

# Constructing Topographic Maps in Networked Sensor Systems

Mitali Singh, Amol Bakshi, and Viktor K. Prasanna  
 University of Southern California  
 Los Angeles, CA 90089 USA.  
 {mitali, amol, prasanna}@usc.edu

**Abstract**—Topographic maps represent the delineation of features of interest in a terrain by means of contour lines. There are several benefits of maintaining topographic maps in sensor networks - they can be used for resolving spatial queries, visualizing available resources in the network, and understanding spatio-temporal characteristics about the events being monitored. In this paper, we consider the problem of distributed construction of topographic maps in sensor networks. The contour lines in the map represent contiguous points (sensor nodes) in the terrain that have the same value of the measured feature. We propose an energy-efficient algorithm that constructs these maps in a distributed manner. We also discuss how the maps can be used for efficient execution of some common sensor network queries as compared to existing techniques. The performance of the proposed algorithm is demonstrated to be significantly better than the state-of-the-art in terms of overall energy and latency. Analytical inferences are validated through lower-level simulations.

## I. INTRODUCTION

Networks of smart sensors communicating through wireless links are being deployed for a host of applications that involve monitoring and responding to the physical environment. A sensor node consists of a processing unit, a transceiver, one or more sensors, and a limited power supply. Significant advances are being made in the hardware architectures of sensor nodes [1], [2], paradigms for in-network query processing, routing and storage [3], system-wide network monitoring [4], etc. Many research challenges in embedded networked sensing stem from energy constraints and limited communication and computation capabilities of the individual node.

Environment monitoring is a fundamental application of sensor networks. A sensor network could be deployed over a two-dimensional terrain with each sensor node equipped with a temperature sensor. The end user might be interested in extracting a variety of topographic information about the metadata in the network. He might want to know the boundaries of all regions where the temperature exceeds a certain threshold, or just the number of disjoint regions where temperature exceeds a threshold, or might want the complete topographic map of the terrain where contours correspond to temperature levels. The end user might also want to periodically construct topographic maps representing the battery power of the sensor nodes for the purpose of resource monitoring and load-

balancing in the network. Energy constraints make it infeasible to employ centralized approaches that involve transmission of data from individual sensor nodes directly to the query node. Instead, it is desirable to perform distributed in-network processing of the sensor data to minimize the total amount of data moved through the network.

In this paper, we propose an *energy-efficient algorithm*, which exploits a *divide and conquer* approach for constructing topographic maps in sensor networks. The algorithm creates a distributed, hierarchical storage infrastructure in the network for maintaining topographic metadata in the network. We also demonstrate how the stored information can be used for efficiently resolving some common spatial queries in these networks. Even though such divide and conquer techniques have been used in classical image processing [5], we are not aware of their application in sensor networks. We compare the performance of our algorithm with the centralized approach and the state-of-the-art [4]. Our analytical results show that the performance of our algorithm is significantly better in terms of overall energy and latency. Results from lower level simulations validate our analytical results.

The rest of the paper is organized as follows. We discuss topographic maps in Section II. Section III and Section IV define our system model and the problem to be solved. Our algorithm is presented in Section V and analyzed in Section VI. Section VII illustrates how a sample query can be efficiently resolved using topographic maps. Section VIII describes our simulation framework, the simulated scenarios and experimental results. An overview of related work appears in Section IX. We conclude in Section X.

## II. TOPOGRAPHIC MAP

A *topographic map* is a data structure that represents delineation of features of interest in a terrain. Figure 1 shows a topographic map that represents (graphically) the surface temperature of the sea, where different features (temperature readings) are depicted by different colors. In this paper, we focus on energy efficient construction of a topographic maps representing features of interest in a two dimension terrain based on the sensor readings.

Application areas where topographic maps are particularly useful include contaminant monitoring, tracking soil moisture levels, water temperature estimation of river beds, etc. [7], [8], [9]. A prototype implementation of a sensor network used for

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.

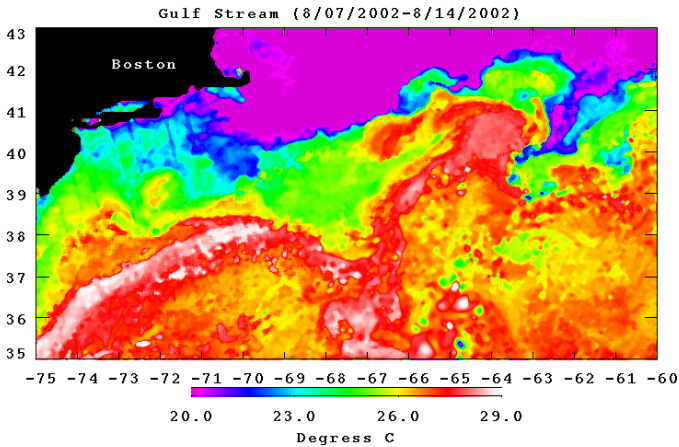


Fig. 1. Topographic map of the sea surface temperature [6]

constructing topographic maps of soil moisture and temperature levels in vineyards is described in [10]. The Sensor Web project at NASA/JPL [11] describes small-scale deployment of sensor networks for understanding spatio-temporal aspects of dynamic phenomenon in habitat monitoring and microsensing applications. Boundary estimation and counting regions of interest can also be used in acoustic monitoring and tracking applications. For example, a spatio-temporal visualization of acoustic readings can be used to determine the number and locations of targets in the region. Maps representing node properties such as residual energy levels are useful for resource management, dynamic retasking, preventive maintenance of sensor networks, etc.

Several queries in sensor networks are targeted towards extracting topological information about features of interest. For example, the end user might be interested in visualizing gradients of sensor readings across the entire terrain; i.e., in periodically estimating the boundary of different regions and the average (or min or max) value of readings in each region. Other useful *topographic queries* of a smaller scope could include enumeration and/or description of regions with sensor readings in a specific range [4] [12]. The above set of queries have the following characteristics. As opposed to locations, they are aimed towards feature regions in the network. Query results cannot be locally computed at individual nodes, but require global collaboration between the feature nodes in the region (e.g., boundary detection of a feature region in the network).

State-of-the-art techniques involving query-initiated explicit data aggregation and exfiltration cannot be used to resolve topographic 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 regions of interest need not trigger any activity other than obtaining the result from a distributed storage node as discussed in Section VII. 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.

### III. SYSTEM MODEL

#### A. System Model

We abstract the sensor network as a distributed system (networked sensor system) consisting of homogeneous, uniquely-identifiable sensor nodes that communicate with each other asynchronously over the locally shared wireless channels. The key assumptions of our model are discussed as follows.

1.  $n$  sensor nodes are uniformly distributed over a two dimensional terrain. The sensor nodes are uniquely identifiable (e.g., by their geographical coordinates). Each sensor node is equipped with a low-power processor, a wireless radio with fixed transmission range  $r$ , a local clock, and several sensing elements.

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. We assume that the graph is symmetric and the sensor nodes that share an edge are called neighbors. The node distribution is sufficiently dense to ensure that the graph  $G$  is connected, and the graph has a small degree. 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 passing. Two types of messages are supported in the system: unicasts and broadcasts. A unicast is a one-to-one message destined to a specific neighbor. A broadcast is a one-to-all message from sender to all its neighbors.

4. The wireless channel is locally shared - a sensor node can receive a message provided only one of its neighbors transmits at a time. A medium access mechanism is supported in the system for resolving contention over the shared channels. When multiple sensor nodes want to access the channel at the same time, the medium access mechanism schedules the communication in conflict-free rounds. All messages are thus reliably transmitted to the receiver nodes in a small (constant) number of rounds.

5. Sensor nodes have two power states: active and switched off. The sensor nodes must be in the active state to perform any computation or communication operation. No energy is dissipated when a sensor node is in the switched off state. The state transition overheads are assumed to be negligible.

6. A low-power paging channel is supported in the system. A sensor node can send a page-all message to switch off or activate all its neighbors. Alternatively, it can also send a page-one message to activate or switch off one specific neighbor. The paging channel is shared between the neighboring nodes. A collision occurs if two or more paging messages are sent over a shared paging channel at the same time. In such scenarios, all sensor nodes on the shared channel receive the message (and detect collision) but are unable to decipher the message contents (type and destination). All sensor nodes that detect a collision are activated.

## B. Time and Energy Analysis

*Time complexity:* As is customary in the analysis of asynchronous distributed systems [13], *for the sake of analysis only*, we use a logical clock. 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 execution of a basic computation operation (such as *add* or *subtract*) on the local processor takes one time step. Transmission or reception of a message between neighboring nodes requires  $T$  time steps. Paging or state transition overheads are assumed to be negligible. The time complexity of an algorithm represents its overall execution time as measured by the logical clock.

