Contour Map Matching for Event Detection in Sensor Networks

Wenwei Xue, Qiong Luo, Lei Chen, Yunhao Liu
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
{wwxue, luo, leichen, liu}@cs.ust.hk

ABSTRACT

Many sensor network applications, such as object tracking and disaster monitoring, require effective techniques for event detection. In this paper, we propose a novel event detection mechanism based on matching the contour maps of in-network sensory data distribution. Our key observation is that events in sensor networks can be abstracted into spatio-temporal patterns of sensory data and that pattern matching can be done efficiently through contour map matching. Therefore, we propose simple SQL extensions to allow users to specify common types of events as patterns in contour maps and study energy-efficient techniques of contour map construction and maintenance for our pattern-based event detection. Our experiments with synthetic workloads derived from a real-world coal mine surveillance application validate the effectiveness and efficiency of our approach.

1. INTRODUCTION

Many sensor network applications monitor events in the physical world, such as disaster monitoring [3][29], habitat monitoring [19][27], industrial process control [1] and object tracking [12] [13][21]. A typical event detection mechanism in recent work on sensor databases is to set some thresholds for sensor readings in a query [1][12][29]. The intuition is that, when an event occurs, there will be changes in the readings of the sensors that are affected by the event. For example, when an object moves, the accelerometer attached to the object will report an increased acceleration reading. Based on this reasoning, an application program using thresholds will regard an event has occurred when the sensor readings exceed the pre-defined thresholds.

Although this threshold-based approach is simple in the implementation, it is usually difficult for users to specify suitable threshold values for their events of interest because these values depend on both the environment being monitored and the application semantics. Moreover, thresholds alone may be unable to fully specify an event for some applications. For instance, in a coal mine surveillance application that we are involved with, a gas leakage event is characterized as the *gas_density* sensor readings at the source following a gradual decreasing trend, which cannot be easily captured by discrete threshold values. Therefore, we investigate a new alternative for event detection in sensor networks.

SIGMOD 2006, June 27-29, 2006, Chicago, Illinois, USA.

Our proposed approach for event detection is based on the spatio-temporal patterns in sensor readings instead of simple thresholds. Our observation is that, since sensor networks are deployed in a physical space and sensor readings are collected over time, the changes in the sensor readings of networked nodes that are caused by an event usually exhibit some spatio-temporal pattern. This observation has been confirmed by various field studies and analysis of real-world sensory datasets [11][14] [23][27].

Now that we convert the event detection problem into a pattern matching one, the next question is how to solve the pattern matching problem in sensor networks effectively and efficiently. There has been a great wealth of literature on pattern matching [2][28], but the challenge is to seek a solution that works for a resource-limited network in a distributed, real-time, and energy-efficient way. The solution we find is in contour maps of sensory data distribution.

A *contour map* [5] of an attribute, e.g., temperature, for a sensor network is a topographic map that displays the distribution of the attribute value over the network. In the map, the geometric space occupied by the network is partitioned into contiguous regions, each of which contains sensor nodes of a range of similar readings. These regions are called *contour regions* and the boundaries of the regions are called *contour lines* or *contours* in short. We define a *snapshot of a contour map*, or a *map snapshot* in short, as the instance of the contour map at a specific point in time. Figure 1 shows a snapshot of the *temp* contour map (on the left) for a 2x2 network grid topology (on the right). The two contour regions in different colors represent different temperature readings.

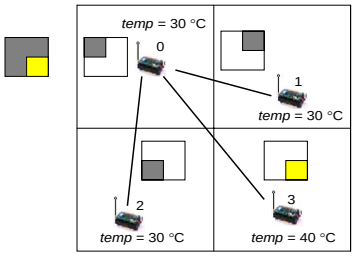


Figure 1: Temperature contour mapping on a 2x2 grid

Contour maps have been shown useful for a variety of sensor network surveillance applications [12][24]. They come naturally to our work because they represent the sensory data distribution over time and space. In our work, we propose energy-efficient

techniques to construct and incrementally update a number of 2-D contour maps in a sensor network. Using these contour maps as building blocks, we define events based on the spatio-temporal patterns exhibited in the contour maps. More specifically, the spatial pattern of an event is captured by the contours in a map whereas the temporal pattern by the evolution of the contours over time. In our approach, an event is detected when a user-specified pattern matches the recent snapshots of a contour map that fall in a sliding window. By this means, we solve the pattern matching problem, in turn the problem of event detection in sensor networks, through contour map matching.

Since events in the physical world are of a great variety, we use a real-world coal mine surveillance application as a case study for our work. In this application, hundreds of sensor nodes are deployed throughout the channels in the mine to measure the density of oxygen, gas and dust as well as the temperature, humidity and structural integrity in the mine. For the safety of the workers, there are two common classes of event-detection tasks in the application:

- *Gas, dust and water leakage detection.* The coal mine manager wants to be notified immediately whenever the digging machines reach a source of gas, dust or underground water in the mine and a large amount of the substance breaks out.

- *Oxygen density monitoring.* A worker patrolling in the mine needs to find a nearby spot of a high oxygen density to take a break from time to time. In addition, the manager requires the detection of regions of a low oxygen density.

We have identified and defined three common types of events for this application in addition to a general pattern-based definition for events. Each type of event is specified using a set of parameters that describe the shapes of the contours in the map, including the pyramids, the faults, and the islands. We have made simple SQL extensions to specify these types of events as user-defined methods and designed efficient algorithms to implement these methods in our event-oriented query processor. Finally, we have evaluated our approach using synthetic workloads that are derived from the application.

The remainder of this paper is organized as follows. We describe the in-network construction and incremental update of the contour maps in Section 2. We give our pattern-based event specification in Section 3. In Section 4, we briefly present the design of our event-oriented query processor, mainly on the query execution and the algorithms for contour map matching. In Section 5, we present a performance study of our approach using synthetic workloads derived from the real-world coal mine surveillance application. We discuss related work in Section 6 and conclude the paper in Section 7.

2. CONTOUR MAPPING

In this section, we describe contour maps, the building blocks of our pattern-based event specification. Our approach to contour mapping is to construct and incrementally update a contour map hop by hop from bottom up in the network as a special kind of data aggregation [12][18][29], rather than collecting all sensor readings and transmitting them to a server (i.e., the base station) to construct the map centrally. The motivation is that, for current generation battery-powered sensor nodes, power is the most limited resource and the communication cost on wireless radio channel is the dominating factor of power consumption. For instance, on the widely-used Crossbow MICA2 motes [6], the cost of transmitting a bit is about that of executing 1,000 instructions [19]. As a result, in-network contour mapping is more energy-efficient than a simple, centralized approach.

Section 2.1 describes the in-network construction of a contour map and Section 2.2 the incremental update of the map. The spatial interpolation on a random network topology is described in Section 2.3. In the remainder of the paper, we use the two terms "sensor reading" and "attribute value" interchangeably.

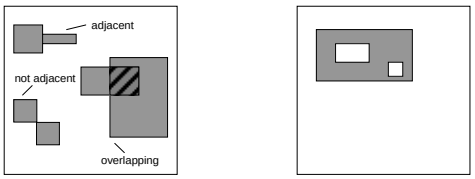
2.1 In-Network Map Construction

In this paper, we assume a stationary sensor network and the base station knows each node's location in 2-D Cartesian coordinates. This location information can be either measured manually, or acquired by special hardware, such as a GPS module, attached to a node. Moreover, as in the work by Hellerstein et al. [12], we impose a rectangular $m \cdot n$ grid with the square cell length l on the network topology. Each cell of the grid has at most one node inside but not on the cell boundaries. The grid information (i.e., the values of m, n and l) is produced at the base station and is disseminated throughout the network. Subsequently, each node in the network can calculate which grid cell it lies in.

Because wireless radio channels are unreliable, we adopt a multi-path, ring-based routing scheme [4][20] for our in-network map construction instead of using a single-path, tree-based routing scheme [18][19][29]. In this multi-path routing scheme, the data transmitted by a node is received by and processed on every neighbor of the node that is one hop closer to the sink node. We call these neighbors the *parents* of the node and the neighbors that are one hop farther from the sink the *children* of the node. The same as in single-path routing, a node in multi-path routing needs to transmit its data only once in a sample period.

