

**Kennesaw State University**  
**DigitalCommons@Kennesaw State University**

---

Faculty Publications

---

10-1-2012

# Area Query Processing Based on Gray Code in Wireless Sensor Networks

Chunyu Ai

Yueming Duan

Mingyuan Yan

Jing He

Kennesaw State University, [jhe4@kennesaw.edu](mailto:jhe4@kennesaw.edu)

Follow this and additional works at: <https://digitalcommons.kennesaw.edu/facpubs>

 Part of the [Computer Engineering Commons](#), and the [Computer Sciences Commons](#)

---

## Recommended Citation

Ai, Chunyu; Duan, Yueming; Yan, Mingyuan; and He, Jing, "Area Query Processing Based on Gray Code in Wireless Sensor Networks" (2012). *Faculty Publications*. 3820.

<https://digitalcommons.kennesaw.edu/facpubs/3820>

This Article is brought to you for free and open access by DigitalCommons@Kennesaw State University. It has been accepted for inclusion in Faculty Publications by an authorized administrator of DigitalCommons@Kennesaw State University. For more information, please contact [digitalcommons@kennesaw.edu](mailto:digitalcommons@kennesaw.edu).

# Area Query Processing Based on Gray Code in Wireless Sensor Networks

Chunyu Ai<sup>1,\*\*</sup>, Yueming Duan<sup>2</sup>, Mingyuan Yan<sup>2</sup>, Jing He<sup>3</sup>

1. Department of Computer Science, Troy University, Troy, AL 36082, USA;
2. Department of Computer Science, Georgia State University, Atlanta, GA 30303, USA;
3. Department of Computer Science, Kennesaw State University, Kennesaw, GA 30144, USA

**Abstract:** Area query processing is significant for various applications of wireless sensor networks since it can request information of particular areas in the monitored environment. Existing query processing techniques cannot solve area queries. Intuitively, centralized processing on Base Station can accomplish area queries via collecting information from all sensor nodes. However, this method is not suitable for wireless sensor networks with limited energy since a large amount of energy is wasted for reporting useless data. This motivates us to propose an energy-efficient in-network area query processing scheme. In our scheme, the monitored area is partitioned into grids, and a unique gray code number is used to represent a Grid ID (GID), which is also an effective way to describe an area. Furthermore, a reporting tree is constructed to process area merging and data aggregations. Based on the properties of GIDs, subareas can be merged easily and useless data can be discarded as early as possible to reduce energy consumption. For energy-efficiently answering continuous queries, we also design an incremental update method to continuously generate query results. In essence, all of these strategies are pivots to conserve energy consumption. With a thorough simulation study, it is shown that our scheme is effective and energy-efficient.

**Key words:** area query; area query processing; gray code; wireless sensor networks

## Introduction

Wireless sensor networks are widely used in many significant applications such as industrial process control, environmental monitoring, habitat monitoring, and object tracking. Since users of wireless sensor networks focus on raw sensed values or processed information of the monitored area, sensor nodes are designed to cooperatively sense, collect, and process the raw information of the monitored area, and send

the processed information to observers. Sensor-based applications extract different kinds of data via collecting data, processing in-network aggregations, and detecting complicated events. Since sensor nodes have limited resources, novel data management techniques are desired to satisfy application requirements which consider characteristics of wireless sensor networks.

Most existing data collection systems are query-based ones. Traditional query processing techniques of wireless sensor networks mainly deal with retrieving sensor node locations, sensed values, and aggregating the sensed values. However, in many applications, users expect information about areas of their interests. For instance, workers in a coal mine want to find an area with a high oxygen density to take a break, and this area

---

Received: 2012-07-05; revised: 2012-07-17

\*\*To whom correspondence should be addressed.

E-mail: chunyuai@troy.edu; Tel: 1-334-670-3409

must be big enough to accommodate several workers. In Fig. 1, all the sensed values of the sensors in regions R1 through R5 reach the expected oxygen density. However, regions R3 and R5 are not big enough for several workers, so R3 and R5 are not acceptable. For this application, the existing methods may only return the locations and sensed values of the sensors which satisfy the oxygen density condition, yet it is meaningless and insufficient. Here, users want to find areas instead of multiple sensor locations since the size of an area is also a filtering condition. The traditional methods do not consider spatial correlation among sensor nodes, that is critical for many applications.

Another interesting scenario is an air pollution monitoring application within a city. Users expect the monitoring system to detect regions whose pollution levels reach different thresholds. In reality, the polluted thresholds might be different for different areas. For example, the threshold of an industrial area is 80, while that of a residential area is 50. In Fig. 1, R1 and R4 are in the industrial area, and R2, R3, and R5 are in the residential area. The polluted level of R1 through R5 are 70, 60, 40, 90, and 40, respectively. Usually, the traditional methods apply the same filtering condition for the whole network. In this example, if the threshold 80 is used, polluted area R2 is missed since R2's polluted level actually exceeds the threshold for residential areas while the filtering condition does not

indicate this. If the threshold 50 is used, R1 is identified as a polluted area which triggers a false alarm. So the existing methods cannot be used for this kind of applications. These applications concern specific areas, and each area has its own specific filtering conditions.

According to the requirements of area query applications, we define *area query* as requests for area information including area locations, sizes of areas, and collected and aggregated data of the areas. Sensor nodes with expected readings and adjacent sensing coverage are divided into the same group. The total coverage area of the sensor nodes in the same group is a possible result area. An area query can retrieve not only sensed values but also specific geographical information compared with traditional queries of wireless sensor networks. Area query is more practicable and useful than traditional queries for some applications, which require geographical information. A common property of area query applications is that the results of these queries are areas and the aggregation values of sensors in these areas. Size of areas can also be query conditions. Furthermore, queries might be run for either the whole network or sub-areas of the network. For different areas, it is possible to use different querying conditions.

Area query is a new kind of query in wireless sensor networks. Intuitively, Base Station can collect data from all sensor nodes and then process area queries in a centralized manner; however, sensor nodes will drain energy quickly due to frequent reports of sensed values. The existing techniques of in-network query processing suppress and aggregate sensed values to save energy. Nevertheless, these methods do not consider sensor coverage area as part of the results. Consequently, a new in-network area query processing technique is necessary.

