Abstract

Energy efficiency is a major design requirement of the sensor network system because current sensor nodes have only finite energy reserves from a battery. Unattended and ad-hoc nature also makes sensor network systems more energy consuming. In order to prevent sensor networks from hazardous conditions due to power depletion, detecting a node death or abnormal working nodes is very important. Residual energy level monitoring can be a underlying application which indicates the sensor network health. eScan is one of remaining battery power monitoring infrastructures. It shows abstracted view of residual energy level distribution over sensor networks trading-off data accuracy and system lifetime. In this study, I implemented eScan in TinyOS on Mica2 motes since eScan was simulated in C++ packets only. Some empirical issues were found during implementing such as accurate battery voltage, routing protocol selection, and packet size. This paper discusses those issues and suggest empirically efficient solutions.

1 Introduction

Energy efficiency is a major design requirement of the sensor network system. Current sensor nodes have only finite energy reserves from a battery. For instance, 2 AA batteries are only power for a MICA2 mote. So it is important to detect node death or unpredictable working before actually sensor network is getting hazardous conditions. It is necessary for almost all sensor network system components such as routing[6], wireless communication[20][11], query processing[9][8] and security[23]. Residual energy level may be one of the most significant resources in order to avoid it.

Unattended and ad-hoc nature makes sensor network systems more energy consuming. Sometimes sensor nodes are deployed in an unplanned fashion. For example, they may be dropped into battle field from an airplane. Thus, if we need energy-aware sensor network systems, continuously updating the sensor network health indications would be critical.

However, there are some challenges to monitor residual power level continuously. All severe constraints such as limited battery life, small memory, a limited bandwidth contribute for impractical extracting of residual energy level of individual node. In addition, an application to monitor and report of residual energy level should use minimum resources because it would work under many other higher-level protocols or applications such as energy-efficient routing protocol, network management applications.

An efficient monitoring infrastructure for sensor networks, eScan, has been proposed by Zaho et al to monitor remaining energy in energy efficient manner. By performing in-network aggregation and providing abstracted view of remaining energy level distribution, they decrease the total number of packets to transmit thus save energy for it. They simulated it using C++ packet and successfully demonstrated its energy saving performance and scalability. This project aims at seeing how eScan works at a real sensor network. As well, it finds out important empirical issues which should be considered when designing and implementing eScan. An experiment to evaluate eScan performance compare total number of packets delivered to a base station when aggregating eScan packets at intermediate nodes with when collecting all local energy level of each sensor node without any in-network processing. Before we discuss eScan implementation, I would like briefly introduce eScan ideas first for better understanding of eScan implementation.

2 eScan Overview

eScan is an efficient residual monitoring infrastructure for sensor networks. To decrease transmission cost, it trades-off system lifetime and data accuracy. In other words, to depicts the remaining energy levels of sensor nodes, eScan performs in-network aggregation using data value similarity and spatial coverage adjacency.

Figure 1 shows an example of residual energy map. As shown, detailed data value is hidden. Instead, it provides abstracted residual energy map like weather map. Darker area means more severe energy depletion area. This map shows that energy depletion of middle left area is serious.
now. Now look at eScan’s key ideas.

2.1 eScan Tuples
A eScan tuple consists of energy value and its coverage, \textless value, coverage \textgreater. Value is a minimum and maximum residual energy level pair in coverage area. Coverage is a polygon region described by value, which is the locations of boundary nodes. For example, look at Figure 2 (c). Nodes covered by Scan A have minimum 35 included in coverage are ones connected with solid lines.

2.2 Constructing a eScan
The process of constructing a eScan of a sensor field can be briefly described as follows:

- **Determining local eScans**: Each sensor generates its own local eScan tuple. As shown in Figure 2, eScan of node x has battery power 35 and checks if the energy level drops significantly since last reported its local eScan. If yes, it updates its local eScan.

- **Disseminating eScans**: Newly generated or updated local eScans are disseminated across the network to compute a composite eScan of the entire network. In [22], the authors use a tree-based routing protocol to do that. Local eScans is sent along the aggregation tree towards a base station.