2.1.1 Partial Map Aggregation

In our in-network contour map construction, the data aggregate generated and transmitted by a node is the contour map of a sub-network rooted at the node. We call this data a *partial map*. A partial map of a node consists of a set of disjoint contour regions. As shown in Figure 2, each contour region is an orthogonal polygon in 2-D plane in our grid setting. An orthogonal polygon is a polygon whose edges are in parallel with either the x-axis or the y-axis. Two contour regions *overlap* if their intersection has a non-zero area [22]. Regions that do not overlap are disjoint. We say that two disjoint regions are *adjacent* if they have one or more coincident edges (Figure 2 (a)).



(a) Adjacent and overlapping regions (b) Regions with holes

Figure 2: Example contour regions

A 2-D polygonal contour region is stored as a linked list [22] in a partial map. Each element in the linked list is an array of vertices. The first array in the linked list contains the vertices on the outer

boundary of the region in a counterclockwise order. Each of the other arrays contains the vertices in a clockwise order on an inner boundary of the region, i.e., the boundary of a hole (Figure 2 (b)). Each vertex in an array is stored as a pair of x-y coordinates.

The map construction starts from each node generating a partial map of its own. The partial map contains a single *contour region unit*, which is the grid cell of the node. After a node receives the partial maps from all children, it puts each contour region in these partial maps and its own into a new set P_w. We call P_w the *working partial map* of the node. Next, the node merges each pair of adjacent/overlapping regions in P_w that satisfies some criterion. The merging repeats until no pair of adjacent/overlapping regions in P_w can be merged. We call the partial map generated at the end of this merging the *final partial map* of the node, denoted as P_f. Then P_f is transmitted to the parents of the node by broadcast.

Such partial map aggregation on a node requires only simple operations on the polygonal contour regions. Three main classes of operations involved are [22]:

(1) *Area calculation*. Compute the area of a polygon using the coordinates of the vertices.

(2) *Relationship checking.* Identify whether two polygons overlap or are disjoint. If two polygons overlap, check whether they are exactly the same, one is contained in the other, or otherwise. If two polygons are disjoint, check whether they are adjacent.

(3) *Boolean operations*. Compute the union, intersection and difference of two polygons.

There are many existing algorithms in the literature for these basic polygon operations [22]. The small computation overhead and storage requirement of these algorithms make them applicable to current generation resource-limited sensor nodes.

A crucial problem we have omitted so far is what criterion we should adopt for a node to determine whether two adjacent or overlapping contour regions in P_w can be merged or not. The aggregation performance will be poor if we only merge regions that contain nodes having the same attribute value. Moreover, this criterion is unlikely to improve the mapping accuracy because readings sampled by physical sensors are intrinsically unreliable [29]. Another alternative is to divide the range of attribute value into a number of equal-width buckets and merge regions that are in the same bucket [12].

(a) (b)

Figure 3: Adjacent contour regions with different relative size

The major drawback of these criterions is that they only consider the attribute value but ignore another important factor, the area of a region. Combining region area with attribute value, we are able to capture the users' tradeoff between the mapping accuracy and the communication cost in a more flexible way. For instance, it is usually reasonable to merge a tiny contour region into a neighbor whose area is relatively much larger (Figure 3(a)), even if there is a large difference between the attribute values of nodes in these two regions. This merging saves communication cost of the tiny region without affecting the mapping accuracy much. In contrast, from some users' perspective, it may be undesirable to merge two adjacent regions when each of them has already occupied a large portion of the grid space (Figure 3(b)), even if the attribute values of the nodes in these two regions do not differ much.

Addressing the drawback, we design a criterion for contour region merging that takes both the attribute value and the region area into account. The criterion uses two user-specified parameters: (1) an error bound $\varepsilon \in (0, 1)$, and (2) a merging limit $p \in (0, 1]$. ε specifies the maximum degree of inaccuracy that the user can tolerate for the contour map constructed. p is the maximum area of the region to be merged in terms of the percentage of the grid area. The values of these two parameters are provided by the user in the query specification. ε is mandatory and p is optional with a default value of one.

A linear regression model $v = f(x, y) = c_0 + c_1 \cdot x + c_2 \cdot y$ [25] is built for each contour region in a partial map. In the model, v is the sensor reading of a node at location (x, y); c_0, c_1, and c_2 are the model coefficients and 1, x, y are the set of basis functions of the model. Consider a region that contains n nodes whose locations and sensor readings are (x_1, y_1), (x_2, y_2), …, (x_n, y_n) and v_1, v_2, …, v_n ($n \geq 3$) respectively. The coefficients of the model for the region can be computed by solving the matrix equation:

$$\mathbf{A}\mathbf{w} = \mathbf{b} \tag{1}$$

where $\mathbf{A} = \mathbf{V}^{\mathrm{T}}\mathbf{V} = \begin{pmatrix} 1 & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} y_i & \sum_{i=1}^{n} x_i y_i & \sum_{i=1}^{n} y_i^2 \end{pmatrix}$, $\mathbf{V} = \begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & y_n \end{pmatrix}$,

$\mathbf{w} = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix}$, $\mathbf{b} = \mathbf{V}^{\mathrm{T}}\mathbf{f} = \begin{pmatrix} \sum_{i=1}^{n} v_i \\ \sum_{i=1}^{n} v_i x_i \\ \sum_{i=1}^{n} v_i y_i \end{pmatrix}$ and $\mathbf{f} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}$.

When the initial partial map that contains a single region unit is generated by a node, the matrices $\mathbf{w}$, $\mathbf{A}$ and $\mathbf{b}$ in the linear regression model built for the region are stored with the region in addition to an error bound that has a value of zero. To ensure that the matrix $\mathbf{A}$ is non-singular [25], the model is built using three location-attribute pairs (x_1, y_1, v), (x_2, y_2, v) and (x_3, y_3, v). (x_1, y_1) and v are the location and attribute value of the node; (x_2, y_2) and (x_3, y_3) are two different locations that are randomly picked inside the region unit and are not equal to (x_1, y_1). Later, in the partial map aggregation on a node, the error bound ε_{ij} of merging a pair of adjacent/overlapping regions (R_i, R_j) in P_w is computed using Equations (2)-(4):

$$\varepsilon_{ij} = (\varepsilon_{ij}' + \varepsilon_{ij}'')(1 + \alpha_{ij}) \tag{2}$$

$$\varepsilon_{ij}' = \frac{\left| \iint_{Ri} (f_{ij}(x,y) - f_i(x,y))\,d\sigma \right| + \left| \iint_{Rj} (f_{ij}(x,y) - f_j(x,y))\,d\sigma \right|}{\iint_{Ri} f_i(x,y)\,d\sigma + \iint_{Rj} f_j(x,y)\,d\sigma} \tag{3}$$

$$\varepsilon_{ij}'' = \frac{\varepsilon_i \bullet \iint_{R_i} f_i(x,y)d\sigma + \varepsilon_j \bullet \iint_{R_j} f_j(x,y)d\sigma}{\iint_{R_i} f_i(x,y)d\sigma + \iint_{R_j} f_j(x,y)d\sigma} \quad (4)$$

In the equations, $f_i(x, y)$, σ_i and ε_i are the regression function, area and error bound of region R_i, and $f_j(x, y)$, σ_j and ε_j are those for region R_j. $f_{ij}(x, y)$ is the regression model built on $R_i \cup R_j$. To compute the coefficient vector $\mathbf{w}_{ij}$ of $f_{ij}(x, y)$ using Equation (1), we set $\mathbf{A}_{ij}$ and $\mathbf{b}_{ij}$ in the equation as follows [25]:

$$\mathbf{A}_{ij} = \mathbf{A}_i + \mathbf{A}_j \quad (5)$$

$$\mathbf{b}_{ij} = \mathbf{b}_i + \mathbf{b}_j \quad (6)$$

where $\mathbf{A}_i$, $\mathbf{b}_i$ are the matrices stored with region R_i, and $\mathbf{A}_j$, $\mathbf{b}_j$ are those of R_j. α_{ij} in Equation (2) is a penalty factor computed using Equation (7). $\sigma_{grid} = l^2 \cdot m \cdot n$ is the area of the grid.

$$\alpha = \max(0, \frac{\min(\sigma_i, \sigma_j) - p \bullet \sigma_{grid}}{p \bullet \sigma_{grid}}) \quad (7)$$

Note that $\iint_R f(x,y)d\sigma$ corresponds to the volume of the 3-D cylindroid that is beneath the surface $f(x, y)$ over region R [15]. Consequently, Equation (2) estimates the error bound ε_{ij} of merging the pair of regions (R_i, R_j) as the accumulation of the errors derived from two sources: the percentage of variation in the cylindroid volume over each region (ε_{ij}' in Equation (3)), and the previous degree of error inherited from each region (ε_{ij}'' in Equation (4)). In the worse case, the value of ε_{ij} for some pair of regions (R_i, R_j) computed using Equation (2) may exceed one. However, in this case the two regions are impossible to be merged so that no problem is incurred.