*Energy complexity:* We define one unit of energy as the energy dissipated by the local processor in execution of a basic computation operation. A unicast or broadcast message transmitted by a sensor node dissipates  $E$  units of energy at the sender and at each of the receivers. The energy complexity is defined as the sum of the computation and communication energy. It represents the overall energy dissipation in the system on execution of an algorithm.

## C. Rationale

Models similar to the one defined above have been previously used for designing and analyzing algorithms for sensor networks in [14] [15]. However, these models abstract more complex systems that support heterogeneous node architectures [15] and global time synchronization [14].

Two dimensional node deployment is characteristic of several environment monitoring sensor applications [10], [11]. Also, the network is assumed to be connected, i.e., it should be possible to route data from any part of the network to another to facilitate global collaboration.

We assume that a sensor node can use its local clock to synchronize the node-level activities such as local computation, sensing and data transmission. However, no time synchronization is assumed between communication, computation or sensing activities taking place at different sensor nodes in the network. Note this model is in conformance with the TinyGALS [16] programming model, which assumes globally asynchronous and locally synchronous (GALS) communication model.

In state-of-the-art sensor node hardware, communication costs are significantly higher than the computation costs. The communication bandwidth is in the range 20-40 kbps, and transmission energy is in the order of  $1 \mu\text{J}/\text{bit}$ . Also, for short-range radios, the transmission energy is approximately equal to the reception energy [17]. The processor speeds range from 4 MHz to 500 MHz. On the average, execution of a single instruction on the processor takes 10 ns and dissipates 4 nJ of energy [18]. Assuming that one basic computation operation (such as comparison or summation) translates to 10 instructions on average (after compilation), the computation costs can be approximated to 100 ns/operation and 40 nJ/operation respectively. Communication takes place in small, fixed-size packets ranging from 10 to 20 bytes. One unit of

data represents one communication packet. Suppose we define a time step to be 100 ns and one unit of energy to be 40 nJ (time and energy to perform a single computation operation), then based on the above values, we can approximate the energy and time costs for communication as  $T=40000$  and  $E=3000$ .

Several collision avoidance medium access schemes (or MAC protocols) have been discussed in the literature for contention resolution over the wireless channels in the network - e.g., 802.11 and its variants such as S-MAC. The slot reservation and data-ack scheme supported in these protocols can be used for ensuring reliable delivery of unicast messages. To reduce energy overheads of multiple ACKS, the above protocols do not use slot reservations or acks for broadcast messages. However, broadcasts can be implemented as multiple reliable unicasts. Alternatively, reliable communication can also be ensured by using localized TDMA techniques to prevent collisions. The time and energy costs for ensuring reliable broadcasts are larger than unicasts. However, to keep our analysis simple, we assume same communication costs for the two messages.

Sensor nodes dissipate significant amount of energy in the active state receiving packets that are not destined to them. Energy can be saved by switching off sensor nodes for the time duration in which they do not participate in any computation or communication in the network. Power management of sensor states is particularly useful in applications that have a low duty cycle. Sensor nodes can be switched off for majority of the time and activated only when an event of interest takes place. The facility to switch a sensor node from one of multiple modes (such as shutdown, idle, and active) is also commonly available [2] [1]. Wakeup mechanisms that switch a node from shutdown to idle (or active) range from carrier detection circuits to a separate ultra low-power wakeup radios [2]. The paging overheads are significantly lower than communication costs. We do not consider them in our analysis. The costs could be viewed as part of the communication costs, if we assume that a wake-up paging message is transmitted before each data message in the network.

## IV. PROBLEM DEFINITION

### A. Preliminaries

A *feature* is essentially a user specified predicate on sensor data. For example, a feature could correspond to sensor data that satisfies the following property: *temperature greater than forty*. Sensor nodes satisfying a user specified predicate (feature) are called *feature nodes* and the value of the data corresponding to a specific feature is called the *feature value*. All sensor nodes that measure temperature greater than forty are the feature nodes for the above feature and the values of the measured temperature readings are the feature values. In this paper, we assume that the features specified by the user partition the network such that each sensor node satisfies at most one feature.

A *feature region* refers to a geographically contiguous area in the network consisting of sensor nodes that satisfy the same feature and is defined as follows. Consider a partition of the network into a grid consisting of cells of size  $\delta \times \delta$  as illustrated

in Figure 2. Two sensor nodes are said to be *g-connected* (geographically contiguous) if they belong to the same or neighboring (N, S, W, E) cells. The system is represented by a graph  $G_f = (V, E_f)$ . The vertices  $v \in V$  represent the sensor nodes, and an edge  $e(u, v) \in E_f$  exists between two vertices  $u$  and  $v$  iff they are *g-connected* and they satisfy the same feature. Each connected component in  $G_f$  represents a feature region. Here,  $\delta$  is a user-specified value that determines the resolution of the map. Note that we assume that each feature node satisfies at most one feature, which implies that the feature regions are non-overlapping.

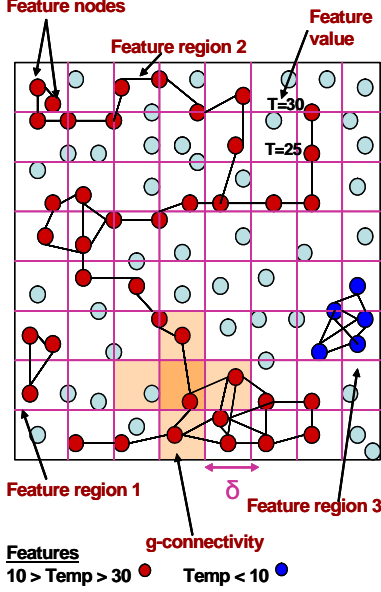


Fig. 2. Feature regions

### B. Problem to be solved

Given a network of  $n$  sensor nodes that store features of interest to the end users, construct the topographic map of the network in a time and energy efficient manner. This involves the following.

1. Given a list of (user-specified) features, identify and label the feature regions in the network.
2. Assignment of region labels to the sensor nodes in a globally consistent manner, which implies the following. All sensor nodes are informed of the termination of the algorithm, and the label of the regions to which they belong. All the sensor nodes belonging to a feature region have the same region label and the sensor nodes in distinct regions have different labels.

## V. OUR ALGORITHM

In recent years, several research efforts have focused on customization of routing protocols to facilitate collaborative computation in sensor networks. Such techniques are suitable for implementing simple data aggregation primitives and localized algorithms that do not maintain any global state (information) in the system. However, for non-trivial computation in sensor systems, the notion of a *stateful algorithm* (as against a

*stateless protocol*) is important to raise the level of abstraction for application developers [19].

This section outlines a distributed, stateful, divide-and-conquer algorithm for construction of topographic map(s) in sensor networks, for the system model defined in the previous section. The latency and energy performance of this algorithm compared to baseline algorithms is analyzed in Sec. VI.

### A. Identification and Labeling of Regions

The key step involved in construction of the topographic map is identification and labeling of regions in the network, which can be abstracted as the widely studied component labeling problem in classical image processing [20] [5]. The algorithm proceeds in the following steps.

**Step I: Cluster formation.** In the first step, the  $n$  sensor nodes partition themselves spatially into single-hop clusters each of size  $c \times c$ . A single node is elected as the *clusterhead* within each cluster, and it is desired that each clusterhead has (at least) 4-connectivity, i.e., has single-hop connectivity with its north, south, east, and west neighbors. To ensure 4-connectivity, each spatial cluster should be of size  $\frac{r}{\sqrt{5}} \times \frac{r}{\sqrt{5}}$  for the following reason. Assuming that any of the nodes within this area can be chosen as clusterhead, the distance between two clusterheads in adjacent clusters could be up to  $c\sqrt{5}$ . Since single-hop connectivity is required,  $c\sqrt{5}$  should be equal to the radio range  $r$ . Hence,  $c = \frac{r}{\sqrt{5}}$ . Let  $1 \leq s \leq n$  denote the number of sensor nodes in each of the clusters. The clusterheads form a mesh topology of size  $\frac{\sqrt{n}}{\sqrt{s}} \times \frac{\sqrt{n}}{\sqrt{s}}$ , as depicted in Fig. 3, which also shows some feature regions present in the network.