Due to inherent limitations of wireless sensor networks, designing an in-network area query processing mechanism is a challenging issue. The first challenge of in-network area query processing is that it is hard to suppress useless data as early as possible. For instance, an area satisfies all the conditions except the size condition. Obviously, this area is not the ultimate expected result. However, we cannot drop this area since it is difficult to decide the boundary of an area locally. Moreover, how to describe an area in-network is another challenge. Because areas are also part of query results, an ideal area description can reduce the amount of the transmitted data and lower

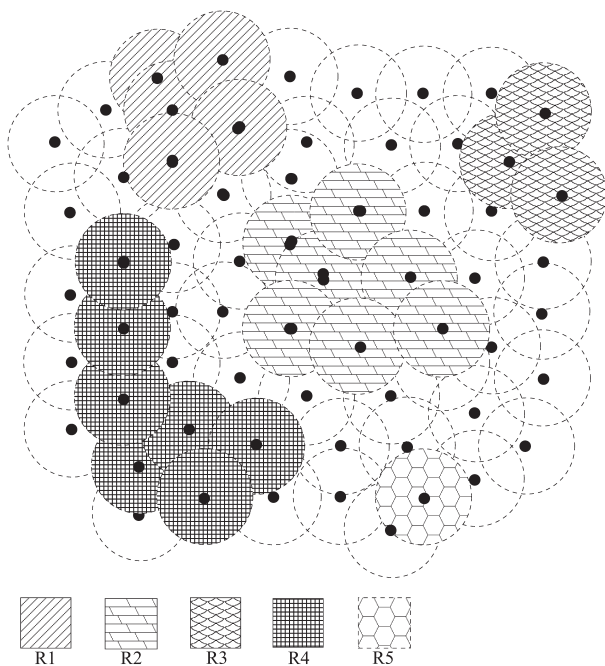


Fig. 1 Area queries.

the complexity of area size computation, which are two primary considered factors for energy conservation.

In this paper, we propose an energy-efficient in-network area query processing scheme. The whole network is partitioned into grids, and a gray code is used to represent each grid. We construct a reporting tree to merge areas and process aggregations. To conserve energy, incremental update techniques are used to process continuous queries. Our contributions are summarized as follows:

- (1) A new area query is studied in this paper.
- (2) A smart area description, Grid ID (GID) list, is used to reduce the size of query results.
- (3) An energy-efficient in-network area query processing scheme is proposed. By using GIDs and merging strategies, partial results can be merged to reduce the size of results and useless data can be dropped as early as possible to reduce the number of messages.
- (4) An incremental update method is addressed to reduce the size of reports for continuous queries.

## 1 Related Work

Area query processing is a special kind of data processing in wireless sensor networks. Data processing of wireless sensor networks has been already well-studied. It mainly involves three aspects, data collection, data aggregation, and query processing. Resource-limitation and unreliable communication links are the main concerns.

Data collection is involved by most of the applications, which collects all sensed data continuously (SELECT \* query). In Ref. [1], an approximate data collection scheme was proposed. Temporal and spatial correlations are used to construct data prediction models. Sensed data is reported only if it differs significantly from prediction, thus it can avoid frequent reports to save energy. In Ref. [2], a data-driven approach was presented. Model-based suppression is used to provide continuous data without continuous reporting. In addition, a key problem for data suppression, link failure, was addressed. A mobile filtering approach for error-bounded data collection was proposed in Ref. [3]. By migrating filters wisely the amount of transmitted data is reduced significantly. In Ref. [4], a sensor network is divided into clusters, and the leader of each cluster discovers local data correlations. At the sink, a global approximate data collection is performed

according to model parameters reported by cluster leaders. The data collection techniques achieve ideal data reduction when bounded error is acceptable to users. Data collection may be used to process area queries. That is, Base Station can compute the results after collecting data from all sensor nodes. Because errors are not tolerable for area queries, methods which may provide results with errors are not suitable for area query processing. If the methods which provide accurate results are employed, the energy consumption will be larger since most of data cannot be suppressed and must be sent to Base Station for processing.

In-network data aggregation is an efficient way to reduce energy consumption since data items are compressed on their way to the destination. TAG<sup>[5]</sup> and TiNA<sup>[6]</sup> aggregate sensed data as early as possible through tree-based routing. Aggregation of values from multiple sensor nodes is routed to one destination (Base Station). In Ref. [7], a many-to-many aggregation scheme was proposed. In this scheme, there are multiple destinations, and each destination may also have multiple source nodes. In Ref. [8], a continuous spatial aggregation was studied. Users specify a set of spatial regions, and then data aggregation is performed in each region. The existing data aggregation methods either consider the sensor nodes in the whole network as belonging to one group or divide the sensor nodes into static groups such as the works in Refs. [7, 8]. Area query processing requires dynamic sensor grouping based on readings from each sensor; therefore, the above mentioned methods cannot be used to process area queries.

Query processing technique involves query optimization, decomposition, distribution, and result retrieval. A query processing system must have the ability to execute multiple continuous queries simultaneously. The Cougar<sup>[9]</sup> system reduces communication cost by pushing operations such as selections and aggregations into the network to reduce the size and the number of messages. TinyDB<sup>[10]</sup> is a robust query processing system. An SQL-like interface is used to specify the data expected by users. It also supports event detection. Power-aware optimization, dissemination, routing, and execution techniques are used in TinyDB to prolong network lifetime. Query processing can be utilized for event detection. In an event detection system, when an event is detected, a warning should be delivered to users. In Refs. [10, 11], thresholds are set for sensor readings in a query

to detect events. In Ref. [12], events are abstracted into spatial-temporal patterns. In this scheme, event detection is effectively carried out through matching contour maps of sensed data to event patterns. To the best of our knowledge, area query processing has not been investigated. This is the first work to address area queries.

In this paper, we propose an area query processing scheme, which can handle not only normal queries like selection, aggregation, and threshold-based event detection but also area queries. More importantly, this scheme is energy-efficient since it processes area queries in an in-network manner. Compared with the existing techniques, the remarkable difference is that this scheme supports dynamic sensor grouping which is an indispensable step for area query processing. For dynamic sensor grouping, we consider not only geographical correlations of sensors' sensing coverage but also sensed data correlations to derive expected results.

## 2 Area Query in Wireless Sensor Networks

Sensor nodes might be equipped with several sensing components to monitor environment or detect events. Users can specify a query, which describes an event or specific areas of users' interests. An area query is defined as follows:

```
SELECT areas and/or aggregation functions
FROM entire sensor network or subareas
[WHERE predicates]
[GROUP BY adjacency]
[HAVING predicates]
[DURATION time – span]
EVERY time – interval]
```

SELECT presents query results. It can be areas and some aggregation functions of the sensed values of each area. FROM specifies the queried areas. We can query the whole network or just focus on some particular areas. Users can write complicated query conditions on sensing attributes by the WHERE clause. GROUP BY is used to divide sensor nodes into groups according to the expected readings and adjacent sensing coverage. This is the main difference between traditional queries and area queries. HAVING presents the conditions on the aggregation functions. The requirements of the result areas can be specified via the HAVING clause. DURATION specifies the lifetime of a continuous query. EVERY defines

an execution interval, which means the query is continuously executed and the results are returned every *time-interval* time unit. If a query does not specify DURATION and EVERY, it is not a continuous query, and it will be executed just once. A continuous query can also be used to describe events since it will be executed periodically. The coal mine example mentioned in the Introduction section can be described in terms of the following area query.

```
SELECT area, area.avg(sensors.oxygen)
FROM sensors
WHERE sensors.oxygen > 80
GROUP BY adjacency
HAVING area.size > 50 m2
DURATION 240 hours
EVERY 60 seconds
```

It depicts that an area is qualified when the oxygen density is greater than 80, and the acreage of this area is greater than 50 m<sup>2</sup>. This query will be continuously executed for 240 h. The results will be reported to users every 60 s. To guarantee the safety of workers, this query continuously reports the results to users. This query also reports the average oxygen density of each qualified area to users. Most previous aggregation methods provide the aggregated values of some particular properties, but not for each area.