The regression model can be easily extended to use polynomial of a higher degree if necessary [25]. Moreover, we can adopt any model that is guaranteed to be continuous over an arbitrary contour region and can be incrementally recomputed in a way similar to Equations (5)-(6).

After giving the equations to estimate the merging error bound, we present the procedure of contour region merging in the partial map aggregation on a node in Algorithm 1. The algorithm calls the function *checkMerging* to examine whether a pair of regions in P_w can be merged or not. In this function, the error caused by a merging must not exceed the user-specified ε, and the size of the union of the two regions must be smaller than the sum of their sizes (Line 2 in Function *checkMerging*). This is because a large final partial map P_f requires a large communication cost so that its size should be reduced as much as possible to save the energy. *sizeof*(*R*) returns the total number of bytes of R stored in the partial map. This size is proportional to the total number of vertices on all boundaries of R.

Algorithm 1 merges regions in P_w pair by pair in a decreasing order of the merging benefit. The benefit of merging a pair of regions (R_i, R_j) is the inverse of $\varepsilon_{ij} / \Delta_{ij}$. The intuition is that, the smaller the error bound ε_{ij} of the merged region and the larger the size reduction Δ_{ij} achieved by the merging, the larger the merging benefit. After two regions are merged into a new region, the new region is immediately checked to see whether it can be further merged with other regions in P_w (Line 9). The process repeats until no more adjacent/overlapping regions in P_w can be merged.

Algorithm 1 Contour Region Merging

Input: the working partial map P_w
Output: the final partial map P_f

1: build an empty balanced binary search tree T;
2: **for** each R_i ($1 \le i < n$) in P_w **do** /* n is the number of regions in P_w */
3: **for** each R_j ($i < j \le n$) in P_w **do** *checkMerging*(R_i, R_j, T);
4: **while** T is not empty **do**
5: extract node a from T that has the smallest key value;
6: merge the pair of regions (R_i, R_j) that a points to into a region R;
7: delete all nodes in T that point to either R_i or R_j;
8: remove R_i and R_j from P_w;
9: **for** each R_k in P_w **do** *checkMerging*(R, R_k, T);
10: insert R into P_w; /* the linear regression model and error bound of
11: **while** *true* **do** R are $f_{ij}(x, y)$ and ε_{ij} correspondingly */
12: randomly select a pair of overlapping regions (R_i, R_j) in P_w;
13: **if** there is no such pair **then** break; /* exit the loop */
14: $r_i = sizeof(R_i - R_j) + sizeof(R_j)$; $r_j = sizeof(R_j - R_i) + sizeof(R_i)$;
15: **if** $r_i < r_j$ **then** $R_i = R_i - R_j$; **else** $R_j = R_j - R_i$;
16: return $P_f = P_w$;

Function *checkMerging*(R_i, R_j, T)

1: **if** R_i and R_j overlap or are adjacent **then**
2: **if** $\varepsilon_{ij} \le \varepsilon$ **and** $(\Delta_{ij} = sizeof(R_i) + sizeof(R_j) - sizeof(R_i \cup R_j)) > 0$ **then**
3: insert a node in T that has key value of $\varepsilon_{ij} / \Delta_{ij}$;
4: store a pointer to each of R_i and R_j with the node;

There may be overlapping regions in P_w that cannot be merged with each other according to our merging criterion. However, we need to ensure every two regions in P_f are disjoint so that there is no duplicate in the contour map constructed. To solve this problem, we remove the intersection of two overlapping regions that are non-mergeable from one of them so that the total size of the two regions is smaller after such removal (Lines 11-15 in the algorithm). The intersection is randomly removed from a region if the size reduction is the same for both regions.

2.1.2 Size Reduction of Partial Maps

The final partial map P_f output by Algorithm 1 on a node is usually of a large size. Each boundary of a contour region in the map consists of a sequence of (x, y) value pairs and all these lengthy pieces of information must be encapsulated into the radio packets and transmitted to the parent of the node in multi-path routing. In order to save the communication cost in this transmission, we adopt two techniques to reduce the size of P_f on each node. One technique is a scheme to compress the contours in the map and the other an optimization of the map transmission based on packet snooping. They both trade more computation cost for potentially less communication cost.

First, we use a scheme on each node to compress P_f before the node transmits it to the parents. The compression scheme includes two steps. In the first step, the node checks each contour region with holes in P_f to see whether any inner boundary of the region is the outer boundary of another region in P_f. If this is true, the inner boundary of the first region is replaced by a pointer to the outer boundary of the second region. By this means, only one copy of every contour line in the map needs to be transmitted.

The second step of the compression scheme is based on a nice property of the contour regions in our scenario, which is all edges of a region are orthogonal. Suppose there are totally n vertices V_1, V_2, …, V_n ($n \ge 4$) on a region boundary. Because the vertices are

stored in a specific order, we can reduce one half of the size of the boundary by storing the vertices V_1, V_3, …, V_{n-1} only. Later when the whole boundary needs to be recovered in the contour region merging, the vertex v_{2i} ($1 \le i \le n/2$) can be uniquely identified by v_{2i-1} and v_{2i+1} based on the counterclockwise or clockwise order of the triple of vertices (v_{2i-1}, v_{2i}, v_{2i+1}) [22].

In addition to the contour compression scheme, we adopt a simple optimization based on packet snooping [18] for the transmission of P_f on a node. The motivation of this optimization is that the packet size in a sensor network has a limit L that is determined by the hardware or network protocols. Consequently, if the size of P_f on a node is larger than L, the node has to divide the map into multiple packets for transmission. This case is very likely to happen when the sensor readings vary significantly along space, which causes the final partial map on a node to consist of many small regions that cannot be merged. Note that a variable number of packets per partial map does not affect the synchronization of the multi-path routing ring [4][20], because all packets for a partial map on a node are still transmitted within the fixed epoch allocated for the level in the ring that the node belongs to.

The snooping-based optimization is executed in parallel with the data transmission on each node. A node transmits its whole P_f in a single packet if the size of P_f is not larger than L. Otherwise, the node sorts the contour regions in P_f in an increasing order of their areas and transmits as many regions ahead in the order as possible in a packet. The node then removes all transmitted regions from P_f and snoops on the packets that are transmitted by the neighbors in the same hop. For each such packet P_{nf} snooped, the node removes each untransmitted region in P_f that is contained in some region in P_{nf}. Next the node begins a second round of transmission and the transmission iteration repeats until P_f becomes empty on the node. In each round of the iteration, if L is smaller than the size of any region in P_f, the node transmits a region with the smallest size in multiple packets.

This technique is feasible because multiple-path routing produces duplicate packets. As a result, small contour regions in the final partial map of a node may have been aggregated into larger ones on the same-hop neighbors. The transmission of these small regions on a node can be suppressed if the node detects that they are contained in some large regions on the same-hop neighbors of the node and that they have been transmitted by the neighbors.

2.2 Incremental Map Update

Our compression techniques reduce the size of a partial map, but it is still considerable. Moreover, in real-world surveillance applications, events are rare [9] and sensor readings are mostly unchanged or only slightly changed over time [14]. Consequently, consecutive map snapshots constructed for these events are extremely similar and incremental updates of the maps could save a large amount of energy. In contrast, if we blindly reconstruct a new map snapshot in every sample period of a continuous query, the energy of the nodes in the network will be depleted quickly due to the heavy communication cost.

Motivated by this observation, we propose an incremental update scheme to maintain a contour map after the first map snapshot has been constructed. This scheme is shown in Algorithm 2. In the algorithm, each node stores its working partial map P_{wo} and final partial map P_{fo} in the previous sample period. The node then generates its final partial map P_{fn} in the current sample period based on P_{wo}, P_{fo} and its working partial map P_{wn} in this period.