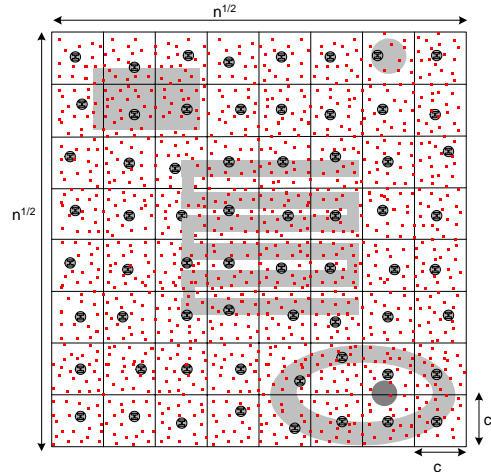


Fig. 3. Cluster formation

At the end of Step I, the state at each node indicates: (i) whether it is a feature node or not a feature node, (ii) whether it is a clusterhead or not a clusterhead, and (iii) if it is not a clusterhead, which of its (uniquely addressable) neighbors is the clusterhead.

**Step II: Intra-cluster identification and labeling of feature regions.** Next, every feature node in the network that is not a clusterhead, transmits its location and reading to its

clusterhead. The local state information at the end of Step I is sufficient for a node to execute this step independently. After the transmission, all nodes that are not clusterheads switch themselves off for a pre-specified duration depending upon the duty cycle of sampling. Only the clusterheads are active at this point.

Based on the information sent to it by feature nodes within its cluster, each clusterhead constructs a local *feature graph*. Vertices of this graph correspond to feature nodes, and an edge exists between two vertices if (i) the corresponding feature nodes are  $g$ -connected, and (ii) the sensor readings denote the same feature.

The clusterhead now runs a depth-first-search algorithm to identify connected components in its local feature graph. Each connected component represents a feature region within the cluster. Boundary estimation and labeling of regions that lie entirely within the cluster can be accomplished at the clusterhead. For regions that cross cluster boundaries, communication with other clusterheads is required to identify their extent.

Now, each local region (connected component) is represented as a single vertex and assigned a unique label  $v = \langle I, i \rangle$ , where  $I$  is the id of the clusterhead, and  $i$  is a locally unique region id. For each vertex  $v$  corresponding to a region with feature  $x$ , the clusterhead maintains  $l_v$ , a list of the feature nodes that lie on the cluster boundary. Each boundary crossing is represented as an outgoing edge denoted by  $\langle v, x, l_v, d \rangle$ . Here  $d \in \{N, S, E, W\}$  identifies the adjacent cluster with which the boundary is shared.

At the end of Step II, only clusterheads are in the active state and all other nodes are switched off. Each clusterhead has a (possibly null) feature graph that represents the feature regions within its  $\frac{r}{\sqrt{5}} \times \frac{r}{\sqrt{5}}$  area.

**Step III: Network-wide distributed graph formation.** Each clusterhead inspects its feature graph and identifies the regions whose feature nodes lie on one or more boundaries of the cluster. The boundary information is transmitted to the corresponding neighbor. This effectively results in an exchange of information between neighboring clusterheads that share a feature region.

For each outgoing edge  $\langle v, x, l_v, d \rangle$ , the clusterhead identifies the corresponding label  $v' = \langle I', i' \rangle$  in the neighboring cluster. If there exists more than one outgoing edge to the same vertex in the neighboring cluster from one or more vertices of the local feature graph, these edges and vertices are reduced to a single vertex since they belong to the same feature region.

At the end of Step III, a distributed graph is constructed in the network. Vertices in the graph that share an edge belong to the same region. Each connected component of the distributed graph represents a region in the network. At this stage, vertices belonging to the same region are not guaranteed to have the same label.

**Step IV: Recursive graph reduction.** Recursive reduction of the distributed graph is performed using a quad-tree approach that can be informally described as follows. In the first step of recursion, the network is conceptually divided into blocks of  $2 \times 2$  clusterheads each. Within each block, one clusterhead

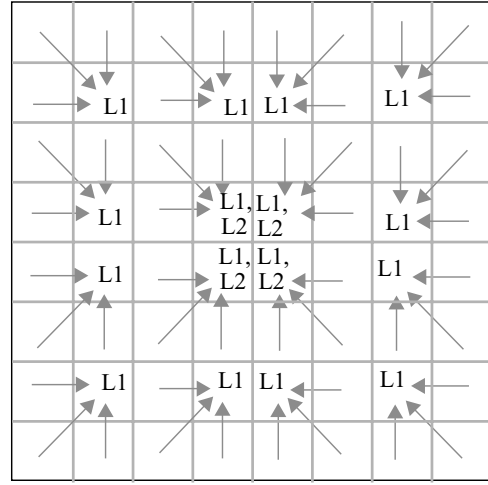


Fig. 4. Leader selection scheme

is the leader and the other 3 clusterheads send information about their edges to the leader, which now computes the connected components within the subgraph, which correspond to the feature regions in its  $\frac{2r}{\sqrt{5}} \times \frac{2r}{\sqrt{5}}$  area of oversight. Each connected component is reduced to a single vertex, and the minimum of all the vertex labels in the connected component is assigned to this new vertex. The set of outgoing edges of the connected component is used to define the edges of the new vertex. If this edge set is empty, it means the vertex represents a feature region. Now, the block leaders organize themselves into  $2 \times 2$  blocks and one of them is elected as a leader for the next level of recursion, and the process is repeated until all components have been identified and labeled. In each stage, *information from exactly 4 clusterheads is merged, which leads to the quad-tree structure.*

More formally, the above process is accomplished in a distributed manner at each clusterhead as follows.

Each clusterhead maintains the current level of recursion  $k$ , initialized to 1. It is also aware of its own id  $I$ , which denotes its position in the mesh. The following routines are also implemented on each clusterhead.

- **REDUCE( $g1, g2, g3, g4, G$ ):** This routine takes 4 sub-graphs ( $g1, g2, g3, g4$ ) as input, and produces a reduced graph  $G$  as the output. The semantics of what the subgraph representations mean and how the reduction is accomplished were described previously.
- **LEADER( $id, rec\text{-}Level$ ):** This routine takes as input a clusterhead id and a level of recursion. Note that the clusterhead id implies a unique block at that level of recursion. This function returns the id of the leader of that block, based on the leader selection scheme being used. The scheme adopted in our algorithm is shown in Fig. 4, and is intended to always move the data to the center of the mesh. Due to space constraints, we do not discuss the exact labeling scheme (a variation of shuffled row major labeling) and the expressions used at each node to compute the next function. In the figure, L1 and L2 denotes leaders of  $2 \times 2$  and  $4 \times 4$  blocks respectively. The arrows show the sources and destinations of data transfers

in the  $2 \times 2$  blocks.

- **SEND(*data*, *id*)**: This routine sends the data item to the clusterhead with the given id.

Every clusterhead performs the following sequence of actions. First, it uses the **LEADER** function with its own identifier as input, and determines the leader id for its block. If the leader id is not the same as its own id (which means it is not a leader), it uses the **SEND** function to send its subgraph to the leader; else, it waits till subgraphs from the three non-leader clusterheads are received, and uses the **REDUCE** function to reduce the four subgraphs (including its own) into a new subgraph *G*. If *G* contains one or more vertices that correspond to regions entirely within the geographic oversight of that block leader, they are not part of the subgraph that is transmitted up the quad-tree. Only the information about cross-boundary regions is communicated at the higher level of recursion. Note that if all regions are within the block boundary, the subgraph that is sent up the quad-tree could be empty. Even in this case, we require the clusterhead to transmit an empty subgraph to the leader of the next higher level to facilitate termination detection of the algorithm.

Next, *k* is incremented by one and the process is repeated till  $k = (\log \frac{\sqrt{n}}{\sqrt{s}} - 1)^1$ . When this happens, the clusterhead sends its subgraph to a uniquely labeled *root* node. In our algorithm, we choose the bottom-right leader at the final recursion level to also act as the *root*. Since every clusterhead knows the values of *n* and *s*, the id of the *root* can be determined.

At the end of Step IV, all regions in the network have been assigned a unique label, which is stored at the leader of the smallest block that contains the entire region.

**Step V: Label propagation through backtracking.** The region labels stored at different clusterheads are propagated down the quad-tree, which is accomplished by backtracking. Since this process is basically an inversion of Step IV above, we do not describe this step in the same detail. When a clusterhead receives new labels for the vertices from the block leader, it updates the locally stored graph. Vertices belonging to different local components are merged if they are assigned the same label. The above steps are repeated till the labels are routed to every clusterhead in the mesh. The clusterhead now activates the feature nodes in turn by using the paging mechanism, and transmits to each feature node the label(s) of the region to which it belongs.

At the end of Step V, each feature node knows the label of the region(s) it belongs to. Each clusterhead stores detailed information about the feature nodes in its cluster.