Another example, air pollution monitoring, is described in terms of the following query.

```
SELECT area, area.avg(sensors.pollution – level)
FROM industrial area as R1, residential area as R2
WHERE R1.sensors.pollution – level > 80,
      R2.sensors.pollution – level > 50
GROUP BY adjacency
HAVING R1.area.size > 1000 m2, R2.area.size > 100 m2
DURATION 72 hours
EVERY 20 minutes
```

In this example, we independently specify the pollution-level threshold for industrial area as 80 and for residential area as 50. Also, in the HAVING clause, the size requirement of the result areas for industrial area is 1000 m<sup>2</sup> and for residential area it is 100 m<sup>2</sup>. Area query allows different WHERE and HAVING conditions for different areas.

A scheme which can efficiently process area queries is then necessary. Due to limited computation and energy resources of sensor networks, energy-efficiency is the main optimization goal, and reducing in-network computation complexity should also be a concern.

### 3 In-network Area Query Processing Scheme

In this section, an in-network area query processing scheme is presented. As we know, energy is the most limited resource for current battery-powered sensor nodes. Our approach obtains an initial query result and incrementally updates the result along a reporting tree in a bottom-up manner, rather than collecting all the sensor reports and transmitting them to Base Station to process queries centrally. Since communication cost is the dominating factor of energy consumption, in-network area query processing is more energy-efficient than a simple centralized approach.

#### 3.1 Area partition

We assume that every point in the monitored area is covered by at least one sensor node. In addition, the sensor network is static, and the locations (2-D Cartesian coordinates) of sensor nodes are known to Base Station where the location information can be either acquired by Global Positioning System (GPS) or measured manually. Localization algorithms also can be used to calculate the locations of the sensor nodes. To easily and clearly describe areas, the whole monitored area is divided into small grids. First, the whole monitored area is vertically partitioned into two even subregions. Then, these two subregions are further horizontally partitioned into four subregions. For each subregion, we recursively partition it vertically and horizontally. The partition process ends either there is only one sensor node in that subregion or the size of this subregion is not greater than the partition threshold  $P_t$ .  $P_t$  is the minimum size of a grid defined by users. If there are enough sensor nodes, we suggest that the diagonal length of a grid is no larger than a sensor's sensing range, so that each sensor can cover its resident grid. The accuracy can then be better guaranteed. This partition method can be applied to any area with irregular shape. A monitored area with any shape can be bounded by a rectangle. We can partition this rectangle step by step as mentioned above. During the partition process, if a subarea is totally outside of the monitored area, it is ignored. In this way we divide the whole network into grids as shown in Fig. 2.

The grid at the top-right corner has two sensor nodes. Since the size of this grid is less than  $P_t$ , we can stop the partition even though there are more than one sensor node in that grid. The grid at the bottom-right

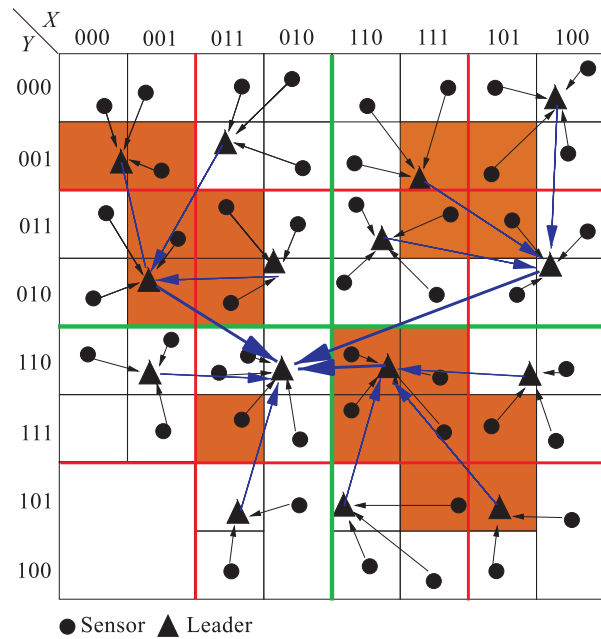


Fig. 2 Working scenario.

corner cannot be further divided because there is only one sensor node in it. There are no sensors in the grid at the bottom-left corner since they are outside of the monitored area. After partition, there is at least one sensor node in each grid within the monitored area. The average sensed values of all the sensor nodes in each grid is treated as the sensed value of that grid. To prolong network lifetime, the sensor nodes in a grid can be on duty alternately. The benefit of using this partition method is that it is convenient to merge the qualified subareas and carry out size calculation since each grid is a rectangle.

As Base Station is in charge of partitioning, it stores geographical location information of all the grids. We use a unique GID to represent each grid so that Base Station can easily acquire location information according to a GID. Gray code<sup>[13]</sup> is a binary numeral system where two successive numbers differ by only one bit. We adopt gray codes as GIDs. Let  $X$  represent a gray code number in horizontal direction, and  $Y$  represent a gray code number in vertical direction. A GID is formatted as  $(X, Y)$ . As shown in Fig. 2, the grid at the top-right corner is  $(100, 000)$ . The grid at the bottom-right corner is  $(100, 10x)$ , where  $x$  means either 0 or 1 indicating the union of  $(100, 101)$  and  $(100, 100)$ . Obviously, the length of a GID is the number of partition iterations in both vertical and horizontal directions. Furthermore, we can use one GID to express the union of multiple adjacent grids by

replacing the union of 0 and 1 with  $x$ . For instance,  $(10x, 00x)$  is the union of the four grids,  $(100, 000)$ ,  $(100, 001)$ ,  $(101, 000)$ , and  $(101, 001)$ , at the top-right corner. This scheme can significantly reduce the amount of the maintained information.