Algorithm 2 computes P_{fn} on a node by calling Algorithm 1 with a partial map P as the input (Line 15). P consists of three sets of contour regions: (1) the regions in P_{wo} that correspond to the update units in P_{wn} (Lines 2-6), (2) the regions in P_{wn} (Line 7), and (3) the regions in P_{wo} that do no overlap any of those in (1) and (2) sent by the same child (Lines 8-14). It contains all information a parent node gets from its children in the current sample period, and the information obtained in the previous sample period that can be reused.

If a node detects a region in P_{fn} is the same as a region in P_{fo}, the region in P_{fn} is replaced by an update unit before the transmission of P_{fn}. The update unit contains the new error bound, regression model and the leftmost vertex on the outer boundary of the region (Lines 21-22). By this means, we save the communication cost of the lengthy boundaries of the region. The leftmost vertex on a polygonal boundary is the one that has the smallest x-coordinate among all vertices on the boundary. If there are multiple vertices have the same smallest x-coordinate, we define the leftmost vertex as the upper one among them. The update unit can even be removed if the regression model of the corresponding region has not changed and the error bound has not increased (Line 23).

Algorithm 2 Incremental Update of Partial Map

Input: the working and final partial maps P_{wo} and P_{fo} in the previous sample period, the working partial map P_{wn} in the current sample period

Output: the final partial map P_{fn} in the current sample period

```
1:  P = ∅;
2:  for each update unit U in P_wn do
3:    find R_i in P_wo sent by the same child Ch as U that U.p is the leftmost
      vertex on the outer boundary of R_i;
4:    if R_i is missing in P_wo then request R_i from Ch;
5:    ε_i = U.ε;  f_i(x, y) = U.f;        /* ε_i and f_i(x, y) are the error bound and
6:    insert R_i into P;                    linear regression model of R_i */
7:  for each region R_i in P_wn do insert R_i into P;
8:  for each region R_i in P_wo do
9:    missing = true;
10:   for each region R_j in P do
11:     if R_i and R_j are not sent by the same child then continue;
12:     if R_i ∩ R_j ≠ ∅ then
13:       missing = false;  break;
14:   if missing then insert R_i into P;
15: P_wo = P;  call Algorithm 1 using P as the input;
16: for each region R_i in P_fn do
17:   origin = false;
18:   for each region R_j in P_fo do
19:     if R_i = = R_j then
20:       origin = true;  create a new update unit U;
21:       U.ε = ε_i;  U.f = f_i(x, y);
22:       U.p = the leftmost vertex on the outer boundary of R_i;
23:       if R_i.f ≠ R_j.f or R_i.ε > R_j.ε then insert U into P_fn;
24:       break;
25:   if origin then remove R_i from P_fn;
26: P_fo = P_fn;  return P_fn;
```

When a parent node receives an update unit from a child, the corresponding region of the unit is located from the parent's P_{wo} via the leftmost vertex stored in the update unit. It is possible that the region cannot be found in P_{wo} of the parent, because of the

transmission failure from the child to the parent in the previous sample period. In this case, the parent gets back the region from the child by a request message (Line 4). Note that a parent node can easily identify which region in its P_{wo} is from which child by attaching the id of the child that sent the region to the node. The algorithm finally updates P_{wo} and P_{fo} to P to P_{fn}, respectively.

2.3 Spatial Interpolation on Random Network Topologies

The topology of a sensor network in practice is usually a random connected graph instead of a grid. When we establish a grid on top of a random topology, some cells in the grid may be empty. These empty cells will not be involved in the in-network contour map construction, because none of them have a node inside. This absence of some cells makes the final partial map P_{sf} output by the sink incomplete, i.e., the union of all regions in P_{sf} covers only part of the entire grid. This problem will also occur on a grid topology if the partial maps of some nodes are lost due to the unreliability of radio transmission.

To make P_{sf}, the final partial map at the sink, always complete regardless of the network topology and packet loss, we perform spatial interpolation on P_{sf}. The interpolation is an iterative process. In each iteration, a grid cell c that is not contained in any region in P_{sf} but is adjacent to some region in P_{sf} is identified. Then for each region R in P_{sf} adjacent to c, the value $d = max(0, |max_n - min_n| - |max_o - min_o|)$ is computed for R. max_o and min_o are the maximum and minimum values of the regression function of R over region R; max_n and min_n are those of the function over region $R \cup c$. c is finally merged into the adjacent region in P_{sf} that has the smallest d value. The merged region inherits the regression model and error bound of the original region in P_{sf}. The iteration stops when each cell in the grid is contained in some region in P_{sf}.

3. PATTERN-BASED EVENT SPECIFICATION

Having presented the construction and maintenance of contour maps, we define the events and event-driven queries studied in our work.

3.1 Definitions

In our pattern-based approach, an event is specified as a kind of spatio-temporal pattern in a contour map of an attribute, as given in Definition 1.

Definition 1. [Event] An event is a time series $E = ((t_1, P_1), (t_2, P_2), \ldots, (t_n, P_n))$ with an equal time interval Δt between any two consecutive elements. Each element $P_i = (R_{i1}, R_{i2}, \ldots, R_{im})$ in the time series is a user-specified partial map of attribute A on a sensor network topology $(1 \le i \le n)$. Each contour region R_{ij} in P_i is associated with a single attribute value v_{ij} $(1 \le j \le i_m)$. $T = t_n - t_1$ is called the *event duration*.

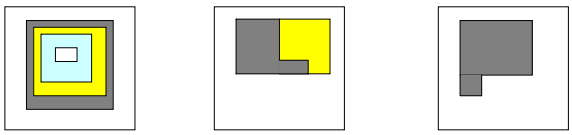
We say a snapshot C of the contour map of A matches P_i if and only if every region in P_i matches its overlapping regions in C within a user-specified confidence level $(1 - \alpha) \in (0, 1)$.

We say the contour map of A matches E at time t if and only if for each P_i, it matches the snapshot C_i of the contour map of A at time $t - \Delta t \cdot (n - i)$. t is called the time of event detection.■

This definition requires a user to give a specific time series of partial maps as the spatio-temporal pattern generated by an event. In each sample period of the query that monitors the event, this use-specified pattern is compared with the snapshots of a contour map to determine whether the event has occurred. Δt in the definition is the length of the query sample period. The reason we only consider a contour map of one attribute in the definition is to make the structure of an event as simple as possible. Moreover, we only require partial maps instead of complete map snapshots from the user so that the user does not need to consider the regions in the grid that are of no interest.

A drawback of our event definition is that, if a user does not have perfect knowledge about an event, the user may not be able to specify the value distribution of an attribute over space and the variation of this distribution over time incurred by the event. As a first attempt to address this limitation, we further define three common types of events that we observe in several sensor network applications, especially the coal mine application that we described in the Introduction. Each type of event corresponds to a rough shape in a contour map and is specified using a set of parameters to fix the relative positions and values of the contour regions in the shape. To distinguish from these common events, we call the events defined by Definition 1 "general events" in the remainder of the paper.

Figure 4 shows the shapes in the contour map incurred by the three types of events, which we call the pyramid, the fault, and the island respectively.



(a) Pyramid (b) Fault (c) Island

Figure 4: Illustration of three common types of events

Intuitively, a pyramid event generates a continuous, gradual increasing or decreasing trend of attribute value in all directions originated from a small region in the space. For instance, when the gas begins to leak in the coal mine, the contour map of the *gas_density* attribute matches a decreasing pyramid event, with the inmost region of the event in each map snapshot containing the source of the leakage.

Different from a pyramid event, a fault event corresponds to a sudden and large change in the level of the attribute value in the space. For instance, when the underground water begins to leak in the coal mine, the *humidity* map matches a fault event. One region involved in the event definition suffers from the water leakage but the other does not. Note that water leakage is different from gas leakage in that the change in humidity does not occur in all directions as that in the gas_density attribute does.

Different from both pyramids and faults, an island event corresponds to a region in the space that has a consistently large or small level of the attribute value. For instance, when a worker in the coal mine wants to find a place of a high oxygen level to take a break, it corresponds to detecting a large-valued island event in the *oxy_density* map.

In the following, we give the formal definitions of these three types of events.