### B. Distributed Information Storage

In addition of identification and labeling of feature regions, the above algorithm also constructs a *hierarchical storage infrastructure* that can be used maintaining topographic information in the network in a distributed manner. Note that at the end of the labeling algorithm (described above), the clusterheads store detailed information about the feature nodes in their respective cluster. Aggregate topographic information

(such as the average feature value, perimeter, area, and the number of regions contained in the cluster) can be computed by the clusterheads for the feature nodes in the corresponding cluster and transmitted to the block leaders. A clusterhead that is also a leader at a certain level of recursion stores information about the feature regions that are completely contained in its geographic oversight. It follows that the *root* stores aggregate information about all regions in the network, along with the details of feature nodes within its  $c \times c$  cluster.

## VI. ANALYSIS

### A. Distributed Construction of the Topographic Map

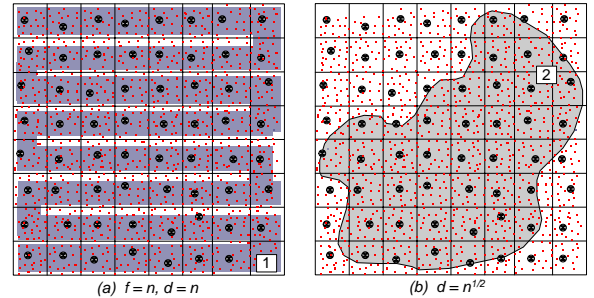


Fig. 5. Sample Regions

We analyze the proposed algorithm as well as the state-of-the-art algorithms for time and energy. To keep the analysis simple, we consider that the network consists of a single feature region. The analysis can be easily extended to multiple regions. We evaluate the performance of the algorithms as a function of the following parameters: *n*, the number of sensor nodes in the network, *s* the number of nodes in a single hop cluster (see Section V), *f* the number of feature nodes in the region, and *d* the diameter of the region. The *diameter* is defined as the number of feature nodes on the longest feature chain in the region. A feature chain represents the shortest sequence of *g*-connected feature nodes belonging to a region that connect any two feature nodes in the same region. Figure 5 illustrates regions with diameter *n* and  $\sqrt{n}$  respectively.

As discussed in Section III, we assume a single computation operation requires one time step and dissipates one unit of energy. Transmission or reception of a unit of data (over a single hop) takes *T* time steps and dissipates *E* units of energy. Each unicast (one-to-one) message dissipates  $2E$  units of energy (for transmission and reception). A transmitted broadcast message is received by all  $5s$  neighboring nodes (*s* denotes the number of sensor nodes in a cell of size  $\frac{r}{\sqrt{5}} \times \frac{r}{\sqrt{5}}$ ) and thus dissipates  $((1+5s) \cdot E)$  units of energy. The above can be implemented by an efficient paging scheme such that sensor nodes are only active when they compute or communicate, and are switched-off otherwise.

#### a.) Proposed Algorithm

*Communication time and energy*: No communication takes place in Step I of the proposed algorithm. In Step II, each of the *f* feature nodes contends for the wireless channel and on gaining access, transmits one unit of data to the clusterhead. This involves *f* one-to-one transmissions resulting in overall

<sup>1</sup>All logs are to the base 2 unless specified otherwise.

energy dissipation of  $(2f \cdot E)$  units. Any clusterhead in the network has  $5s$  neighbors:  $s$  within its cluster and  $4s$  in the four (N, S, W, E) adjacent clusters. In the worst case, all the neighbors are feature nodes and  $(5s \cdot T)$  time steps are required to schedule the data communication between the feature nodes and the clusterheads.

Step III of the algorithm requires each clusterhead to exchange boundary information with its neighbors. In the worst case scenario, all feature nodes are on cluster boundaries and thus  $f$  units of data are exchanged between the clusterheads. This results in  $f$  transmissions, dissipating  $(2f \cdot E)$  units of energy in the system. A clusterhead transmits and receives no more than  $\min(f, 4\sqrt{s})$  units of data. There are at most 9 clusterheads in an area of  $r^2$  that may contend for the shared channel at any given time. The above communication can be scheduled in fewer than  $(\min(9f, 36\sqrt{s}) \cdot T)$  time steps.

Step IV consists of at most  $(\log \frac{\sqrt{n}}{\sqrt{s}} - 1)$  recursive steps. We consider communication in the  $k^{th}$  recursive step. Consider a block of size  $2^k \times 2^k$ . It consists of four smaller blocks (we call them sub blocks) of size  $2^{k-1} \times 2^{k-1}$ . The leader of one of these sub blocks is assigned the role of the block leader. The leaders of the other 3 sub blocks communicate information about their subgraph to the block leader (of the larger block). A sub block leader only transmits information about the vertices that have edges crossing the sub block boundaries. There are at most  $(4 \cdot 2^{k-1} \cdot \sqrt{s})$  such edges (and vertices). Note the sub block leader of one of the blocks is the block leader itself. Each of the other three sub block leaders is less than  $2^k$  hops away from the block leader. Thus, there are at most  $(12 \cdot 2^{k-1} \cdot \sqrt{s})$  data units in a block that must be communicated over  $2^k$  hops. There are  $(\frac{n}{s} \cdot \frac{1}{2^{2k}})$  such blocks in the network. The communication can take place in parallel in the various blocks. Efficient in-block routing can avoid contention over shared channels (we do not discuss these in this paper). Thus, the  $k^{th}$  recursive step can be completed in  $(12 \cdot 2^{k-1} \cdot \sqrt{s} \cdot T)$  time steps. In the worst case (when the region spans over the entire network),  $((\frac{n}{s} \cdot \frac{1}{2^{2k}}) \cdot (2 \cdot 12 \cdot 2^{k-1} \cdot \sqrt{s}) \cdot 2^k \cdot E = \frac{12n}{\sqrt{s}} \cdot E)$  units of energy is dissipated in the  $k^{th}$  recursive step. Since there are  $\log \frac{\sqrt{n}}{\sqrt{s}} - 1$  recursive steps, Step IV requires  $(6\sqrt{n} \cdot T)$  time steps and dissipates  $(\frac{12n}{\sqrt{s}} \cdot (\log \frac{\sqrt{n}}{\sqrt{s}} - 1) \cdot E)$  units of energy in the worst case.

The above analysis assumes the worst case scenario (region consisting of  $n^2$  feature nodes). Consider a region with diameter  $d$  consisting of  $f < n$  feature nodes. Step IV may be completed in fewer recursive steps if the region is contained in a smaller sized block ( $\frac{d}{\sqrt{s}} \cdot T$  time steps in the best case, when the region lies in a block of size  $d \times d$ ). When the region is located in the center of the network,  $\log \frac{\sqrt{n}}{\sqrt{s}} - 1$  recursive steps are required as in the worst case. However, energy dissipation is lesser for a smaller region. In a region with  $f$  feature nodes, in the worst case, all information is routed over  $\frac{\sqrt{n}}{\sqrt{s}}$  hops in the network since the manner we elect leaders ensures that information flow is towards the center. The overall energy dissipation would be  $O(f \cdot \frac{\sqrt{n}}{\sqrt{s}} \cdot E)$  in the worst scenario. Thus, the overall energy dissipation in this step is  $\min(O((\frac{n}{\sqrt{s}} \cdot \log \frac{\sqrt{n}}{\sqrt{s}}) \cdot E), O(f \cdot \frac{\sqrt{n}}{\sqrt{s}} \cdot E))$ .

In Step V, the labels are transmitted to the sensor nodes by

a backtracking procedure, and we do not analyze it in detail. However, it is easy to see that the time and energy of this step is not larger than steps I-IV combined.

Thus, overall energy dissipation for the algorithm is  $O((f + \min(O(\frac{n}{\sqrt{s}} \cdot \log \frac{\sqrt{n}}{\sqrt{s}}), O(f \cdot \frac{\sqrt{n}}{\sqrt{s}}))) \cdot E)$  units and it requires  $O((\sqrt{n} + s) \cdot T)$  time steps.

*Computation time and energy:* The proposed algorithm involves significant in-network computation. In Step I, each sensor node locally determines whether it is a feature node based on the sampled data. Based on its ID it also determines the ID of the clusterhead. The above requires  $O(1)$  computation operations at all the sensor nodes in the network. This can be accomplished in  $O(1)$  time steps with  $O(n)$  units of energy dissipation.

In Step II each clusterhead constructs the local feature graph and executes a depth-first-search (dfs) on the graph to identify and label components. Each subgraph has  $O(s)$  vertices and  $O(s^2)$  edges.  $O(s^2)$  operations are performed for the dfs at each node. Moreover, the clusterhead also computes metadata for each component identified. This requires  $O(s)$  computation operations. There are  $\frac{n}{s}$  clusterheads. Step II can be performed in  $O(s^2)$  time steps with  $O(n \cdot s)$  units of energy dissipation.