The recursive partition algorithm is described in Algorithm 1. First, the whole monitored area is bounded by a rectangle. We use two vertices on the same diagonal to decide a rectangle's location since Base Station needs to record the location information of a particular GID. Base Station runs Algorithm 1 to divide this rectangle into grids. Since the monitored area may have an irregular shape, it is possible that there exists non-monitored area in this rectangle as shown by the bottom-left corner grids in Fig. 2. In the process of partitioning, once a non-monitored

subregion is detected, it is ignored (Algorithm 1, Lines 1-3). The rectangle is recursively partitioned in vertical and horizontal directions alternatively until the *return* condition is satisfied. The *return* condition is true when either the subarea is outside of the monitored area or the current subarea cannot be further partitioned since there are not enough sensors to guarantee that there is at least one sensor in each partitioned subarea. For each generated grid, the location information, GID  $(X_b, Y_b)$ , and the sensor nodes in this grid are recorded. In Algorithm 1,  $(X_l, X_r)$  and  $(Y_l, Y_r)$  are coordinates of the two vertices representing the current partition area.  $D$  is the partition direction, and it can be either  $X$  (vertical) or  $Y$  (horizontal).  $C_X$  and  $C_Y$  are parameters to generate gray codes. Function  $\text{Shift}_{\text{left}}$  is used to shift a binary number one bit left. This algorithm generates GID for each grid and guarantees that GIDs of adjacent grids differ by only one bit. After partition, for each grid, Base Station sends its unique GID to the sensor nodes within that grid.

---

**Algorithm 1: Partition** $(X_l, X_r, Y_l, Y_r, X_b, Y_b, D, C_X, C_Y)$

**Input:** Partition area.

**Output:** Grids with GIDs.

/\* Initially, Partition( $X_{\text{left}}, X_{\text{right}}, Y_{\text{left}}, Y_{\text{right}}, \text{null}, \text{null}, X, 0, 0$ ) is called. \*/

---

```

1: if area( $(X_l, X_r), (Y_l, Y_r)$ ) has no intersection with the
   monitored area then
2:   return;
3: end if
4: if  $\sqrt{(X_r - X_l)^2 + (Y_r - Y_l)^2} \leq P_t$  then
5:   Output the Grid  $((X_l, X_l), (X_r, Y_r))$  with GID: $(X_b, Y_b)$ .
6:   return.
7: end if
8: if it cannot be partitioned in direction  $D$  then /* If the
   condition, at least one sensor node in each grid, cannot be
   guaranteed. */
9:   if  $D == X$  (i.e., verticality) then /* If it cannot
   be partitioned in vertical direction, try to partition it in
   horizontal direction. Otherwise, return the current grid. */
10:    Partition( $X_l, X_r, Y_l, Y_r, X_b, Y_b, Y, C_X, C_Y$ )
11:   else
12:    Output the Grid  $((X_l, X_l), (X_r, Y_r))$  with
   GID: $(X_b, Y_b)$ .
13:   end if
14: end if
15: if  $D == X$  then
16:   Partition( $X_l, \frac{(X_l+X_r)}{2}, Y_l, Y_r, \text{Shift}_{\text{left}}(X_b) +$ 
    $C_X, Y_b, Y, 0, C_Y$ ).
17:   Partition( $\frac{(X_l+X_r)}{2}, X_r, Y_l, Y_r, \text{Shift}_{\text{left}}(X_b) +$ 
    $C_X, Y_b, Y, 1, C_Y$ ).
18: else
19:   Partition( $X_l, X_r, Y_l, \frac{(Y_l+Y_r)}{2}, X_b, \text{Shift}_{\text{left}}(Y_b) +$ 
    $C_Y, X, C_X, 0$ ).
20:   Partition( $X_l, X_r, \frac{(Y_l+Y_r)}{2}, Y_r, X_b, \text{Shift}_{\text{left}}(Y_b) +$ 
    $C_Y, X, C_X, 1$ ).
21: end if

```

---

### 3.2 Reporting tree construction

To perform in-network queries, a reporting tree needs to be constructed by Base Station. In Fig. 2, an example reporting tree is shown. This tree is constructed from leaf nodes to the root. After deploying sensor nodes, Base Station knows the initial energy and GID of all the sensor nodes. We group the grids whose GIDs differ by the last bits of  $X$  and  $Y$  codes (i.e.  $(\%x, \%x)$ , where  $\%$  can represent any binary string of any length) together. In each group, a sensor node is chosen to be the leader (triangular nodes shown in Fig. 2) of this group. Other sensor nodes in the group report data to its leader. In other words, the leader is the parent node of other sensor nodes in the same group. Then, among the leaders in the groups of grids  $(\%xx, \%xx)$ , one sensor node is chosen to be the parent node of the other leaders identified in the previous step. Next, we consider the groups with grids  $(\%xxx, \%xxx)$  and so on. The process is repeated until the root of this tree is reached. That is, finally we consider all the grids as an entire group. The height of this reporting tree is  $\text{Max}(|X|, |Y|) + 1$ , where  $|X|$  and  $|Y|$  are the length of gray code  $X$  and  $Y$  respectively. After the reporting tree is constructed, Base Station sends children and parent information to each sensor node. In our scheme, the reporting tree does not serve as the routing tree. The routing can be generated by any routing protocol such as DSDV in Ref. [14].

Since the leaders, non-leaf nodes, have to collect the reports from their children and report to their parent, they consume much more energy than leaf nodes. To prolong network lifetime, we should reconstruct the reporting tree periodically, and let the sensor nodes with more residual energy serve as leaders. However, to construct a new tree, Base Station has to collect energy information from all the sensor nodes, and after generating the new tree, Base Station has to broadcast the tree structure to the whole network. This consumes a large amount of energy. Therefore, we periodically generate a new reporting tree through switching child and parent roles in the tree in a distributed way. This switching process starts from the bottom of the tree and ends at the root. When the network is initialized, a synchronized tree reconstruction timer is set for every sensor node. For each sensor node, when this timer goes off, it triggers the tree reconstruction process and the timer is reset for the next tree reconstruction. For a leaf node, it simply sends its residual energy information to its parent. When a leader receives residual energy reports from its children, it compares with its own residual energy. If the current leader's energy is smaller than that of its children, it chooses the child with the maximum residual energy as the new leader. Also, it informs all its children to change their parents to the new leader including itself. The old leader also sends its parent information to the new leader and the new leader can reset its parent. After this level's role switch of child and parent, the new leader sends its residual energy information to its parent, then the next level role switch can be triggered. Switching process is accomplished in a bottom-up manner until it propagates to the root. When the new root is chosen, it informs Base Station that a new root node is generated. Since the whole tree reconstruction is performed in a distributed way without Base Station collecting residual energy information, the energy consumption is reduced significantly compared to centralized processing. Most importantly, the tree reconstruction can prevent network lifetime end early due to workload skew.

### 3.3 In-network area query processing

The motivation of in-network processing is that the number of transmission messages can be reduced by merging multiple reports as one report and filtering useless data as early as possible, thereby prolonging network lifetime.

When Base Station receives an area query request, it registers this query. Then, the query is decomposed into query conditions, merging conditions, query lifetime, and executing interval. Query conditions, lifetime, and executing interval are sent to every sensor node in the queried area. Merging conditions are sent to non-leaf nodes of the reporting tree. If a query is a continuous query, it is executed every time interval.