Definition 2. [Pyramid Event] Given a user-specified bucket size k, a snapshot C of the contour map of attribute A matches an increasing pyramid event if and only if a list of contour regions $P = (R_1, R_2, \ldots, R_n)$ in C satisfies the following four conditions:

(1) Each region R_i in P is associated with a bucket, i.e., it corresponds to a range of attribute values $[b_i \cdot k, (b_i + 1) \cdot k)$ $(1 \le i \le n)$. $b_i \ge 0$ is an integer and $b_{i+1} - b_i \ge 1$. R_1 is called the inmost region and R_n the outmost region of the pyramid event.

(2) R_i is a hole in R_{i+1} and the ratio of their areas $\sigma_i / \sigma_{i+1} \le sf$. $sf \in (0, 1)$ is a user-specified scaling factor.

(3) $\sigma_1 \ge \sigma$. $\sigma > 0$ is a user-specified area bound.

(4) $n \ge N$. $N > 1$ is a user-specified nesting level.

Given a user-specified event duration T, we say the contour map of A matches an increasing pyramid event at time t if and only if every snapshot of the contour map falling into the interval $[t - T, t]$ matches an increasing pyramid event, and the inmost regions of the event found in any two consecutive map snapshots have an overlapping area of at least $\sigma \cdot (1 - \alpha)$. $(1 - \alpha) \in (0, 1)$ is a user-specified confidence level. t is called the time of event detection.

A decreasing pyramid event is defined symmetrically by replacing "$b_{i+1} - b_i \ge 1$" with "$b_{i+1} - b_i \le 1$" in Condition (1).■

Note that we do not require the value of each b_i in Definition 2 to be fixed by the user. It can be any integer as long as the condition $b_{i+1} - b_i \ge 1$ $(1 \le i \le n)$ is satisfied.

Definition 3. [Fault Event] Given a user-specified bucket size k, a snapshot C of the contour map of attribute A matches a fault event if and only if two contour regions R_1 and R_2 in C satisfy the following three conditions:

(1) R_1 is associated with the bucket $[b_1 \cdot k, (b_1 + 1) \cdot k)$ and R_2 the bucket $[b_2 \cdot k, (b_2 + 1) \cdot k)$. $b_1, b_2 \ge 0$ are two integers and $b_2 - b_1 \ge \Delta$. $\Delta \ge 1$ is a user-specified degree of difference.

(2) R_1 and R_2 are adjacent. The coincident polygonal curve on their outer boundaries has a length of at least $N \cdot l$. $N \ge 1$ is a user-specified number of coincident cell edges. l is the width of a cell.

(3) The areas of the two regions $\sigma_1 \ge \sigma$ and $\sigma_2 \ge \sigma$. $\sigma > 0$ is a user-specified bound of area.

Given a user-specified event duration T, we say the contour map of A matches a fault event at time t if and only if every snapshot of the contour map falling into the interval $[t - T, t]$ matches a fault event, and the corresponding regions of the event in any two consecutive map snapshots have an overlapping area of at least $\sigma \cdot (1 - \alpha)$. $(1 - \alpha) \in (0, 1)$ is a user-specified confidence level. t is called the time of event detection.■

Definition 4. [Island Event] Given a user-specified threshold value $\tau > 0$, a snapshot C of the contour map of attribute A matches a large-valued island event if and only if a contour region R can be found in C that satisfies the following two conditions:

(1) R is associated with a range of attribute values $[\tau, +\infty)$.

(2) The area of R is not smaller than σ. $\sigma > 0$ is a user-specified area bound.

Given a user-specified event duration T, we say the contour map of A matches a large-valued island event at time t if and only if every snapshot of the contour map falling into the interval $[t - T, t]$ matches a large-valued island event, and the regions of the event in any two consecutive map snapshots have an overlapping area of at least $\sigma \cdot (1 - \alpha)$. $(1 - \alpha) \in (0, 1)$ is a user-specified confidence level. t is called the time of event detection.

A small-valued island event is defined symmetrically by replacing "$[\tau, +\infty)$" with "$(-\infty, \tau]$" in Condition (1).■

Finally, we note that even though there are a number of user-specified parameters in the definitions, most of the parameters have default values in our implementation. As a result, when users use these events in the form of system-provided functions, their effort is minimized.

3.2 Event-Driven Queries

Based on the definitions of the events, we extend an existing SQL-style sensor query language [19][29] to support the specification of our pattern-based event detection.

We encapsulate the general events and the three common types of events as system built-in Boolean methods. The four methods are *event(mapSnapshot, confFile)*, *pyramid(mapSnapshot, confFile)*, *fault(mapSnapshot, confFile)* and *island(mapSnapshot, confFile)*, correspondingly. The first parameter of each method is a snapshot of a contour map. The second parameter is the pathname of an XML configuration file given by the user. This file lists the sequence of partial maps that define a general event, or the values of the parameters that define a common event. A method returns *true* when the contour map matches the use-specified pattern at the current time; otherwise it returns *false*.

All of these methods are used in the WHERE clause of an SQL query. Multiple methods that are evaluated on different contour maps can be connected by the AND/OR SQL keywords to specify complex relationships between events. We call queries that are embedded with one or more of these methods *event-driven queries*. The SELECT clause of an event-driven query may contain attributes, SQL aggregates as well as user-defined functions. Two example event-driven queries are listed as follows:

Query 1:

```
SELECT      c.snapshot
FROM        contour_map(temp,0.2, 0.5) c
WHERE       event(c.snapshot, "fire_emergency.xml")
SAMPLE PERIOD     10 sec
```

Query 2:

```
SELECT      alarm()
FROM        contour_map(gas_density,0.3) c
WHERE       pyramid(c.snapshot, "gas_leakage.xml")
```

As shown in these two example queries, the construction of a contour map used by some event is specified in the FROM clause using the table-valued function *contour_map(attribute,ε, p)*. The first parameter of this function is the attribute of the map, and the latter two are the user-specified parameters ε and p for the map construction. Note that p is optional, as shown in Query 2. Similar to the *sensors* table [19], each *contour_map* function defines a virtual table for the query. The table contains two main fields, *timestamp* and *snapshot*. *snapshot* is the map snapshot at *timestamp*. A list of *contour_map* functions, possibly on different attributes, can appear in the FROM clause of a single query.

4. EVENT-ORIENTED QUERY PROCESSING

Given the pattern-based event specification, we now present our event-oriented query processing framework. We focus on how the system built-in methods are handled during the execution of continuous queries (Section 4.1) and the algorithms for contour map matching (Section 4.2).

4.1 Query Execution

Query execution in our framework is event-oriented. When a new event-driven query is issued by a user, the query text and the parameters for each *contour_map* function in the query are parsed into a query message. The message is then injected into the sensor network. When the query message is received by a node, the distributed query processor on the node generates a sub-query evaluation plan based on the information encapsulated in the message and begins to execute the plan.

The sub-plan of a query on a node is in charge of the in-network contour map construction. After the query has been installed on the nodes, the contour maps in the query are constructed in the first sample period and are incrementally updated in the subsequent periods. In the case that a map of the same attribute is already being used by other running queries, one snapshot of the map will be aggregated in the network every minimum sample period of these multiple queries, and the aggregation uses the minimum ε and p values of those of the multiple event methods. By this means, multiple concurrent queries share a single construction and maintenance procedure of a contour map.

This contour map sharing among multiple queries saves the energy consumption significantly while preserving the requirement of all queries on event detection accuracy. This benefit is achieved by using a linear regression model for contour region merging rather than using equal-width buckets [12] in the in-network contour mapping. For example, if two queries share a contour map using equal-width buckets of sizes k_1 and k_2, the map must be constructed using a bucket size $k = gcd(k_1, k_2)$. However, even if both k_1 and k_2 are large, k may be small so that the aggregation performance will be poor (e.g., $k_1 = 30$, $k_2 = 100$, and $k = 10$). In contrast, using a linear regression model, the map is constructed in a way independent of the bucket size of each query.

In addition to the sub-query plans on the nodes, a main evaluation plan is generated for a query at the base station. The main plan is responsible for event detection according to the configuration files of the methods in the query. Each method is implemented as a data structure in this main plan. The structure contains following fields: (1) a flag indicating whether the method defines a general event or a common event, (2) the user-provided partial maps for a general event or parameter values for a common event, and (3) a copy of each snapshot of M that falls in the sliding window $[t_c - T, t_c]$. Here M is the contour map that the method is evaluated on, t_c is the current system time and T is the event duration. For brevity, in the following we call these map snapshots stored with a method in the main query plan "the map snapshots of the method".