In Step III the clusterheads exchange and merge the boundary information. In order to merge the boundary information, each clusterhead must compare all elements in the list of outgoing edges with the lists received from the neighbors. There are at most  $4\sqrt{s}$  outgoing edges for each cluster. Thus this step requires  $O(s)$  comparison operations. This step can be executed in  $O(s)$  time steps with  $O(n)$  units of energy dissipation.

Step IV consists of  $\log \frac{\sqrt{n}}{\sqrt{s}} - 1$  recursive steps. At the  $k^{th}$  recursive step for each block of size  $2^k \times 2^k$ , the leader performs the following computation (a) merges boundary information of its four sub blocks, (b) performs dfs on the resultant subgraph to identify local components, and (c) computes metadata for each component.

There are  $(\frac{n}{s} \cdot \frac{1}{2^{2k}})$  such blocks in the network. There are at most  $4 \cdot 2^{k-1}$  outgoing edges in each sub block. Thus the subgraph formed at the leader of the block has at most  $O(2^k)$  edges (and fewer or equal number of vertices). Thus merging of boundary information can be merged in  $O(2^{2k})$  computation operations and dfs can be performed in  $O(2^k)$  computation operations. The  $k^{th}$  recursive step can be computed in  $O(2^{2k})$  time steps with energy dissipation of  $O(\frac{n}{s})$  energy units. Thus, Step IV can be performed in  $O(n)$  time steps with  $O(\frac{n}{s} \log \frac{\sqrt{n}}{\sqrt{s}})$  units of energy dissipation.

Step V involves no computation.

The overall computation time for this algorithm is  $O(s^2 + n)$  and energy dissipation is  $O(n \cdot s + \frac{n}{s} \log \frac{\sqrt{n}}{\sqrt{s}})$  units.

### b.) Baseline Algorithms

*Centralized Approach:* This approach involves routing of information (such as feature and ID) from all feature nodes to a centralized node (or base station). All processing takes place at the centralized node. Once, the feature regions have been identified and labeled, the labels are transmitted by the

centralized node to all feature nodes in the network. Each of the feature nodes transmits one unit of data to the centralized node. Consider the scenario, when feature nodes are at the boundary of the network. In this scenario, each data unit must traverse at least  $\frac{\sqrt{n}}{2\sqrt{s}}$  hops to the centralized node.

Total number of transmissions in the network is  $\frac{f \cdot \sqrt{n}}{2\sqrt{s}}$ . If we assume perfect scheduling (no delay incurred in the network due to congestion), at least  $\frac{\sqrt{n}}{2\sqrt{s}} \cdot T$  time steps are required for communication of data from the feature nodes to the centralized node. A sensor node can receive only one data unit at a time. At least  $f \cdot T$  time steps are required by the centralized node to receive  $f$  units of data. Thus, overall time complexity of the algorithm is  $O((f + \frac{\sqrt{n}}{\sqrt{s}}) \cdot T)$ . An efficient paging scheme would switch off redundant sensor nodes to ensure that each transmission is received by exactly one sensor node. In such a scenario, the overall energy dissipation in the network is twice the transmission energy, which is  $O((\frac{f \cdot \sqrt{n}}{\sqrt{s}}) \cdot E)$ .

*Decentralized, Self-organizing Algorithm:* This approach has been discussed in [4]. Only feature nodes participate in the algorithm. Every node maintains the variables  $m_{id}$  and  $m_h$  that represent the ID of the lowest ID sensor node in the network and the minimum number of hops to reach this node. At the start of the algorithm, each sensor node initializes  $m_{id}$  to its own label and  $m_h$  to infinity. All feature nodes broadcast their ID with hop initialized to one. When a feature node receives a broadcast, it compares the received value to the stored  $m_{id}$ . If the value is smaller, the node updates the variable  $m_{id}$ , increments and stores the hop count, and broadcasts a message containing the  $m_{id}$  and the hop count incremented by one. The node sets as parent the node from which it received the broadcast. The above step is also repeated if the  $m_{id}$  received is equal to the stored value but the hop count is smaller. The system stabilizes when all feature nodes have the minimum value of  $m_{id}$  and the shortest path to the feature node with  $ID = m_{id}$ . This results in a spanning tree with the minimum ID node acting as the root. The ID of the root can be used to label the feature region. All feature nodes transmit the feature information up the tree to the root node. The root node collects data from all the feature nodes. The topographic metadata and boundary information about the feature region is maintained at the root node.

As opposed to the proposed algorithm that uses unicast messages, all messages in the above algorithm are broadcasts. Since each transmitted message must be received by all the neighbors, all sensor nodes must be active for the entire duration of the labeling phase. Paging techniques cannot be used to switch off sensor nodes to reduce energy dissipation. Although the broadcast model (reliable or unreliable) assumed for the above algorithm is not explicitly outlined in [4], a reliable broadcast scheme seems to be an essential requirement for the algorithm to function correctly. Therefore, it cannot be implemented using state-of-the-art medium access protocols (such as the 802.11) that do not support reliable broadcasts due to the large overheads involved.

Next we analyse the energy and time complexity of the algorithm. Consider a feature region that consists of a single

TABLE I  
ENERGY DISSIPATION

Algo	$f = O(n)$ $d = O(n)$	$f = O(n)$ $d = O(\sqrt{n})$	$f = O(\sqrt{n})$ $d = O(\sqrt{n})$
Central.	$O(n^{3/2} \cdot E)$	$O(n^{3/2} \cdot E)$	$O(n \cdot E)$
Decentral.	$O(n^2 \cdot E)$	$O(n^{3/2} \cdot E)$	$O(n \cdot E)$
Proposed	$O(n \cdot \log n \cdot E)$	$O(n \cdot \log n \cdot E)$	$O(n \cdot E)$

TABLE II  
TIME PERFORMANCE

Algo	$f = O(n)$ $d = O(n)$	$f = O(n)$ $d = O(\sqrt{n})$	$f = O(\sqrt{n})$ $d = O(\sqrt{n})$
Central.	$O(n \cdot T)$	$O(n \cdot T)$	$O(\sqrt{n} \cdot T)$
Decentral.	$O(n \cdot T)$	$O(\sqrt{n} \cdot T)$	$O(\sqrt{n} \cdot T)$
Proposed	$O(\sqrt{n} \cdot T)$	$O(\sqrt{n} \cdot T)$	$O(\sqrt{n} \cdot T)$

sequence of feature nodes (chain) of size  $l$ . The sequence is monotonically increasing in terms of the node IDs. For  $1 \leq k \leq l$ , let  $f_k$  denote the sensor node in the  $k^{th}$  position in the sequence. In the worst scenario (when the channel access scheme gives priority to higher ID node),  $f_k$  receives  $k - 1$  labels and transmits  $k$  labels.  $O(l)$  time steps are required to route the label of the min ID node to all feature nodes, and  $l^2$  transmissions take place. In the best scenario  $f_k$  node receives and transmits  $\frac{(k-1)}{\sqrt{s}}$  labels. This reduces the time to  $O(\frac{l}{\sqrt{s}} \cdot T)$  time steps and energy to  $O(l \cdot \frac{l}{\sqrt{s}} \cdot E) = O(\frac{l^2}{\sqrt{s}} \cdot E)$  energy units. Next, we consider a region with diameter  $d$  and  $f$  feature nodes such that all feature nodes lie on monotonically increasing chains of length  $d$ . As discussed above routing of the min ID label on a chain requires  $O(\frac{d}{\sqrt{s}} \cdot T)$  time steps and  $O(\frac{d}{\sqrt{s}})$  transmissions. There are  $\frac{f}{d}$  chains in the network, and thus the total number of data transmissions is  $O(\frac{f \cdot d}{\sqrt{s}})$ . Each transmission is received by  $O(s)$  neighboring nodes, resulting in energy dissipation of  $O(f \cdot d \cdot \sqrt{s} \cdot E)$  energy units. Thus overall time complexity of this algorithm is  $O(\frac{d}{\sqrt{s}} \cdot T)$  and energy complexity is  $O(f \cdot d \cdot \sqrt{s} \cdot E)$  energy units.

A major drawback of this algorithm is that sensor nodes cannot detect the termination of the labeling phase (stabilization of paths to the root) in a consistent manner. A sensor node on a shorter route to the root might stabilize much before the most distant sensor node. An additional termination detection phase must be included in the algorithm after the labeling phase. One heuristic could involve introducing a wait state after the labeling phase. Once a sensor node detects local stabilization it waits for time  $O(d)$  to ensure global stabilization in the system. Since the value of  $d$  is not known and  $d = n$  in the worst case, termination detection will incur an additional overheads of  $O(n \cdot T)$  time steps.