In essence, query conditions are these filtering conditions on sensing attributes defined by the WHERE clause on properties such as temperature, moisture, and oxygen density. Query lifetime and executing interval are specified in the DURATION and EVERY clauses respectively. A noteworthy point is that different sensor nodes might accept different query conditions if they are in different query areas since users might define different conditions for different areas such as the air pollution monitoring example mentioned in the Introduction section. Throughout the lifetime of a query, a sensor node sends a report to its parent every interval if its readings satisfy the query conditions. The reporting message is formatted as  $(SID, GID, RID, V_1, V_2, \dots, V_M)$ , where SID is a sensor node ID, GID is the grid ID, RID is the area ID to indicate different query areas such as industrial area and residential area in the example mentioned in Introduction, and  $V_i$  is the sensed value of the attribute  $i$ .

Merging conditions include aggregation operations and filtering conditions specified by the HAVING clause. The number of merging steps  $S$  is  $\text{Max}(|X|, |Y|)$ . We start from step 1 involving grids  $(\%x, \%x)$  and stop at step  $S$  involving all the grids. In other words, the merging process follows the reporting tree in the bottom-up order. In the reporting tree, every non-leaf node is responsible for at least one merging step. If a non-leaf node is the root for grids  $(\%x \dots x, \%x \dots x)$ , it is the merging node for these grids. The merging step  $i$  is the number of  $x$  in  $X$  or  $Y$ . Some non-leaf nodes might participate in several merging steps, and the root node participates in all the merging steps. For instance, in Fig. 2, the root node (in the grid (010, 110)) processes steps 1, 2, and 3 merging for grids (01x, 11x), (0xx, 1xx), and (xxx, xxx) one by one. In a merging process, a leader merges adjacent subareas from its children into one and filters areas which cannot be merged by the up-level leader. Moreover, aggregations are calculated after merging. Then, the leader generates a report and sends



it to its parent. Specially, the root sends a report to Base Station. The report is formatted as (RID, Subarea<sub>1</sub>, GID list, data list, RID, Subarea<sub>2</sub>, GID list, data list, ..., RID, Subarea<sub>n</sub>, GID list, data list). A GID list is used to describe an area. For instance, the shaded areas at the top-left corner in Fig. 2 is denoted as (00x, 001), (0x1, 01x) instead of 6 GIDs. Consequently, GID lists can reduce the size of area description information since one GID can represent multiple grids.

There are two main difficulties for merging areas. Firstly, how to judge two subareas are adjacent since location information is not maintained by sensor nodes. Secondly, how to identify an isolated area as early as possible. For a non-leaf node, it cannot decide whether a subarea is adjacent to the subareas of its siblings since it just has information of its children. However, using gray codes as GIDs provides an intelligent way to solve these two difficult problems.

For the example shown in Fig. 2, suppose the shaded grids are eligible grids which satisfy the requirements. Suppose the requirement of acreage is that the returned area should not be less than  $2A$ , where  $A$  is the acreage of a smallest grid. At step 1, the grids in each group ( $\%x, \%x$ ) are merged. For example, the result of the group (00x, 00x) is (00x, 001) which means the two lower grids are involved, and the result of the group (11x, 11x) is (11x, 11x) which means all the grids in this group are involved. At step 2, the grids in each group ( $\%xx, \%xx$ ) are merged. For example, the result of group (1xx, 0xx) is (1x1, 0x1), and this is an isolated area since it is not adjacent to the border of the group (1xx, 0xx). That is, only the grids adjacent to the border of a group have chances to be merged with other grids in other groups at the next step. We can infer the following rule. At step  $i$ , the grids whose last  $i - 1$  bits are all 0's on  $X$  or  $Y$ , are on the border of a group. Once an isolated area is found, it is sent to Base Station instead of its parent if it is large enough. Otherwise, it is dropped. As a result, the amount of transmitted data along the reporting tree can be reduced significantly. Thus, (1x1, 0x1) is sent to Base Station since its size is  $4A$ , and (011, 111) is dropped since its size is less than  $2A$ . The other two shaded subareas, ((00x, 001), (0x1, 01x)) and ((11x, 11x), (101, 111), (1x1, 101)), are sent to the leader in the grid (010, 110) which is the root of the tree, because they have chances to be merged. Now we give another rule for merging. At step  $i$ , only GIDs, whose last  $(i - 1)$ -th bit is 1 and last  $i - 2$  bits are all

0's on  $X$  (or  $Y$ ), might be merged on  $X$  (or  $Y$ ). For example, at step 2, grids in ( $\%1, \%$ ) and ( $\%, \%1$ ) might be merged; at step 3, grids in ( $\%10, \%$ ) and ( $\%, \%10$ ) might be merged. This rule is important for reducing the complexity of merging computation. The properties of gray codes and these two rules are crucial for efficiently solving these two difficult problems mentioned above.

The merging algorithm is described in Algorithm 2. After receiving reports from its children, a non-leaf node invokes this algorithm to merge the subareas with the same GID and sends the merged results to its parent. In this algorithm, if  $L_0$  and  $L_1$  are ordered by  $Y$ -GID in a gray code order when we try to merge on  $X$ , the complexity can be reduced since we try to find GIDs with intersection on  $Y$ . XNOR (the inverse

---

**Algorithm 2: Merging( $i$ , GID lists from its children)**


---

**Input:** Step  $i$  and subareas from children.

**Output:** Merged results.

---

```

1: for each grid in the reporting messages do
2:   if the last  $(i - 1)$ -th bit of GID on  $X$  is 1 and last  $(i - 2)$ 
   bits are 0 then
3:     Add this GID into merging list  $L_0$  if the  $i$ -th bit is 0,
   otherwise add it to  $L_1$ .
4:   end if
5: end for
6: for all pairs of GID1 in  $L_0$  and GID2 in  $L_1$  do
7:   if GID1. $Y \oplus$  GID2. $Y == 1$  then
8:     if (GID1. $Y ==$  GID2. $Y$ ) AND (GID1. $X ==$  GID2. $X$ 
   except the last  $i$ -th bit) then
9:       Merge GID1 and GID2 into one GID by
   replacing 0 and 1 on the last  $i$ -th bit with  $x$ .
10:    end if
11:    Merge two GID lists (to which GID1 and GID2
   belong) into one and Calculate aggregations.
12:  end if
13: end for
14: Merge subareas on  $Y$  coordinate using the similar method
   from line 1 to 13.
15: for each subarea  $A$  do
16:   if GIDs of this Area list have no intersection with ((%
    $\underbrace{0 \dots 0}_{(i-1)\text{bits}}, \% \cup (\%, \% \times \underbrace{0 \dots 0}_{(i-1)\text{bits}})$ ) then
17:     if Size of  $A$  does not satisfy the condition then
18:       Delete  $A$ .
19:     else
20:       Send this result  $A$  to Base Station and delete  $A$ .
21:     end if
22:   end if
23: end for
24: Send merged subareas to its parent.