  - **Aggregating eScans**: Each node also is a router, which receives eScans from other nodes and transmit them to a base station. At this intermediate node, received eScans are combined into one aggregated eScan based on value similarity and spatial adjacency. Tolerance T denotes value similarity threshold as the maximum allowed relative error of residual energy value by aggregation. Resolution R is a spatial proximity threshold. Aggregation conditions are summarized as follows:

\[
\frac{\text{max}(A.\text{max}, B.\text{max}) - \text{min}(A.\text{min}, B.\text{min})}{\text{avg}(A.\text{min}, A.\text{max}, B.\text{min}, B.\text{max})} < T
\]

when value difference between two eScans is below tolerance T and distance between any boundary nodes is less than resolution R, two eScan A and B are aggregated into a new eScan C as this:

\[
\begin{align*}
C.\text{min} &= \text{min}(A.\text{min}, B.\text{min}) \\
C.\text{max} &= \text{max}(A.\text{max}, B.\text{max}) \\
C.\text{Coverage} &= \text{Merge}(A, B, R)
\end{align*}
\]

Minimum value of new eScan C is a min value among A.min and B.min. Similarly, maximum value of new eScan C is max value among A.max and B.max. Coveragy of new eScan C is merged coverage using resolution R. Let’s look at Figure 2. eScan A has minum 35 currently and it is covered by 10 boundary nodes and one inside it which are located at upper part of network. eScan B has minimum 35 energy covered by 9 boundary nodes and three inside it placed in bottom part. Suppose we use two eScans is around 5 covered area by resolution R, that is, 5 meters far from boundary nodes. As shown, two covered areas are intersect. So this two eScans now composited into a new eScan C which has value (35) because the number of boundary nodes for eScan C, 16, is less than sum of boundary nodes of eScan A and B, 19 = 10 + 9, size of eScan C is smaller than sum of size for eScan A and B. Instead of transmitting two eScans, an intermediate node needs to transmit only one smaller eScan C. This is the key point to save transmission cost.

2.3 Contributions of a eScan
The authors in [22] claim eScan monitors residual energy level of each sensor node in energy saving manner and display them as an abstracted view. It is also auto-scaled by varying the spatial resolution R and value tolerance T. Very low resolution and low tolerance may work similar to extracting individual energy level. And with high resolution and tolerance, eScan maps would report average remaining energy level over entire network. In addition, incremental updates when its local state has changed significantly contribute to save more energy. Another is that data aggregation
for eScan is completely in new angle. Usually, aggregation is performed for sensor data from environment. And energy consumption is indirectly computed according to communication and computation cost. But eScan gathers informative data, energy levels, and aggregates them. It may be able to notify to users when additional sensors should be deployed due to serious energy depletion or abnormal working.

3 eScan Implementation

The performance comparison of eScans to centralized collection of individual residual energy information from each node in [22] successfully demonstrates its low transmission cost and scalability. However, it was C++ packets simulation. It is possible that they miss some important empirical design issues when eScans work for real sensor networks. In order to figure out those issues, in this project, I firstly implemented an eScan on TinyOS[18][5], the most popular open-source operating system designed for wireless embedded sensor networks. I used TOSSIM[7], a discrete event simulator for TinyOS sensor networks, to test and debug eScan implementation. Then I installed eScans into Mica2 motes to run an experiment. The Mica2 mote is the second generation of the Mica Mote. Figure 3 shows a Mica2 mote and Table 1 summarizes its spec.

![Figure 3: A Mica2 mote](image)

Following subsections present eScan packet format and implementation key components. Several issues such as determining packet size and eScan dissemination time interval, reading accurate battery voltage, and selecting a proper routing protocol will be discussed together. In order to do all of these, tutorials in TinyOS official website[18] were very helpful.