At the end of a sample period of a query, each method in the query is evaluated using the map snapshots of the method. The core of the evaluation procedure is a corresponding algorithm for contour map matching we design. The algorithms are described in Section 4.2. If the combination of all methods in the WHERE clause is evaluated to *true* at this time, the data acquisition, aggregation or user-defined functions specified in the SELECT clause of the query are performed.

4.2 Algorithms for Contour Map Matching

We have designed and implemented four algorithms for contour map matching in our framework, one for the general events and each of the other three for a common type of events. These algorithms strictly follow the definition of their corresponding events in Section 3.1. Consequently, we only give the algorithm for matching a general event, Algorithm 3, as an example and omit those for the common events.

We point out that the matching of a common event defined in a method requires further post-processing of the map snapshots of the method. Such post-processing is conducted before each map snapshot is stored with the method. Specifically, we associate each grid cell with a bucket based on the mean value of the regression function [15] on this region in the map snapshot. How the whole range of the attribute value is divided into a number of buckets is specified by the user in the configuration file of the method. After associating a bucket with each grid cell, adjacent cells that are in the same bucket are merged and a different version of snapshot is obtained.

Algorithm 3 Matching of a General Event

Input: the general event E defined in method m
Output: the current return value of m

1: **for** each partial map P_i $(1 \le i \le n)$ in E **do**
2: C_i = the map snapshot of m at the time $t_c - (n - i) \cdot sp$; $p = 0$;
3: **for** each region R_j $(1 \le j \le i_m)$ in P_i **do**
4: **for** each region R_k $(1 \le k \le l)$ in C_i **do**
5: **if** $(R_h = R_j \cap R_k) \neq \varnothing$ **then**
6: $p \mathrel{+}= (\sigma_h / \sigma_j) \cdot (| \iint_{Rh} f_j(x, y) d\sigma - v_j \cdot \sigma_h | / (v_j \cdot \sigma_h))$;
7: **if** $1 - p < (1 - \alpha)$ **then** return *false*; **else** $p = 0$;
8: return *true*;

5. EXPERIMENTS

In this section, we evaluate the performance of our proposed pattern-based event detection mechanism using a homegrown sensor network simulator.

5.1 Experimental Setup

We simulated the event detection scenario of a coal mine surveillance application in our experiments. As we have described in the previous sections, this application involves several event detection tasks, such as gas leakage detection, water leakage detection and oxygen density monitoring. The sensor readings and the coefficients of the regression model are all 2 bytes in the experiments. We set the size limit of a packet in our simulator to be 49 bytes based on our application development experiences on the Crossbow MICA2 motes [6]. We use the GPCJ library [10] for the polygon operations in in-network contour mapping.

5.1.1 Data Generation

We have conducted field studies in a coal mine and collected a small amount of real-world sensory data. This real-world dataset mainly contains three attributes: *gas_density*, *oxy_density* and *humidity*. However, due to the resource and environment

constraints, the collected data is coarse-grained in both temporal and spatial granularity. The sample period of the nodes was set at the minute level and there were only a few nodes deployed in the mine. Consequently, we generated synthetic datasets based on the settings and the sensory data characteristics revealed in the real-world dataset, and used the synthetic datasets in the experiments.

Each synthetic dataset we generated contains the three attributes. The nodes form an $N \cdot N$ grid with a cell length of 10 meters. There is a node at the center of each grid cell and the sink node is at the upper-left corner of the grid. The dataset contains 1000 tuples from each node, with one tuple generated every second. The initial value v_{i0} for an attribute on a node i in the dataset is randomly selected from a set of seed values that we found most common for that attribute in the real-world dataset. The attribute value on the node in a subsequent second is randomly picked from the range $[v_{i0} \cdot (1-d), v_{i0} \cdot (1+d)]$ with a probability of P_i, or remains unchanged from the previous second with the probability of $1 - P_i$. The value of P_i for a node i is uniformly chosen from the range (0, 0.3]. $d \in (0, 1)$ is the maximum variation percentage of the attribute value in the real-world dataset.

Finally, we change the synthetic dataset to embed event data for each query workload. The event data contain spatial-temporal patterns that the query is supposed to detect. They are embedded into the dataset by replacing the attribute values of the tuples in the dataset. The locations of these patterns in the dataset are randomly selected. The details of the query workload are given in Section 5.1.2.

5.1.2 Query Workload

For each synthetic dataset, we generated a query workload consisting of four classes of queries QC1-QC4 and ran the workload over the dataset in our experiments. Each class in a workload contains 30 queries embedded with a kind of event detection method. QC1-QC4 contain the *event*, *pyramid*, *fault* and *island* methods, respectively. Each query in a class is to detect a randomly generated event instance of the corresponding type.

In a query workload, the queries in QC2, QC3 and QC4 involve the *gas_density*, *humidity* and *oxy_density* attributes respectively, so that they correspond to gas leakage detection, water leakage detection and oxygen density monitoring. In contrast, each 10 out of the 30 queries in QC1 involve one of these three attributes so that QC1 can evaluate the performance of our mechanism in detecting general events that produce arbitrary spatio-temporal patterns in sensory data. All of the 120 queries in the workload use the parameter values $\varepsilon = 0.2$ and $p = 1$ for the contour map construction and has a sample period of 1 second. The bucket size of a query in QC2-QC4 is uniformly picked from (0, 1000].

5.1.3 Performance Metrics

We used *accuracy of event detection* and *network traffic* as the two metrics for performance evaluation. The accuracy of event detection measures the effectiveness of our approach and the network traffic reflects the power efficiency. Both metrics are important for sensor network surveillance applications.

The accuracy of event detection includes two sub-metrics similar to those used in Information Retrieval: (1) *precision*, which is the percentage of real events detected over all events reported by a query, and (2) *recall*, which is the percentage of events in the synthetic dataset that are successfully detected by a query. The network traffic is defined as the total number of bytes transmitted by all nodes in the network during the execution of a query for a fixed period of time.

5.1.4 Approaches Compared

The main idea of our pattern-based event detection mechanism is to convert the problem of event detection in sensor networks into pattern matching and then contour map matching. Therefore, the approach to contour map construction and maintenance is most important to the performance of our mechanism. Consequently, in our experiments we focused on evaluating the effectiveness of our linear regression based approach to in-network contour mapping.

We compared the performance of our approach with two other alternative approaches: (1) equal-width bucket based in-network contour mapping [12], and (2) server-side contour mapping using our proposed regression model for contour region merging without in-network aggregation. The first alternative is for comparing equal-width bucket with linear regression in contour mapping whereas the second for comparing a centralized approach against in-network aggregation. We denote these approaches as INLR (In-Network Linear Regression), INEB (In-Network Equal-width Bucket) and SSLR (Server-Side Linear Regression), correspondingly.

For a fair comparison, all three approaches used multi-path routing for data transmission. Moreover, similar to INLR we applied simple techniques for duplicate elimination and incremental update on each node for INEB and SSLR.

5.1.5 Parameters Considered

We varied a number of system parameters in our simulator when comparing the performance of the three approaches of contour map construction. These parameters are listed as follows:

(1) *Event Frequency* (F). It is the frequency at which the spatio-temporal pattern of the event appears in the dataset. For instance, suppose a pattern appears at 100 positions of a 1000-tuple dataset, the event frequency is computed as $(100/1000) \cdot 100\% = 10\%$.

(2) *Network Diameter* (D). It is the width in meters of the square $N \cdot N$ grid.

(3) *Transmission Range* (T). It is the maximum distance between two nodes in a grid topology that can communicate with each other directly.

(4) *Link Loss Rate* (R). It is the probability with which a packet from a child node to its parent will be dropped in our simulator.

5.2 Efficiency of Our Approach

Before comparing the performance of our INLR with the other two approaches of contour mapping, we first validated the usefulness of the techniques it adopts. These techniques include: (1) the contour compression scheme (Section 2.1.2), (2) the snooping based optimization of partial map transmission (Section 2.1.2), and (3) the incremental update scheme (Section 2.2). Because none of these techniques affect the accuracy of event detection, we only show the results on network traffic.