```

---

of the exclusive OR operation) denoted as  $\overline{\oplus}$  is used to judge whether two GIDs have intersection on  $Y$  or  $X$ . We define the result of  $x\overline{\oplus}\omega = 1$ , where  $\omega$  is 0, 1, or  $x$ . The acreage of a GID is  $2^n \times A$ , where  $n$  is the number of  $x$ 's in the GID. The acreage of an area or subarea is the sum of the acreage of all GIDs in its GID list. Acreage calculation is also simplified by using GIDs. The aggregation function sum and avg of an area are calculated by Eqs. (1) and (2) respectively.  $V_i$  and  $A_i$  are the sensed value and acreage of grid  $i$  respectively.

$$\text{sum} = \frac{\sum_{i=1}^n V_i \times A_i}{\sum_{i=1}^n A_i} \quad (1)$$

$$\text{avg} = \frac{\sum_{i=1}^n V_i \times A_i}{n \times \sum_{i=1}^n A_i} \quad (2)$$

### 3.4 Incremental result update

In most applications, sensed values only change slightly or mostly remain unchange over a long time. Therefore, consecutive merged results are extremely similar for a continuous query and incremental updates could conserve a large amount of energy. By comparison, regenerating query results at intervals using Algorithm 2 will quickly deplete energy because of the heavy communication cost.

We now present an incremental updating scheme to maintain the merged results on non-leaf nodes of the reporting tree after the initial results are calculated. Each sensor node stores its previous report, and non-leaf nodes store previous merged results. We classify changed subareas into three classes, *only value*, *positive*, and *negative*. For a subarea in the previous report, if only sensed values or aggregation values are changed, it is an *only value* subarea; for a subarea not in the previous report but in the current report, it is a *positive* subarea; for a subarea in the previous report but not in the current report, it is a *negative* subarea. The incremental updating process also updates the merged results along the reporting tree step by step. For a *negative* subarea, a merging node removes the dropped grids. A subarea might be split if some grids are dropped. For a *positive* subarea, it is merged into the previous results by using the similar method

described in Section 3.3. For *only value* subareas, aggregation values are updated.

A leaf node does not send a report if its readings are not changed or both previous and current readings do not satisfy the WHERE conditions. A non-leaf node generates an update report by comparing new merged results with the previous results and sends it to its parent. An update report includes *positive* and *negative* subareas and new aggregation values for other subareas if these values are changed. If the size of an updated report is greater than the merged results, we still send *regular* merged results rather than the updated report to save energy. The incremental merging algorithm is described in Algorithm 3. In line 34, for generating an update report, we will generate a positive GID list for the grids in  $R_{\text{curr}}$  but not in  $R_{\text{pre}}$  and a negative GID list for the grids in  $R_{\text{pre}}$  but not in  $R_{\text{curr}}$ . For each positive and negative report, we also include the difference of the aggregation functions.

Our scheme also can be used to answer non-area queries via performing reporting along the tree without merging areas.

### 3.5 Advantages of using gray codes

Communication cost is a significant guideline to evaluate the efficiency of an algorithm in wireless sensor networks. Gray codes are used to represent grids in our scheme to reduce data communication. If we use 14-bit gray codes, i.e., 7 bits for  $X$  and 7 bits for  $Y$ , 16384 (that is  $2^7 \times 2^7$ ) grids can be represented. However, we cannot use binary representation since  $x$  (either 0 or 1) is also used in GIDs. Hence, 14-bit ternary (base 3) representation is used to represent GIDs. Due to the fact that  $2^{24}$  is greater than  $3^{14}$ , 3 bytes (i.e., 24 bits) are enough to express a GID after converting it to the binary representation. Another effective method for representing a grid is using two vertices. Each vertex is stored as a pair of  $(x, y)$  coordinate. If we use 2 bytes for  $x$  and  $y$  coordinates respectively, 8 bytes are needed to represent a grid. Eight bytes are quite long compared with 3 bytes of a GID. For a merged area, GID list description is even more efficient than vertex description. For instance, we can just use two GIDs to describe the shape of the merged area as shown in Fig. 3. For the shaded area in the top-left corner in Fig. 2, we just use  $(00x, 001)$  and  $(0x1, 01x)$  to describe it. If we use vertex description for this area, we need to record the coordinates of all eight

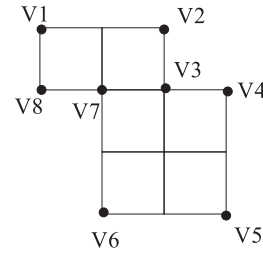
**Algorithm 3: Incremental merging(Node<sub>*i*</sub>, GID lists from its children)****Input:** The current Node<sub>*i*</sub> and reports from children.**Output:** Incremental merged reports for its parent.

```

1: if Nodei is a leaf node then
2:   if Both previous and current sensed values satisfy WHERE
   conditions then
3:     if Current and previous sensed values are different on aggregated
   attributes of the query then
4:       Generate only value reports via calculating the difference
   between current and previous sensed values for each aggregated attribute
   and sent it to Nodei's parent.
5:     else
6:       Don't send anything.
7:     end if
8:   else
9:     if Previous sensed values satisfy WHERE conditions, but current
   sensed values not then
10:      Send a negative report including GID of Nodei to its parent.
11:    end if
12:    if Current sensed values satisfy WHERE conditions, but previous
   sensed values not then
13:      Send a positive report including GID of Nodei and sensed
   values to its parent.
14:    end if
15:  end if
16: else
17:   if No report from any child then
18:     Don't send anything.
19:   else
20:     Copy the previous merged results  $R_{pre}$  to  $R_{curr}$ .
21:     for each positive GID or GID list received from children do
22:       Merge it to  $R_{curr}$ .
23:     end for
24:     for each negative GID or GID list received from children do
25:       Delete it from  $R_{curr}$ .
26:     end for
27:     for each regular merged result from child C do
28:       Delete the previous saved merged results of child C from
 $R_{curr}$ .
29:       Merge new received regular merged result of C to  $R_{curr}$ .
30:     end for
31:     for each value only report do
32:       Use it to update aggregation functions of  $R_{curr}$ .
33:     end for
34:     Compare  $R_{pre}$  and  $R_{curr}$  to generate update reports  $R_{update}$ .
35:     if the size of  $R_{update}$  is less than  $R_{curr}$  then
36:       Send  $R_{update}$  to the parent.
37:     else
38:       Send  $R_{curr}$  to the parent as a regular report.
39:     end if
40:   end if
41: end if

```

vertices. Since our partition method always divides the monitored area into small rectangles, the merged area can be described via every other vertex. For example, in Fig. 3, V1, V3, V5, and V7 are good enough to describe this shape. Therefore, 16 bytes are needed. If we use GIDs for this same area, only 6 bytes are needed for two GIDs. Obviously, GID list area description incurs smaller sizes of the merged results for the same

**Fig. 3 Vertex area description.**

subarea.