3.1 eScan Packet Format

eScan packet is formatted as shown Figure 4. type denotes if it is local or aggregated eScan. srcAddr is a source node address and destAddr is a next hop address. Both are

<table>
<thead>
<tr>
<th>Table 1: A Mica2 summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Microcontroller</strong></td>
</tr>
<tr>
<td><strong>CPU Clock (Mhz)</strong></td>
</tr>
<tr>
<td><strong>Program Memory (KB)</strong></td>
</tr>
<tr>
<td><strong>Ram (KB)</strong></td>
</tr>
<tr>
<td><strong>UARTs</strong></td>
</tr>
<tr>
<td><strong>SPI</strong></td>
</tr>
<tr>
<td><strong>I2C</strong></td>
</tr>
<tr>
<td><strong>Nonvolatile storage</strong></td>
</tr>
<tr>
<td><strong>Chip</strong></td>
</tr>
<tr>
<td><strong>Size (KB)</strong></td>
</tr>
<tr>
<td><strong>Radio Communication</strong></td>
</tr>
<tr>
<td><strong>Raid</strong></td>
</tr>
<tr>
<td><strong>Frequency</strong></td>
</tr>
<tr>
<td><strong>Radio speed (kbps)</strong></td>
</tr>
<tr>
<td><strong>Transmit Power Control</strong></td>
</tr>
<tr>
<td><strong>Encoding</strong></td>
</tr>
</tbody>
</table>

16bits long. minV and maxV are minimum/maximum remaining battery voltage respectively. Last four arguments indicate rectangular area covered by given eScan value (minV, maxV); a left bottom location (x1, y1) and right upper location (x2, y2). In original paper[22], coverage consists of all boundary nodes locations. But it is rectangle in this study. It is because rectangle representation is much simpler, save more memory space and reduce much more computational cost. Benefits from rectangular expression will be discussed in 3.2.3 Aggregation eScans part.

![Figure 4: eScan packet format](image)

3.2 eScan Key Components

Figure 5 presents how four main eScan components work together. At every 10 seconds, each node reads its battery voltage (commandBatteryADC.getData() and eventBatteryADC.dataReady()). After generating its own local eScan, it sends it out by calling RadioSend.send() command or posting agg_scans() task in order to aggregate queued eScans with it depending on a queue status. When each node receives eScans from another nodes, they are aggregated and sent out at given time interval.

3.2.1 Reading battery voltage

Reading battery voltage is done via ADCC.ADC[7] channel which is 7th ADC channel its output is battery voltage reference. My interface BatteryADC is wired to that.
Figure 5: Overview of eScan components and their interaction

From ADCC.ADC[7] is not real voltage value, I needed to convert it into voltage value using a following formula given by Mica2 mote manufactures, Crossbos Tehnology, Inc.

\[ V_{out} = \frac{V_{ref} \times ADC_{FS}}{ADC_{output}} \]

\( V_{ref} = 1.223 \) V (battery Vcc voltage)  
\( ADC_{FS} = 1024 \)  
\( ADC_{output}: \) reading from ADC channel 7

Table 2 is pseudo code for BatteryADC.dataReady() event.

Table 2: A pseudo code for BatteryADC.dataRead() event

```plaintext
event result @ BatteryADC.dataReady(uint16_t data) {
    // Convert raw reading into battery voltage;
    // Update if voltage value is dropped more than Tolerance T;
    if (isQNull) Insert a local eScan into a queue;
    else Post agg_escans();
}
```

After converting raw reading into battery voltage, this value updated if it is dropped more than Tolerance T. I used 20a queue before being sent out. There is one problem to use the reading of ADC[7] channel. The value from ADC[7] at every minute was not consistent, very fluctuated from time to time. And accoring to Crossbow Technology, Inc. report for Mica2 AA battery pack service life test[2], battery voltage is dropped from 3V to 2V after 200 hours. To get more accurate battery voltage, we need to average voltage reading for sufficient time interval and mesure its change during long period.

3.2.2 Receiving eScans from other nodes

In addition a local eScan, each node as an intermediate node receive eScans from other nodes, too. Received eScans are aggregated with already queued eScans if there are some. Aggregating manner is same as in [22].

Table 3: A pseudo code for RadioReceive.receive() event

```plaintext
event TOS_MsgPtr RadioReceive.receive(TOS_MsgPtr data) {
    if (isQNull) Insert received eScan into a queue;
    else Post agg_escans();
}
```

3.2.3 Aggregating eScans

Task agg_escans() composites two eScans into one new eScan if they have value similarity and spatial adjacency. As shown in Table 4, newly coming eScan X is checked if it has similar value and spatially close to queued eScans. If yes, it is combined and sent out at given time interval. If no queue is already full, new local eScan would be sent out immediately and new received eScan would be dropped.