Figure 5 shows the network traffic of five variants of our INLR. The parameter setting we used in this experiment was $F = 10\%$, D

= 100m, T = 15m and R = 0%. A value in the figure was the average of those of the 30 queries in a class when each of the queries was run individually over the dataset. This is the same for all figures we showed in the following.

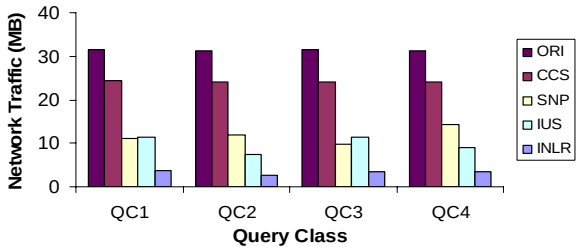


Figure 5: Network traffic of different INLR variants

In Figure 5, ORI is the original version with none of the three techniques enabled in our approach. CCS, SNP and IUS are the variants with only the contour compression scheme, the snooping based transmission optimization and the incremental update scheme enabled, respectively. Finally, INLR is the full version with all three techniques enabled.

As shown in the figure, for the query workload we used, CCS, SNP and IUS saved about 25%, 55-70%, 65-75% communication cost in comparison with ORI. This saving indicates that the incremental update scheme is most beneficial in energy conservation among the three techniques, the next the snooping based transmission optimization, and then the contour compression scheme. When all three techniques are adopted together, they can save nearly 90% communication cost over ORI.

5.3 Comparison of Three Approaches

In this section, we varied each of the four system parameters while keeping the other parameters fixed to investigate the effect of the parameter on the performance of the three approaches. Because there is no concept of bucket involved in the definition of general events, INEB was excluded for QC1 in the experiments due to its inapplicability to this class of queries. For each query in QC2-QC4, INEB used the same bucket size for the contour map construction as that used in INLR and SSLR for event detection.

Our experimental results showed that QC1-QC4 revealed a similar performance trend among the three approaches for all parameter settings. As a result, we only provide the results of QC2 as examples. In all runs of the experiments, the three approaches consistently achieved a 100% precision no matter how each of the four system parameters was varied. Also, the accuracy of event detection was hardly affected by the variation of the other three parameters except for the link loss rate. Therefore, we only report the results on recall when the link loss rate varied.

Figure 6 shows the recall of the three approaches when the network link loss rate was varied from 0 to 30%. In the figure we see that, all three approaches failed to report several occurrences of the event when the link loss rate became large. The accuracy of event detection achieved by our INLR was as good as SSLR and both of them outperformed INEB by 10-20% when the link loss rate is not zero. The poor recall of INEB is mainly because the packet loss makes an original large region in a bucket become individual small pieces, so those regions that satisfy the area bound requirement of our event definition can not be found.

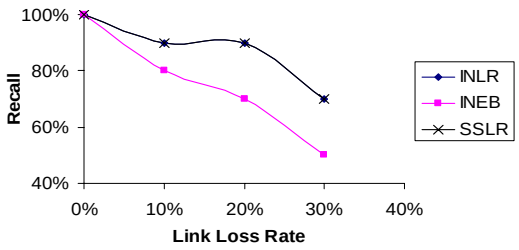


Figure 6: Recall of the three approaches with different link loss rate (F = 10%, D = 100m, T = 15m)

As for the network traffic, Figure 7 shows the metric values of the three approaches when the event frequency, network diameter, transmission range and link loss rate were varied in the range of 5-20%, 50-200m, 10-30m and 0-30% respectively. The other parameters fixed for each sub-figure are listed in the figure title. The results in each of these sub-figures are described in order.

As illustrated in Figure 7(a), both INLR and INEB consumed only 10% more communication cost when the events appeared more frequently. This indicates that frequent occurrences of an event will not cause significant increase in network traffic of in-network contour mapping; the network traffic of an in-network approach remained small compared with a server-side approach. We limited the event frequency to be under 20% because events are usually rare in the physical world [9].

Figure 7(b) demonstrates that in-network contour mapping was more scalable in large networks than the naïve sever-side approach. INLR was slightly more scalable than INEB even though the difference between the two was small.

In Figure 7(c) we see that, the network traffic in INLR decreased almost linearly and that of INEB increased almost linearly with the increase of the transmission range. This is because when the transmission range of the nodes is large, the data from a node can be received by more nodes in the network that are far away from the node. However, the small regions from two distant nodes was very likely to be in different buckets so that the communication cost in INEB for data forwarding throughout the network increased. INLR could still merge these regions into large ones using the linear regression model so that its performance benefited from the increase of the transmission range.

Figure 7(d) shows that the network traffic of SSLR decreased linearly when the link loss rate increased. This is because many packets were dropped at the first few hops of the multi-hop routing and the upper hop nodes that are near the sink did not need to forward these packets any more. INLR was most indifferent to the link loss rate in terms of network traffic among the three. However, the network traffic of INLR still slowly decreased when the link loss rate became larger.

Interestingly, the network traffic of INEB increased when more packets were dropped. The reason was that when some contour regions transmitted from a child node to its parent were lost in the transmission, the probability that on the parent node two regions in the same bucket would become adjacent or overlapping was lowered due to the missing of the region that connects them in space. Consequently, the aggregation performance of INEB got poor since many small, disjoint regions had to be transmitted instead of a single, large one.

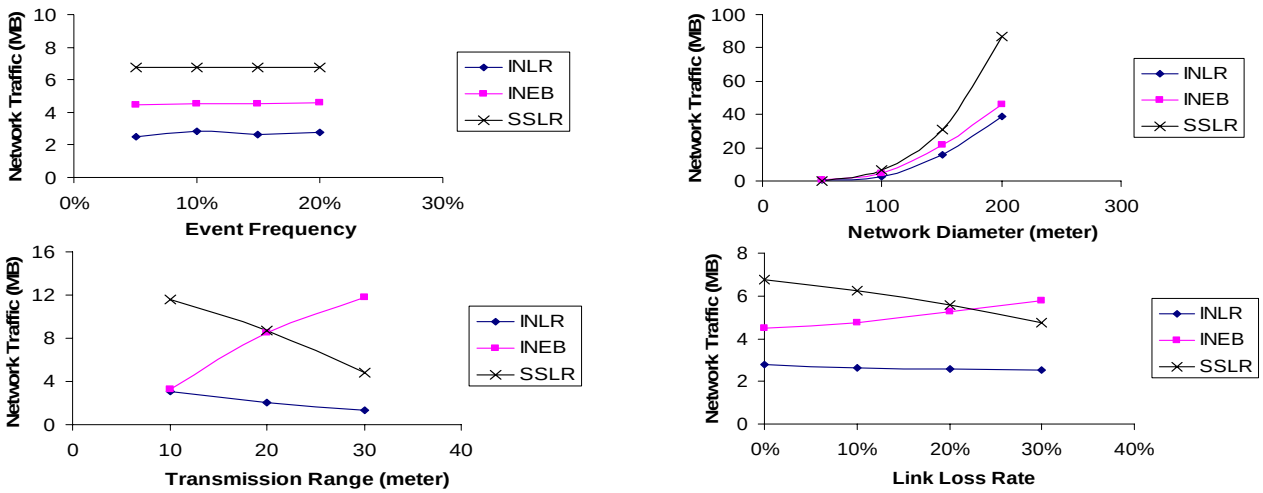


Figure 7: Network traffic of the three approaches with different parameters varying (a) event frequency (*D* = 100m, *T* = 15m, *R* = 0%) (b) network diameter (*F* = 10%, *T* = 15m, *R* = 0%) (c) transmission range (*F* = 10%, *D* = 100m, *R* = 0%) (d) link loss rate (*F* = 10%, *D* = 100m, *T* = 15m)

As a summary of the results in Figure 7, our proposed INLR consistently outperformed the other two approaches in network traffic. The network traffic of INLR is indifferent to the event frequency, scales well with a large network size and decreases properly when the transmission range gets larger.

6. RELATED WORK

There have been a number of recent publications on event-oriented query processing for sensor databases. The Cougar sensor database encapsulates the logic of threshold-based event detection into asynchronous functions in the system [3]. TinyDB [12][19] processes events generated from a threshold-based query, a module hand-coded in the operating system, or an interrupt from some hardware component such as a switch or a motion detector. REED [1] extends TinyDB to support efficient in-network join operations, which are useful for event detection in industrial process control applications. In comparison, our approach does not require the existence of any hand-coded OS modules or hardware equipment. Instead, we define an event based on the spatio-temporal pattern it generates in sensor readings and perform pattern-based event detection by contour map matching.