Our proposed algorithm performs significant in-network computation. However, in state-of-the-art sensor systems the communication costs are significantly large than computation costs. Figure 6 compares the computation and communication costs of our algorithm. The computation overheads are negli-

gible. The communication energy and time costs for the above algorithms (for  $s = 1$ ) are summarized in Table I and Table II respectively.

We observe that the energy and time performance of our proposed algorithm is significantly better for large regions and comparable for smaller regions. The centralized approach performs better than the decentralized algorithm but maintains no topographic information in the network. One of the goals of constructing a topographic map is maintain distributed information in the network. The next section discusses resolution of topographic queries in the network based on the in-network information maintained by the two algorithms (decentralized and ours).

## VII. TOPOGRAPHIC QUERYING

For the sake of illustration, we discuss efficient resolution of the following query in the network. *Count the number of regions in the network for a specific feature.* For example, the end-user may be interested in counting the total number of regions in the network that have temperature greater than thirty. The above query is representative of the class of topographic queries that involve global aggregation of topographic information in the network. For example, the total area, perimeter or average feature value of regions in the network that represent a feature of interest to the end user.

**Our approach:** The query node routes the query to its clusterhead, which routes it to the block leader. On receiving the query, the block leader routes it to the next level leader. This step is repeated until the query reaches the root. The root maintains a list of the regions in the network, the feature represented by each region, and the aggregated information about its attributes (perimeter, area, etc.). A count is obtained for the regions that represent the feature of interest to the end-user, and the response is routed to the sensor node where the query originated (*query node*). The query can be routed to the root in  $\frac{\sqrt{n}}{\sqrt{s}}$  hops. Similarly, the response can be sent to the query node in  $\frac{\sqrt{n}}{\sqrt{s}}$  hops. Thus, the query can be resolved in  $(\frac{2\sqrt{n}}{\sqrt{s}} \cdot T)$  time steps with  $(\frac{4\sqrt{n}}{\sqrt{s}} \cdot E)$  units of energy dissipation.

**Decentralized approach:** The decentralized approach maintains topographic metadata about feature regions at the root nodes. No global information is maintained about the location or number of root nodes in the network, and thus, the query must be flooded in the entire network. On receiving the query, a root node checks its feature value to see if it satisfies the query. If so, it routes a response to the query node, which computes the count. In the best scenario, time taken by the query to reach all sensor nodes is  $(\frac{\sqrt{n}}{\sqrt{s}} \cdot T)$ . Moreover, since each node receives at least one transmission, overall energy dissipation is at least  $(n \cdot E)$ . Let  $C$  denote the regions in the network that respond to the query. A response can be routed back to the query node in  $(\frac{\sqrt{n}}{\sqrt{s}} \cdot T)$  time steps. Since the query node can receive one unit of data in one timestep, the total time taken is  $(\max(\frac{\sqrt{n}}{\sqrt{s}}, C) \cdot T)$ . Energy dissipation is  $(2C \cdot \frac{\sqrt{n}}{\sqrt{s}} \cdot E)$  units. Thus, the total time taken to resolve the query is at least  $(\max(\frac{\sqrt{n}}{\sqrt{s}}, C) \cdot T)$  and

energy dissipation is  $(n + 2C \cdot (\frac{\sqrt{n}}{\sqrt{s}}) \cdot E)$  units. Note that for  $C = O(n)$ , total execution time is  $O(n \cdot T)$  time steps and overall energy dissipation is  $O(\frac{n^{3/2}}{\sqrt{s}} \cdot E)$  units.

## VIII. SIMULATION FRAMEWORK AND RESULTS

### A. Our Simulator

A high-level simulator was developed to verify the functionality of the algorithms described in the previous section, and to evaluate the time and energy performance of our algorithms. This simulator is high level because it abstracts the physical layer and medium access in terms of pre-specified models, and does not perform network-level simulation. The simulator consists of the *pattern generator module*, the *computation module* and the *network simulation engine* which are discussed as follows.

1) **Pattern Generator Module:** The pattern generator can generate the following two types of patterns, by labeling the appropriate subset of nodes in the network as feature nodes. The user is expected to supply the relevant parameters for the patterns.

- *Snake:* This pattern represents a single region that passes through every cluster in the network (see region with label 1 in Fig. 5(a)). This pattern is of interest because it is the *worst-case scenario for decentralized flooding-based algorithms* for labeling regions of interest. The diameter of the pattern is  $O(n)$ . No input parameters are required or accepted for this pattern.
- *Blob:* This pattern consists of disjoint equal-sized rectangular regions (blobs) uniformly dispersed throughout the network. The input parameters specify the diameter of each blob, the number of blobs, and the separation between the blob boundaries.

2) **Computation Module:** The computation module implements our algorithm in a centralized, synchronous manner. The output of the pattern generator is used as input to this module. The module computes on the data and for each step of our algorithm, it outputs the input and output data sets for the sensor nodes into a log file. The log file thus generated, is used as an input to the network simulation engine for simulating communication in the network.

JouleTrack [18], a web-based energy and time profiling tool was used to evaluate the computation time and energy for our algorithm. To minimize energy dissipation, simulations were performed at the slowest setting of the processor speed (59MHz) supported by the tool.

3) **Network Simulation Engine:** The network simulation engine simulates the communication in the network. Its key features are discussed as follows.

1. *Network topology:* The engine can be configured to generate a user-specified node density and distribution (grid, random, uniformly-deployed, etc.). In our simulations, we assumed a uniform topology with unit density. Each unit area cell consisted of one sensor node.

2. *Local information:* Each node in the network maintains the following local information: its own coordinates, its current

power state (active or switched off), next activation time if it is switched off, computation stage of the implemented algorithm and the cumulative energy dissipation till the current simulation time. Each node also maintains a communication buffer for outgoing packets. At the start of the algorithm, the output of the pattern generation module is used to initialize the communication buffers for the sensor nodes.

3. *Global information:* A sensor node also has access to the following global information maintained by the simulation engine: the total number of nodes in the network, the radio range of the sensor nodes and the current simulation time.

4. *Channel access:* A carrier sense multiple access protocol for the wireless channel is simulated at a high level in the following way. At every simulation cycle, all active sensor nodes that are ready to transmit are identified. Subsets of these nodes are determined such that in each subset, more than one transmission will cause a collision at some receiver. Only one node in each subset is elected as the winner and allowed to transmit in that cycle, and the other nodes must compete in the future cycles.

5. *Communication:* The communication in the network is performed in a synchronous manner. The simulations are performed in cycles using fixed-size packets of 10 bytes. Each cycle represents one time step (as defined earlier). At each cycle, the simulator checks for all sensor nodes that have packets to transmit. A subset of the nodes are identified as transmitters based on the channel access mechanism described below. Next, the set of receivers is identified for each of the transmitting node. A sensor node is a receiver to a transmitter if it lies within its radio range and is in the active state. The transmit handler function is invoked for the transmitting node. The receive handler function is invoked for each of the receivers with the top packet in the communication buffer of the transmitter as its argument. Next, the simulator invokes the update function for each sensor node. The functions are described as follows. The *transmit handler function* deletes the top most packet in the communication buffer. It increments the energy dissipation of the transmitting node by one unit. The *receive handler function* also increments the energy dissipation by one unit. It may perform one or more of the following operations: update its data variables, rebroadcast (or consume) the packet, or generate a new packet (discussed in more detail in the following subsection). The *update function* checks whether the node must be activated or switched off depending on the current simulation time.

6. *Communication time and energy:* On termination, the total number of simulation cycles represent the communication time for the algorithm. Total energy dissipation is obtained by summing the energy dissipation at all the sensor nodes.

## B. Simulation Details

As discussed earlier, the centralized algorithm is not of interest to us as it does not maintain any topographic information in the network. Simulations were performed to compare the performance of the decentralized, self-organizing

algorithm with our proposed algorithm. The simulation details are discussed below.

1) *Decentralized algorithm:* At initialization, a packet is placed in the message buffer for each feature node in the network. Each packet contains the label of the node and the minID seen so far, its hop distance and the id of the sender node. At each iteration, sensor nodes with nonempty message buffers try to access the channel to transmit a packet. On receiving a packet, the id of originator and hop distance is compared. If both are smaller, the variables are updated at the receiver node and the packet is rebroadcasted in the network, and else the packet is consumed. No routing protocol is required for communication. The simulation engine monitors the size of the message buffers at the sensor nodes. Empty buffers indicate that no more packets are en route and the spanning tree routed at the min id node has been constructed. Note, that in a real network inavailability of such global information may make it difficult to determine termination of the tree construction phase. Next, the feature nodes route packets containing the sensed feature to the root over the spanning tree constructed in the region.