Table 4: A pseudo code for agg_escans() task

```plaintext
task void agg_escans() {
    if (inT(eScan X, q[i]) and inR(eScan X, q[i]))
        then new eScan Y = aggregate(eScan X, q[i]);
    else{
        if (isQFull and X is a local eScan)
            then call RadioSend.send();
        else if (isQFull and X is a received eScan)
            then Drop received eScan X;
    }
}
```

Function inT, which evaluates value similarity, returns true when two eScan X and q[i] satisfy equation X.X in Subsection 2.2. Function inR tests coverages of two eScan are overlap or not. Since coverages are rectangles, overlap test is very simple as shown in Figure 6. The condition to overlap two rectangles is that at least one of rectanle B’s x-coordinates is between retangle A’s two x-coordinates and at least one of B’s y-coordinates is between A’s two y-coordinates. As I mentioned before, coverage test can be done using this condition when we use rectangle expression to describe boundary nodes. In contrast, [22] uses all location of boundary nodes. Since they don’t limit the number of boundary nodes and the boundary is not convex, computation complexity of spatial coverage test would increase proportional to the number of nodes. Thus, to decrease computation cost as well as to reducing
packet size, rectangle expression would be more appropriate for eScan implementation.

### 3.2.4 TinyOS Beaconing as a routing protocol

Multi-hop routing for my eScan implementation is built in TinyOS Beaconing. Although a eScan is a location associated packet and it is aggregates based its coverage, [22] uses a tree-based routing; a base station floods INTEREST message through the network and once sensor node X receives the message from node Y, X indicates Y as a parent. It does not fully utilize resolution R for its performance. For example, if its parent is located further than R, no aggregations may be performed in worst case. For this reason, built-in Beaconing routing may work more appropriately in order to maximize impact of in-network processing. Here is one assumption to use it. Each node has its own location information a prior because devices such as GPS is too expensive to attached small sensors.

### 4 Performance Evaluation

We compare eScan performance and centralized collecting residual energy level of each individual sensor node. To do that, we measure the total number of packets which a base station delivers to PC through serial port. Eight sensor nodes read their battery voltage, generate local eScans and transmit them through the network. Each node is also router which aggregate receiving eScans and deliver to one base station. 20% tolerance and 5 meters resolution are used in this experiment. Aggregation with 10% tolerance is the value which saved messaging costs by a factor of 12.5 and 5% distortion only in [22]. Resolution 5 meters is selected arbitrarily in this experiment because [22] doesn’t mention about detailed resolution R value.

My primary challenges in the experiment was accurate remaining energy level from ADC 7th channel. As I discussed earlier, reading from ADC 7th channel was not consistent for short period. It would be better that getting average voltage for long period, for instance, 1 hour, and see the changes. Second challenge is the building energy deple-

Experiment result is summarized at Figure 8. This graph shows the total number of eScan packets delivered to PC from base station for 180 seconds. As already expected, without in-network processing, the number of packets are increased very sharply comparing with the number of packets using eScan schema.
5 Related Work

Energy-aware protocols are necessary at all layers of networking protocols stack because limited energy is a major challenge to the design and management of sensor network. Energy aware MAC protocols, low duty cycle, routing for energy-aware load balancing are examples. GEAR[21] is a Geographic and Energy Aware Routing protocol proposed by Yu et al. It uses energy aware neighbor selection to route a packet towards the target region. Also it recursively forwards packets in geographic manner and restrictedly flood packets inside the destination region. GEAR considers remaining energy as well as geographic location to avoid quickly draining energy of the node closest to the destination. LEACH[4], proposed by Heinzelman, is a clustering based hierarchical routing protocol. As a low-energy adaptive clustering hierarchy, LEACH utilize randomized rotation of local cluster heads. It also balances energy evenly thus increases system lifetime. Shah et al also investigate to find minimum energy path to optimize energy usage at a node in their paper "Energy Aware Routing for Low Energy Ad Hoc Sensor Network".[14] They use one of the multiple paths with a probability depending on the energy consumption of each path. It is important because using the minimum energy path all the time will deplete the energy of nodes on that path.