In the networking community, Directed Diffusion [13] detects the emergence of an animal by matching the sensor readings of a node with the pattern libraries stored on the node. This pattern-based event detection is similar to our approach. A difference is that, instead of considering temporal pattern matching on individual nodes, we study the spatial pattern incurred by an event throughout the network and the evolution of this pattern over time. Another piece of work on event detection in sensor networks is DSWare [17], a sensor network middleware that provides event detection services based on node grouping. A new event is registered to DSWare by inserting a tuple that specifies the semantics of the event into a system-level event table. Nevertheless, none of this line of work employs contour map matching as the means for event detection.

Our in-network contour mapping has been influenced by the previous work of Hellerstein et al. [12]. Their work was to use the TinyDB query processor to construct contour maps for sensor network applications whereas ours is to use contour maps in our query processor as the means for event detection. Furthermore, we have presented a systematic solution to the event detection problem, with emphasis on the multi-path routing, the linear regression model for contour region merging, and the size reduction and incremental update of the partial maps.

Both Hellerstein et al. and we establish the grid topology for contour maps. This regularity eases the event specification in our approach, because it is more complex for the users to specify contour regions if the boundaries are not straight lines but arbitrary planar curves. In contrast, Solis and Obraczka have studied using isoclines as the basis for contour map construction instead of using polygonal regions [26]. In addition, they divided the range of attribute values into equal-width buckets and used single-path data routing.

Deshpande et al. proposed to use a centralized probabilistic model to optimize the processing of various types of sensor queries [8]. The model captures the spatio-temporal correlations between the readings of multiple types of sensors on a node. Although the authors commented that the model could be extended to a distributed version with continuous sampling for event detection, no detailed techniques have been presented in this regard yet.

Linear regression has been shown effective for data compression [7] or redundancy suppression [16] in sensor networks to reduce the communication cost of sensory data acquisition. We use linear regression to construct the contour maps that represent the sensor reading distribution over a network.

Finally, the detection of the common types of events that we define bears some similarity to the shape matching queries in traditional [2] or streaming [28] time series databases. Since our work is targeted at sensor network surveillance applications, we focus on enabling the detection of these events in a simple but effective way via in-network contour mapping.

7. CONCLUSION

In this paper, we have proposed a pattern-based approach to event detection in sensor networks. Our approach is implemented by

matching user-specified patterns with the contour maps of sensory data distribution. We give a general pattern-based definition for the events, and propose simple SQL extensions to allow users to specify several common types of events as patterns in contour maps. We propose a number of energy-efficient techniques for in-network contour mapping, including a linear regression based criterion for contour region merging, two techniques for size reduction of the partial map transmitted by a node, and an incremental update scheme. Our experimental results with synthetic workloads derived from a real-world coal mine surveillance application shows that our approach of in-network contour mapping can achieve a good accuracy. Moreover, it greatly saves the network traffic in comparison with an existing equal-width bucket based approach and a server-side approach with contour map matching.

We are working on a prototype implementation of our proposed in-network contour mapping techniques on a kind of sensor nodes that are similar to the MICA2 motes. The algorithms implemented in the current prototype, including multi-path routing, regression-based contour region merging and contour compression, are simplified from those presented in the paper, due to the resource constraints of the current generation motes. Preliminary results on this prototype in our lab show that the computation cost of the current prototype is suitable for MICA2-grade sensor nodes when the network size is small (tens of nodes). Moreover, we have deployed a 27-node network in a coal mine in the mainland China and plan to install and test the prototype on this real deployment after a robust version is finished. Documentation and source code packages of this work are publicly available at our project web site www.cs.ust.hk/aorta.

Other on-going work includes revising the pattern-based event specification to be more user-friendly, evaluating the performance of our mechanism using patterns of events from real-world datasets, and re-implementing and evaluating our approach, including the libraries required for polygon operations, on PDA-grade micro-server sensor nodes.

8. ACKNOWLEDGMENTS

Funding for this work was provided in part by the NSFC Key Project Grant No. 60533110, and the Hong Kong RGC grants AoE/E-01/99, HKUST6158/03E, HKUST6263/04E and DAG05/06.EG03.

9. REFERENCES

[1] Abadi, D., Madden, S., and Lindner, W. REED: Robust, efficient filtering and event detection in sensor networks. VLDB, 2005.

[2] Agrawal, R., Psaila, G., Wimmers, E., and Zaït, M. Querying shapes of histories. VLDB, 1995.

[3] Bonnet, P., Gehrke, J., and Seshadri, P. Querying the physical world. IEEE Personal Communications, 7(5), 2000.

[4] Considine, J., Li, F., Kollios, G., and Byers, J. Approximate aggregation techniques for sensor databases. ICDE, 2004.

[5] Contour Map. http://en.wikipedia.org/wiki/Contour_map.

[6] Crossbow Inc. www.xbow.com.

[7] Deligiannakis, A., Kotidis, Y., and Roussopoulos, N. Compressing historical information in sensor networks. SIGMOD, 2004.

[8] Deshpande, A., Guestrin, C., and Madden, S. Model-driven data acquisition in sensor networks. VLDB, 2004.

[9] Dutta, P., Grimmer, M., Arora, A., Bibyk, S., and Culler, D. Design of a wireless sensor network platform for detecting rare, random, and ephemeral events. IPSN, 2005.

[10] GPCJ. http://www.seisw.com/GPCJ/GPCJ.html.

[11] Guralnik, V., and Srivastava, J. Event detection from time series data. SIGKDD, 1999.

[12] Hellerstein, J. M., Hong, W., Madden, S., and Stanek, K. Beyond average: Towards sophisticated sensing with queries. IPSN, 2003.

[13] Intanagonwiwat, C., Govindan, R., and Estrin, D. Directed diffusion: A scalable and robust communication paradigm for sensor networks. MOBICOM, 2000.

[14] Intel Lab Data. http://berkeley.intel-research.net/labdata/.

[15] Kaplan, W. *Advanced Calculus*. Addison-Wesley, Boston, MA, USA.

[16] Kotidis, Y. Snapshot queries: Towards data-centric sensor networks. ICDE, 2005.

[17] Li, S., Lin, Y., Son, S., Stankovic, J., and Wei, Y. Event detection services using data service middleware in distributed sensor networks. Telecommunication Systems, 26(2-4), 2004.

[18] Madden, S, Franklin, M. J., Hellerstein, J. M., and Hong, W. TAG: A tiny aggregation service for ad-hoc sensor networks. OSDI, 2002.

[19] Madden, S., Franklin, M. J., Hellerstein, J. M., and Hong, W. The design of an acquisitional query processor for sensor networks. SIGMOD, 2003.

[20] Manjhi, A., Nath, S., and Gibbons, P. Tributaries and deltas: Efficient and robust aggregation in sensor network streams. SIGMOD, 2005.

[21] Ni, L. M., Liu, Y., Lau, Y., and Patil, A. LANDMARC: Indoor location sensing using active RFID. Wireless Networks, 10(6), 2004.

[22] O'Rourke, J. 1998. *Computational Geometry in C*. Cambridge University Press, New York, NY, USA.

[23] Papadimitriou, S., Brockwell, A., and Faloutsos, C. Adaptive, hands-off stream mining. VLDB, 2003.

[24] Rahimi, M., Pon, R., Kaiser, W., Sukhatme, G., Estrin, D., and Srivastava, M. Adaptive sampling for environmental robotics. ICRA, 2004.

[25] Ruppert, D., Wand, M. P., and Carroll, R. J. *Semiparametric Regression*. Cambridge University Press, New York, NY, USA.

[26] Solis, I., and Obraczka, K. Efficient continuous mapping in sensor networks using isolines. Mobiquitous, 2005.

[27] Szewczyk, R., Mainwaring, A., Polastre, J., Anderson J., and Culler, D. Lessons from a sensor network expedition. EWSN, 2004.

[28] Wu, H., Salzberg, B., and Zhang, D. Online event-driven subsequence matching over financial data streams. SIGMOD, 2004.

[29] Yao, Y., and Gehrke, J. Query processing for sensor networks. CIDR, 2003.