2) *Clustered, decentralized algorithm:* In this algorithm, the individual feature node is active for only one cycle - when it transmits its location and reading to the clusterhead. Further exchange of information, formation of the spanning tree, and labeling of regions occurs at the clusterhead level (other sensor nodes are switched off), not the feature node level. Therefore, we simulate the above algorithm in the network at a clusterhead granularity, i.e., each node in the simulated network represents a clusterhead. The network is thus simulated as a grid of clusterheads. The communication costs of packet transmission from feature node to the clusterhead (and vice versa) is added to the costs obtained by the clusterhead-level simulation to get the total costs.

3) *Our Proposed Algorithm:* Similarly, the proposed algorithm is also simulated at the clusterhead level. As before, the overall costs are obtained by adding the feature node to clusterhead communication costs to the costs estimated by the simulator in executing the proposed algorithm over the clusterheads. The object which represents the clusterhead itself is initialized with its grid coordinates, a neighbor list, and a fixed-size buffer. Each clusterhead is initialized with a list of feature nodes in its cluster and their position. Each clusterhead reads the log file (created by the computation module) to create a list of output data set and input data set for each recursive step of the algorithm. When a sensor node receives a packet it performs the following operation. If the packet is not destined to itself, it rebroadcasts it to the adjacent clusterhead on the shortest path to the packet destination. If the packet is for itself, it looks the variable storing count of the current recursive step and checks the input data list for this step. If the list not empty, it deletes the element on the list that corresponds to the received data reads the input data list for the next step. If the data received is contained in the list, the corresponding element is deleted. On detecting an empty list, the sensor node reads the output data set for the next recursive step, creates a packet and inserts it into the communication buffer. The algorithm terminates when the input and output data lists have

been consumed.

### C. Simulation Results

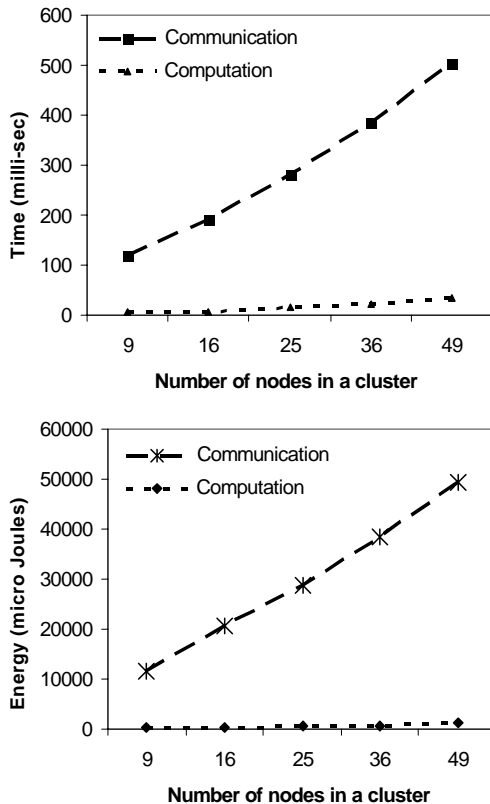


Fig. 6. Computation and communication costs of the proposed algorithm

a) *Computation costs for the proposed algorithm:* Figure 6 compares the computation and communication costs for our algorithm. We assumed a network consisting of  $16 \times 16$  clusters. The time and energy dissipation for communication and computation was evaluated as a function of the number of nodes within each cluster (node density) in the network. We observe that the computation time and energy is negligible in comparison with the communication time and energy. For higher node densities, the communication costs increase more rapidly than the computation costs.

b) *Communication Costs:* We simulate networks with size ranging from  $50 \times 50$  to  $500 \times 500$  sensor nodes. Short range communication is assumed between the sensors. We consider radio range to be between 3 to 6 units. As a consequence, the size of single hop clusters is limited from 9 to 36 nodes.

The clustered decentralized algorithm executes at the clusterhead granularity rather than node granularity. Since the number of participating nodes is decreased, contention in the network is reduced. The main benefit of clustering is in conserving reception energy. In the algorithm in [4], all sensor nodes are active. Each transmission is received by  $r^2$  neighboring sensor nodes. In the clustered version, only four neighboring clusterheads receive the packet. Energy is reduced by a factor of  $r^2$  as illustrated in Figure ?? and ?. Therefore,

we primarily focus on performance evaluation of the clustered decentralized and the proposed algorithm.

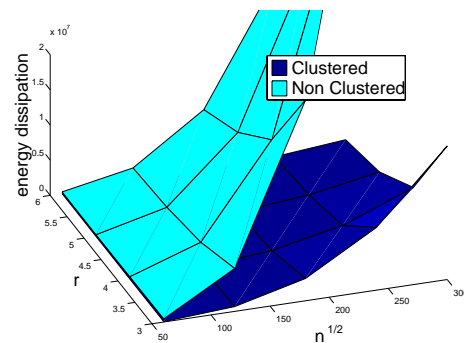


Fig. 7. Decentralized Algorithm: Energy Consumption (Snake pattern)

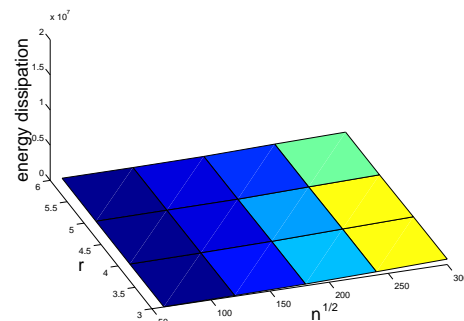


Fig. 8. Proposed Algorithm: Energy Consumption (Snake pattern)

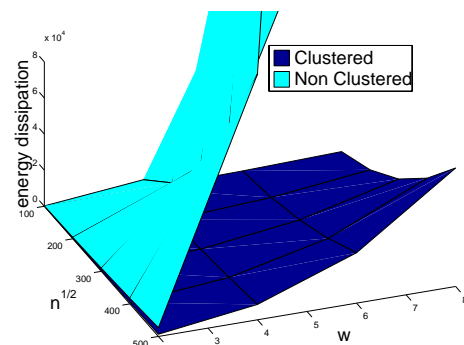


Fig. 9. Decentralized Algorithm: Energy Consumption (Blob pattern)

Fig. 7, 8, 11, and 12 illustrate the energy dissipation and the time performance of decentralized and the proposed algorithms as a function of network diameter  $n$  and radio range  $r$  for the snake pattern. The two algorithms have similar performance for smaller values of  $n$ . However, the performance of the proposed approach is superior to the decentralized approach. This is because the diameter of the snake pattern is proportional to the network size. The average hop distance traveled by a packet using the decentralized algorithm is proportional to the pattern diameter, which is  $O(n)$  for the snake pattern. For the proposed approach no packet makes more than  $\sqrt{n}$  hops resulting in lower energy dissipation and smaller execution time. For instance, for  $\sqrt{n} = 120$  and

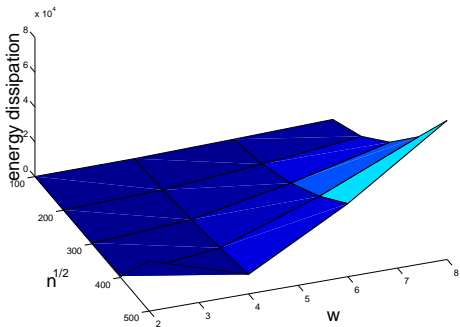


Fig. 10. Proposed Algorithm: Energy Consumption (Blob pattern)

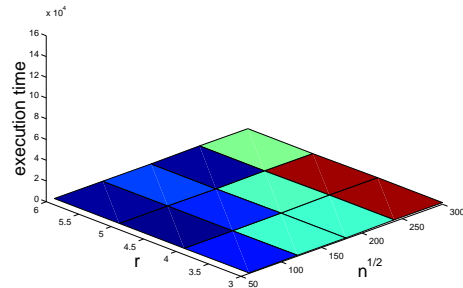


Fig. 12. Proposed Algorithm: Latency (Snake pattern)

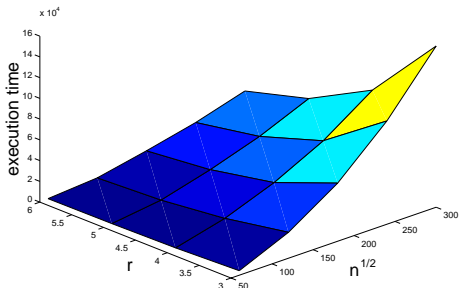


Fig. 11. Decentralized Algorithm: Latency (Snake pattern)

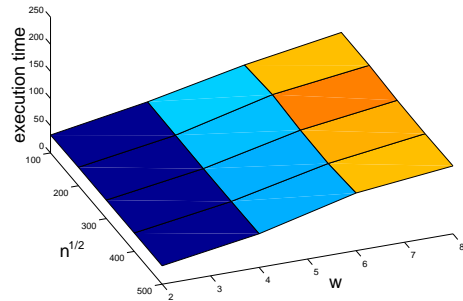


Fig. 13. Decentralized Algorithm: Latency (Blob pattern)

$r = 4$  energy dissipation is 233263 and execution time is 14052 units using the decentralized algorithm. For the same parameters energy dissipation is 81624 and execution time is 469 units using the proposed algorithm. The energy dissipation and time performance is reduced with increasing value of  $r$ . Larger value of  $r$  results in reducing the network diameter and reduces the number of clusterheads participating in the algorithms. Note we assume that a uniform cost model can be used for short range communication. The results may show a different trend for very large values of  $r$ , for which the uniform cost model cannot be used for analysis.