Furthermore, GID-based method does not lose location information, although GIDs do not maintain these information. Location information can be retrieved from Base Station. So, GID-based method is the most effective approach to reduce the size of grid description.

In our merging algorithm, due to the properties of gray codes and two merging rules we found, it is easy to detect an isolate area as early as possible. Consequently, useless data can be dropped early, and the results can be sent to Base Station without further merging. Moreover, adjacent subareas can be merged to reduce the size of GID list. In summary, all these strategies are effective methods to reduce the number of messages and the size of the messages. However, it is very difficult to achieve these benefits by using other methods such as vertex description.

**3.6 Area query used for event detection**

Area query also can be used for event detection. For example, the following area query can be used to detect the gas leak event in a coal mine.

```

SELECT area, area.avg(sensors.gas_density)
FROM sensors
WHERE sensors.gas_density > 0.25
GROUP BY adjacency
HAVING area.size > 5 m2
DURATION ∞
EVERY 200 seconds

```

This query can detect any gas leak area which has gas density greater than 0.25 and area size greater than 5 m<sup>2</sup>. The query will return the results every 200 seconds. For event detection, usually no result will be returned. If there are results returned, it means the event happens. Then, depending on the property of the event, some proper actions might need to be carried out. For emergency or severe events such as fire or gas leak, the system can shorten time interval of

queries automatically to return real time information of the monitored area. Thus, real time continuous query results can be expected. When emergency and severe events happen, these useful information are very valuable for rescuing, evacuation, and reducing property damage.

## 4 Simulation Results

### 4.1 Simulation setup

We simulate the area query of coal mine mentioned in Section 2. The payload size limit of a packet is 29 bytes which are the standard payload size provided by the TinyOS<sup>[15]</sup>. The transmission range of sensor nodes is 30 m, and the sensing range is 15 m.  $P_t$  is 15 m. DSDV<sup>[14]</sup> is used as the routing protocol.

Since *network traffic* greatly affects energy efficiency, we use it as the metric for performance evaluation. The network traffic is defined as the total number of messages transmitted (sent and forwarded) by all the nodes in the network during the execution of a query.

We compare the performance of our scheme, IPGID for short, with two other approaches: (1) Centralized Processing (CP), and (2) In-network Processing based on Vertex description (IPV). For a fair comparison, both CP and IPV use conditions on attributes to filter useless readings. That is, a sensor node does not send a report if its readings do not satisfy the WHERE conditions. IPV is similar with our scheme except using an area's vertices to represent it instead of a GID list. In-network merging techniques are also used on IPV.

We varied several system parameters when comparing the performances of the three approaches. These parameters are listed as follows:

- (1) Length of a GID ( $L$ ). This parameter affects network diameter and the number of deployed sensor nodes. Longer  $L$  indicates bigger network and more deployed sensor nodes.
- (2) Selective rate on attributes ( $S$ ). It is the percentage of sensors' sensed values which satisfy the WHERE conditions.
- (3) Filtered rate of area ( $F$ ). It is the percentage of areas which satisfy the HAVING conditions.
- (4) Changed rate of sensed values ( $C$ ). This parameter is the percentage of changed results compared with the previous set of results.

### 4.2 Efficiency of our scheme

As shown in Figs. 4-8, our IPGID is always the best one, and IPV is better than CP. The reason is that

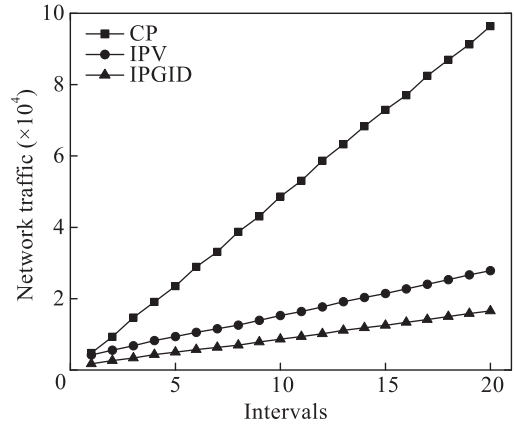


Fig. 4 Network traffic of execution intervals ( $L=10$  bits,  $S=50\%$ ,  $F=50\%$ ,  $C=20\%$ ).

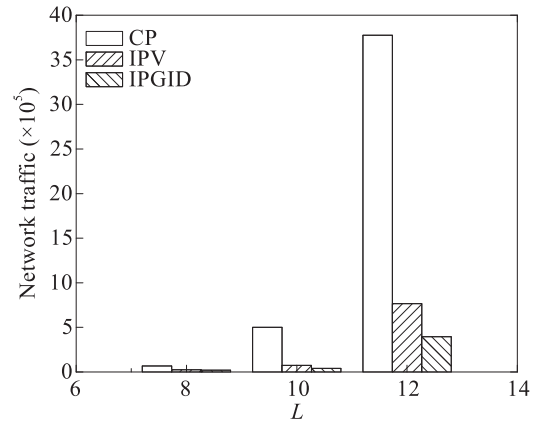


Fig. 5 Network traffic of variant  $L$  ( $S=50\%$ ,  $F=50\%$ ,  $C=20\%$ , 100 intervals).

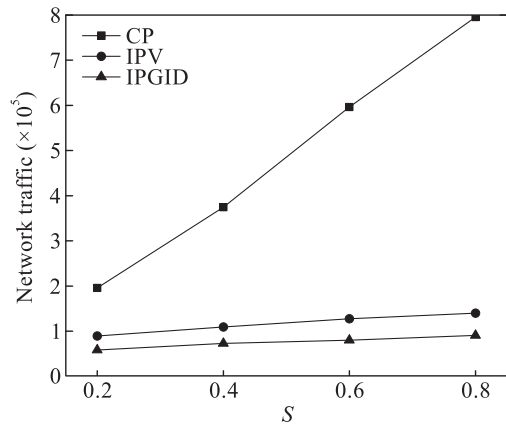


Fig. 6 Network traffic of variant  $S$  ( $L=10$  bits,  $F=50\%$ ,  $C=20\%$ , 100 intervals).

since IPGID and IPV use the incremental updating techniques to maintain results, only the difference between the current and previous set of results are reported. However, CP reports qualified data every interval. IPGID is better than IPV due to the fact that

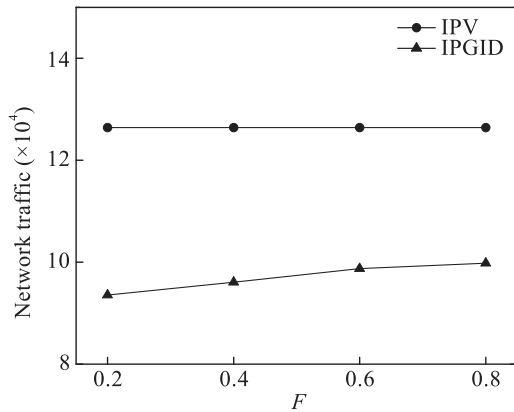


Fig. 7 Network traffic of variant  $F$  ( $L=10$  bits,  $S=50\%$ ,  $C=20\%$ , 100 intervals).