PAMAS[16] is an example of power-aware media access protocol for ad-hoc radio networks. It conserves battery power at nodes by intelligently powering off nodes that are not actively transmitting or receiving packets. Nguyen et al propose an energy-aware contention-based protocol for the medium access control (MAC) sub-layer for wireless sensor networks. [12] To solves the energy efficiency cause by idel listening, control-packet overhead, and overhearing, it keeps frequent sleep mode proactively and aggressively reduces the wasted energy while increasing the throughput and decreasing the latency. Wireless Integrated Network Sensor (WINS)[13] trades local processing for transmission. It focuses on power control to reduce energy consumption then uses the benefits of short-range transmission in order to maximize power conserving. S-MAC[19] also reduces the wasted energy by periodic listen and sleep. Pointing out collision, overhearing, control packet overhead and idle listening as major sources of energy waste, S-MAC avoids collision and overhearing by setting the radio to sleep during transmissions of other nodes. In addition, store-and-forward processing reduces contention latency. It is implemented over a Mote and shows more energy saving than 802.11-like MAC. One problem of this approach is topology is too simple, just two-hops.

Many researchers have been studied energy-efficient continuous monitoring in wireless sensor network. One of solutions to reduce continuous monitoring cost is Event Contour by Meng et al[10]. It is proposed as an efficient data-collection scheme that can be used for event monitoring and network-wide diagnosis. Based on contour maps, which trade off accuracy with the number of samples, Event Contour utilizes both temporal/spatial suppression. It reports sensed data when value is significantly changed. Then a node overhears readings that originated at its neighbors and average them. then when the difference between the average and its own data, a node transmits its data. Thus the number of transmissions decreases required to convey relevant information to the base station. It is similar to eScan but it doesn’t every sensor to communicate. since the base station knows the location of each sensor and the base station constructs the contour map. Deshpande et al propose BBQ[3], which is probabilistic modeling techniques to optimize data acquisition for sensor network queries. It is pull based using correlation models to satisfy queries with a minimum of data acquisition. BBQ answers approximately SQL queries by consulting a correlation-aware probabilistic model which built from historical readings and improved from current readings. It reduces the total number of expensive sensor readings as well as radio transmissions. Coverage issues is another important problem for continuous monitoring. Abrams et al[1] propose an efficient and distributed algorithm that partitions the sensors in a wireless sensor network into K covers such that as many areas are monitored as frequently as possible. By activating groups of sensors in rounds, the battery life of a sensor is not wasted on areas that are already monitored by other sensors. This paper also provides maximum coverage while balancing energy consumption between sensors and given a desired lifetime of the network.

TinyOS[18][5] is an open-source operating system designed for wireless embedded sensor networks. It is developed by University of Berkeley and tightly coupled with the application. Thus many research groups[17] are using it on the Crossbow motes such as mica2 motes which I used to implement eScan in this study. As a component-based event driven system, it aims at minimizing code size, holding small footprint, lowering system overhead and power consumption. Non-preemptable FIFO task scheduling contributes for small space and low power. Function call(event, command) interface between components also is conducive to efficient modularity.

6 Conclusions

A wireless sensor network is a distributed computer network consisting of many micro sensors. Such a sensor enables to monitor environment conditions at different locations, such as temperature, sound, vibration, pressure, motion or pollutants. However sensor nodes have severe resource constraints in terms of energy, memory, computational speed and bandwidth. This has posed many challenges to the design and management of sensor network. Having ability to aware energy across sensor networks has provided big benefits in order to construct efficient sensor network systems. eScan is one of very simple and efficient
infrastructure to monitor the residual energy resource distribution within a sensor field by performing in-network processing. By providing abstracted view of energy distribution, theoretically it reduces the message cost and it is very scalable.

Implementation eScans on TinyOS is presented and observes its performance in a real sensor network in this paper. I found some empirical problems to design and implement eScans on TinyOS; hard to read accurate battery voltage, memory consuming boundary node describing, and inefficiency of tree-based routing. To improve eScan implementation, I suggested and used currently available approaches such as average voltage reading for sufficient time interval, describing coverage as a rectangle in memory saving manner, utilizing a location-based routing protocol for more effective aggregation.

To extend this work, I would like to implement visualization of eScan representation using JAVA packets. Then experiment to see the performance of the energy consumption could be conducted in detail. eScan may be embedded as a complementary to other sensor network application to identify particular network problems within particular region.

References