Fig. 9, 10, 13, and 14 show simulation results for the decentralized and the proposed algorithms for the blob pattern. We plot overall energy dissipation and execution time as a function of the network diameter  $\sqrt{n}$  and blob diameter  $w$ . The separation between blobs is varied to keep the total number of regions equal for a fixed value of  $n$  for different values of  $w$ . This pattern illustrates the worst case performance of the proposed approach. For small size regions, the decentralized approach stabilizes rapidly with very few transmissions. The performance of the proposed algorithm depends on the location of the regions. A region in the center of the network spans inter cluster boundaries for all levels of recursion. This results in higher energy dissipation and latency in the system, but the performance is not much worse. For example, when  $\sqrt{n} = 200$  and  $w = 8$ , the decentralized algorithm stabilizes in 129 time units with overall energy dissipation of 10539 units. The proposed algorithm terminates in 213 units of time with overall energy dissipation of 12800 units.

## IX. RELATED WORK

Feature extraction and maintenance is a problem of interest to the wireless networked sensing community, because it finds application in design of a large number of sensor systems (see Sec. II). A fault tolerant, decentralized protocol for feature extraction was proposed in [4], and described and analyzed in Sec. V and Sec. VI. Our simulations demonstrate that our recursive clustered algorithm is more efficient than a non-recursive clustered algorithm based on [4]. High level models have been employed recently to design algorithms for sorting, list ranking, initialization, etc., in wireless sensor networks [21], [15]. Modeling sensor nodes as pixels and applying image processing techniques for different problems in networked sensing is an emerging area of research. [22] presents techniques for cleaning uncorrelated sensor noise and localized edge detection. Boundary estimation has been studied from an information theoretic perspective in [12]. Concerning the larger problem of topographic querying as discussed in this paper, the DIMENSIONS [3] data handling architecture is a general infrastructure that incorporates multi-resolution data access and spatio-temporal pattern mining. A data handling architecture such as DIMENSIONS will complement efficient algorithms designed specifically for topographic querying.

## X. CONCLUDING REMARKS

In this paper, we presented an energy-efficient divide-and-conquer algorithm for identification and labeling of homogeneous regions in a sensor network. This information is stored in a distributed, hierarchical fashion in the network, and can

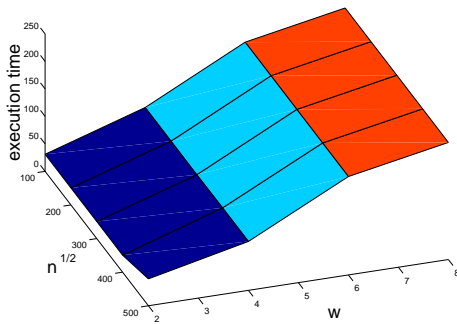


Fig. 14. Proposed Algorithm: Latency (Blob pattern)

be used to synthesize responses to a large class of topographic queries. We defined a high-level system model that abstracted the low level details of the network, thereby allowing us to design and analyze our algorithm largely independent of the specific network protocols and hardware details. For a class of patterns, analytical inferences were borne out by lower level simulation results.

Most algorithms for sensor network applications are being designed using low-level models that expose many details of the underlying network. These models are useful because the algorithm performance can be accurately estimated, and the algorithm specification can be easily synthesized for real sensor node hardware. However, for an end user who might not be an expert in (or concerned with) the details of sensor node hardware and networking, low level models do not provide a suitable abstraction for algorithm design. In this paper, we use a high-level model – with assumptions about computation and communication costs, topology, etc. – in order to demonstrate how such a model can be used for rapid design, first-order analysis, and comparative evaluation of alternate algorithms for the same functionality. High level system modeling, design, and analysis is not a substitute for detailed, low level simulation. However, a well-defined model will hide many details not directly relevant to the problem at hand and make application design and optimization more tractable for the end user.

## REFERENCES

- [1] M. Horton, D. Culler, K. Pister, J. Hill, R. Szweczyk, and A. Woo, "MICA: The commercialization of microsensor motes," *Sensors Magazine*, <http://www.sensormag.com/>, April 2002.
- [2] "The Power Aware Sensing, Tracking and Analysis (PASTA) Project (PACC/ Phase II)," <http://pasta.east.isi.edu/welcome/>.
- [3] D. Ganesan, D. Estrin, and J. Heidemann, "DIMENSIONS: Why do we need a new data handling architecture for sensor networks?," in *Workshop on Hot Topics in Networks*, October 2002.
- [4] B. Krishnamachari and S. S. Iyengar, "Efficient and fault-tolerant feature extraction in sensor networks," in *2nd Workshop on Information Processing in Sensor Networks*, April 2003.
- [5] H. M. Alnuweiri and V. K. Prasanna, "Parallel architectures and algorithms for image component labeling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14(10), pp. 1014–1034, 1992.
- [6] "Physical Oceanography Distributed Active Archive Center (PO-DAAC)," <http://podaac.jpl.nasa.gov>.
- [7] "Contaminant transport monitoring," <http://cens.ucla.edu/Research/Applications/ctm.htm>.
- [8] "The klamath river flow studies," [http://www.krisweb.com/krisweb\\_kt/klamflow.htm](http://www.krisweb.com/krisweb_kt/klamflow.htm).

- [9] "Global soil moisture data bank," [http://climate.envsci.rutgers.edu/soil moisture](http://climate.envsci.rutgers.edu/soil_moisture).
- [10] T. Brooke J. Burrell and T. Beckwith, "Vineyard computing: sensor networks in agricultural production," in *Pervasive Computing Magazine, IEEE*, March 2004, pp. 38–45.
- [11] "The sensorweb project," <http://basics.eecs.berkeley.edu/sensorwebs>.
- [12] R. Nowak and U. Mitra, "Boundary estimation in sensor networks: Theory and methods," in *2nd International Workshop on Information Processing in Sensor Networks*, April 2003.
- [13] L. Lamport and N. Lynch, *Distributed Computing: Models and methods*, Formal Models and Semantics, volume B of Handbook of Theoretical computer Science, 1990.
- [14] R. S. Bhuvaneswaran, J. L. Bordim, J. Cui, and K. Nakano, "Fundamental protocols for wireless sensor networks," in *International Parallel and Distributed Processing Symposium, Workshop on Advances in Parallel and Distributed Computational Models*, April 2001.
- [15] M. Singh and V. K. Prasanna, "A hierarchical model for distributed collaborative computation in wireless sensor networks," in *International Parallel and Distributed Processing Symposium, Workshop on Advances in Parallel and Distributed Computational Models*, April 2003.
- [16] E. Cheong, J. Liebman, J. Liu, and F. Zhao, "Tinygals: A programming model for event-driven embedded systems," in *Symposium on Applied Computing*, March 2003.
- [17] R. Min and A. Chandrakasan, "Top five myths about the energy consumption of wireless communication," *Mobile Computing and Communications Review*, vol. 6(4), 2003.
- [18] A. Sinha and A. P. Chandrakasan, "Jouletrack: a web based tool for software energy profiling," in *Design Automation Conference*, June 2001.
- [19] J. Liu, M. Chu, J. Liu, J. Reich, and F. Zhao, "State-centric programming for sensor-actuator network systems," in *IEEE Pervasive Computing*, 2003.
- [20] H. M. Alnuweiri and V. K. Prasanna, "Fast image labelling using local operators on mesh connected computers," *IEEE Transactions on PAMI*, vol. 13, no. 2, pp. 202–207, February 1991.
- [21] K. Nakano and S. Olariu, "Energy-efficient randomized routing in radio networks," in *4th Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM)*, August 2000, pp. 35–44.
- [22] D. Devaguptapu and B. Krishnamachari, "Applications of localized image processing techniques in wireless sensor networks," in *Aerosense*, April 2003.