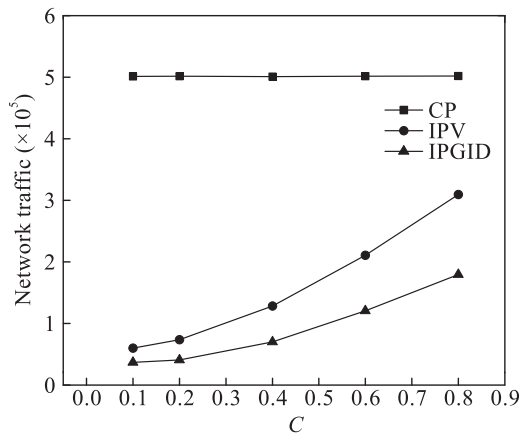


Fig. 8 Network traffic of variant  $C$  ( $L=10$  bits,  $S=50\%$ ,  $F=50\%$ , 100 intervals).

the size of a GID list is much smaller than that of vertex information for describing an area.

Figure 4 shows the network traffic of the first 20 intervals. At the first interval, there's only a marginal difference among the three approaches since both IPV and IPGID construct the initial query results in the first interval. Later on, a clear difference between CP and IPV (or IPGID) presents. The reason is that the number of the reported messages is reduced after the first interval since IPV and IPGID use incremental update techniques. At the 20th interval, IPGID has 83% fewer transmitted messages than CP. IPV has 71% fewer transmitted messages than IPV.

As shown in Fig. 5, with the increasing of  $L$ , the advantages of IPGID becomes more obvious. Although a longer GID incurs more messages, merging multiple reports and filtering useless data in the reporting tree as early as possible can reduce the number of messages, which is the method used by IPV and IPGID. However,

the reporting paths of CP become longer as the network size becomes bigger. Thereby, on average, IPGID has 84% fewer transmitted messages than CP. IPGID has 37% fewer transmitted messages than IPV.

Figure 6 shows the network traffic for variant  $S$ . The network traffic increases with  $S$  because bigger  $S$  causes more sensor nodes to report readings. CP is affected by  $S$  more obviously than IPV and IPGID. The reason is that for CP, the involvement of one more sensor node causes a long message transmitting path to Base Station. However, for IPV and IPGID, it just causes a message to its parent. The network traffic for variant  $F$  is shown in Fig. 7. Since smaller  $F$  implies that more areas can be dropped in the reporting tree according to the HAVING conditions, the smaller the  $F$ , the less the network traffic for IPGID. IPV and CP are not affected by this parameter since IPV and CP cannot filter the useless areas. The network traffic for variant  $C$  is presented in Fig. 8. With the increasing of  $C$ , the task of incremental updating becomes much heavier. As a result, network traffic increases with  $C$ . When  $C$  is close to 1, IPGID and IPV almost have to reconstruct the merged results every interval. IPV is worse than IPGID since IPV uses more messages to describe a big area than IPGID.

In summary, our in-network area query processing scheme is energy-efficient in various circumstances.

## 5 Conclusions

In this paper, we have proposed an energy-efficient in-network area query processing scheme. Our approach is the first work to study area query processing in wireless sensor networks. We define the area queries and partition the network into grids. Gray code is employed to express the ID of grids. Initial query results are generated by merging partial results along the reporting tree. For conserving energy, an incremental updating strategy is used to generate continuous query results. The size of results can be reduced by using binary GIDs to describe areas. In-network processing can drop useless data as early as possible. Furthermore, incremental updating avoids unnecessary reports. All these details fulfill the achievement of energy efficiency. Our simulation results confirm it. Our future work will focus on how to perform multiple area queries efficiently and effectively.

**References**

- [1] Chu D, Deshpande A, Hellerstein J M, Hong W. Approximate data collection in sensor networks using probabilistic models. In: Proceedings of the 22nd International Conference on Data Engineering, 2006: 48.
- [2] Silberstein A, Braynard R, Filpus G, Puggioni G, Gelfand A, Munagala K, Yang J. Data-driven processing in sensor networks. In: Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research, 2007.
- [3] Wang D, Xu J, Liu J, Wang F. Mobile filter: Exploring migration of filters for error-bounded data collection in sensor networks. In: Proceedings of IEEE 24th International Conference on Data Engineering, 2008: 1483-1485.
- [4] Wang C, Ma H, He Y, Xiong S. Approximate data collection for wireless sensor networks. In: Proceedings of 2010 IEEE 16th International Conference on Parallel and Distributed Systems (ICPADS), 2010: 164-171.
- [5] Madden S, Franklin M J, Hellerstein J M, Hong W. Tag: A tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 2002; **36**(SI): 131-146.
- [6] Sharaf A, Beaver J, Labrinidis A, Chrysanthis K. Balancing energy efficiency and quality of aggregate data in sensor networks. *The VLDB Journal*, 2004, **13**(4): 384-403.
- [7] Silberstein A, Yang J. Many-to-many aggregation for sensor networks. In: Proceedings of IEEE 23rd International Conference on Data Engineering, 2007: 986-995.
- [8] Goldin D. Faster in-network evaluation of spatial aggregation in sensor networks. In: Proceedings of the 22nd International Conference on Data Engineering, 2006: 148.
- [9] Yao Y, Gehrke J. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 2002, **31**(3): 9-18.
- [10] Madden S R, Franklin M J, Hellerstein J M, Hong W. Tinydb: An acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 2005, **30**(1): 122-173.
- [11] Abadi D J, Madden S, Lindner W. Reed: Robust, efficient filtering and event detection in sensor networks. In: Proceedings of the 31st International Conference on Very Large Data Bases, 2005: 769-780.
- [12] Xue W, Luo Q, Chen L, Liu Y. Contour map matching for event detection in sensor networks. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, 2006: 145-156.
- [13] Gardner M. The binary gray code. In: Ch. 2 of *Knotted Doughnuts and Other Mathematical Entertainments*. New York, USA: W. H. Freeman & Company, 1986.
- [14] Perkins C E, Bhagwat P. Highly dynamic destination-sequenced distance-vector routing (dsv) for mobile computers. *SIGCOMM Comput. Commun. Rev.*, 1994, **24**(4): 234-244.
- [15] Tinyos faq. Available: <http://www.tinyos.net/faq.html>. March 28, 2012.