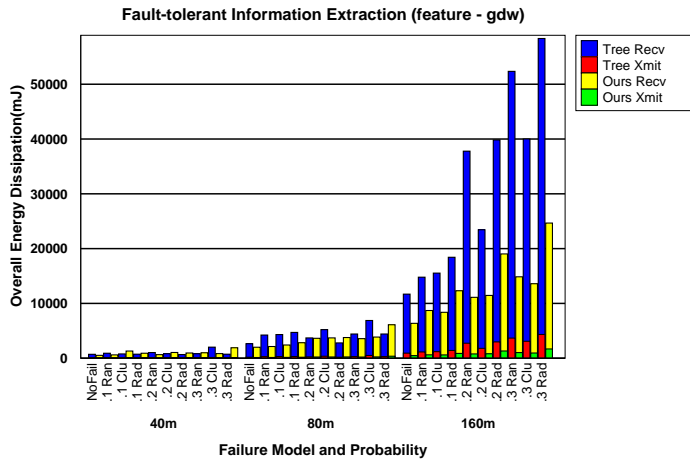


Figure 4. Construction energy

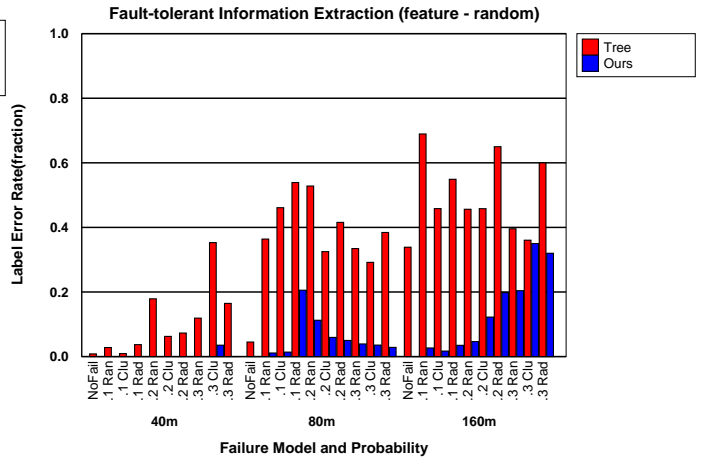


Figure 7. Error in assigning region labels

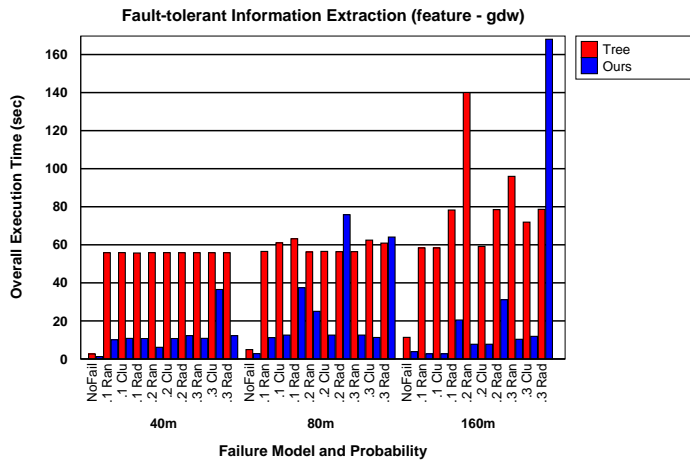


Figure 5. Construction time

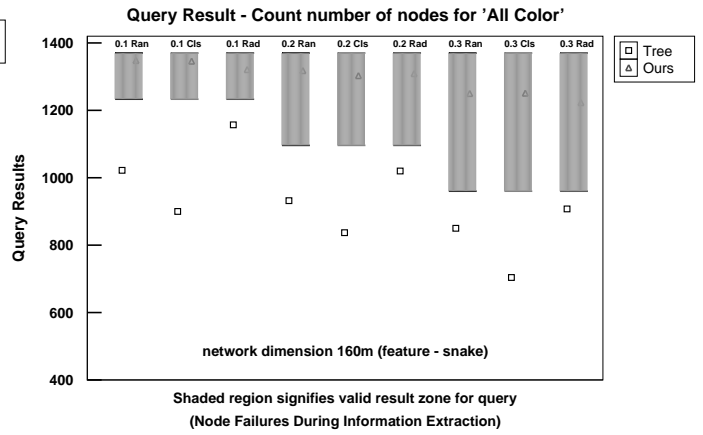


Figure 8. Results for Query I

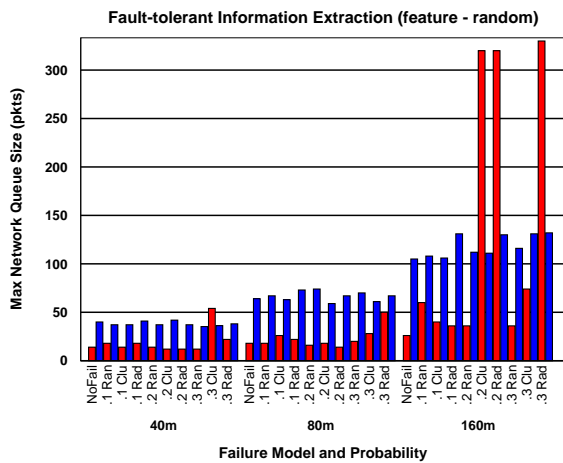


Figure 6. Maximum network queue size

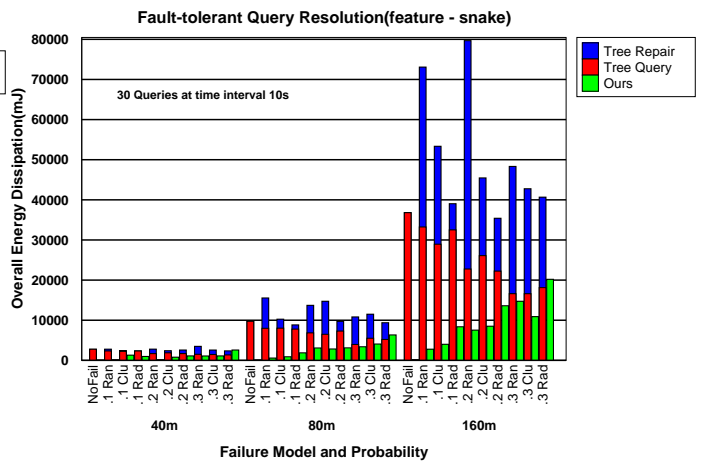


Figure 9. Query energy