

TIME: Time-based Index Management for Event Query Processing in Wireless Sensor Networks

Guilin Li Jianzhong Li
Department of Computer Science
Harbin Institute of Technology, China
{liguilin, lijzh}@hit.edu.cn

Yingshu Li
Department of Computer Science
Georgia State University
yli@cs.gsu.edu

Abstract

Data centric storage is an effective algorithm to organize data in the sensor networks. Even though such kind of storage can save more energy than other algorithms, such as flooding based algorithm, it also wastes energy in some cases. For example, the event frequency is much more than the query frequency. To overcome this problem, we propose an index based query processing algorithm. We analyze the energy consumption of the algorithm and present conditions for the index based algorithm to save more energy than the traditional data centric storage. Finally a time-based index management algorithm is proposed to adopt the proper algorithm in different cases. Extensive experiments showed that our time-based index management algorithm can save more energy than the traditional data centric algorithm.

1. Introduction

Event monitoring is one of the most important applications [1], [2] for Wireless Sensor Networks (WSNs). In these applications, sensors are randomly distributed in a target area to detect events and users issue queries, called event based queries, to collect event information of their interest. As energy is the most important resource for WSNs, event based query processing should be executed in an energy efficient way.

Two kinds of algorithms are proposed for event based query processing, which are the flooding based algorithms [3], [4] and the Data Centric Storage (DCS) algorithms [5], [6]. In the flooding based algorithms, events detected by a sensor are stored locally. Since an event can happen anywhere in the network and therefore the event can be stored anywhere, a query must be flooded all over the network to retrieve results. Flooding consumes much energy and should be avoided.

The DCS algorithms are proposed to overcome the short-

coming of the flooding based algorithms. Compared with the flooding based algorithms, the DCS algorithms organize the storage of event information in an efficient way so that a query can be directly answered without flooding it to the whole network. In a DCS algorithm, a special node called hash node is selected to store all the events of a particular type and answer queries asking for this type of events.

Even though the DCS algorithms avoid flooding in WSNs, in some cases, energy is still wasted. For example, some sensors are deployed to a bridge to monitor the traffic. Each vehicle going across the bridge can be denoted as an event. For traffic condition monitoring, the query 'How many vehicles in total went across the bridge during 9am to 10am?' would be more popular than the query 'Which vehicle went across the bridge at 9am?'. In this case, the sensors continuously monitor the bridge and detect events frequently. Users do not care about each individual event but are more interested in the statistical information about events happening in this period of time. Aggregate queries are therefore more popular. In this case, the query frequency is much less than the event occurrence frequency during the period in which events happened. If a traditional DCS algorithm is adopted, all the events are transmitted to the hash node, which is unnecessary. To answer an aggregate query, the hash node can save energy by directly obtaining the aggregated result from the source node.

We now use a detailed example to illustrate the case mentioned above. As shown in Fig.1, suppose a source node s_1 frequently detects 50 events during time interval (t_s, t_e) . Queries are issued at node u about events happening in the recent time interval $(t_c - T, t_c)$ and $(t_s, t_e) \cap (t_c - T, t_c) \neq \phi$, where t_c is the current system time, and T is the length of the recent time interval. As t_c increases, (t_s, t_e) moves out of $(t_c - T, t_c)$. If a traditional DCS algorithm is adopted, s_1 transmits all the 50 events to the hash node, which needs to transmit 50 messages. If s_1 only registers the time interval (t_s, t_e) to the hash node, and the hash node sends 10 queries to s_1 for events happened during (t_s, t_e) , only 22 messages are needed. These 22 messages involve 2 messages for reg-

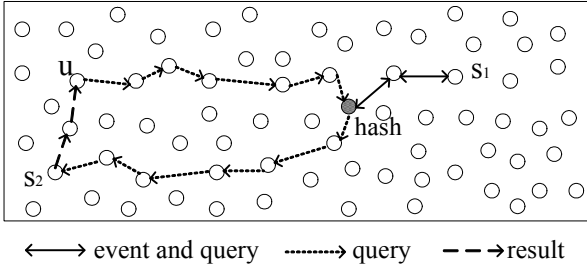


Figure 1. Index based query processing algorithm

istering t_s and t_e to the hash node and 20 messages for sending queries and returning results. The time interval (t_s, t_e) registered by the source node acts as an index entry to guide queries to a proper source node.

The index entry constructed by a source node can also save energy consumption for result collection. In Fig.1, the distance between s_2 and node u where queries are issued is smaller than the distance between s_2 and the hash node. In this case, when the hash node sends a query to s_2 , it can piggyback u 's position in the packet. s_2 can send the result directly to u rather than to the hash node to relay the result to u as usual. As u is closer to s_2 than to the hash node, energy can be saved.

An index based algorithm, however, cannot guarantee to save energy in all cases. When the number of events is less than 22 in the above example, the number of messages transmitted by the index based algorithm is more than that of a traditional DCS algorithm. When this happens, the source node should adopt a traditional DCS algorithm. Therefore, a decision algorithm is needed to adaptively select a proper query processing algorithm for different scenarios.

In this paper, we first propose an index based query processing algorithm, then analyze the condition under which the index based algorithm can save more energy than a traditional DCS algorithm. Based on the condition, a time based index management algorithm is presented, which can adaptively adopt a proper query processing algorithm for different cases. The contributions of this paper are as follows.

- An index based query processing algorithm is proposed.
- The energy consumption of the index based algorithm is analyzed and the condition under which the index based algorithm can save energy is provided.
- A time based index management algorithm is presented to adaptively adopt a proper algorithm for different cases.

- Extensive experiments are performed to compare the energy consumption of the index management algorithm with that of a traditional DCS algorithm.

The rest of the paper is organized as follows. Section 2 discusses the index based query processing algorithm. In section 3, we analyze the energy consumption of the index based algorithm and figure out the conditions under which the index based algorithm saves more energy than a DCS algorithm does. Section 4 presents a time based index management algorithm. Performance evaluation result is shown in Section 5, and we conclude the paper in Section 6.

2. The index based query processing Algorithm

Two kinds of data transmission methods can be used for the communications between a source node and a hash node, which are the active transmission method and the inactive transmission method.

- In a traditional DCS algorithm, once an event is detected by a source node, it is transmitted to the hash node. We call such kind of method as the active transmission method.
- A source node registers a time interval and its position at the hash node, then it stores all the events happened during the interval locally. We call such kind of method as the inactive transmission method.

A traditional DCS algorithm only employs the active transmission method. As all the events are stored at the hash node, a query can retrieve the result directly at the hash node. We propose an index based query processing algorithm which adopts the inactive transmission method. The index based algorithm stores events locally at the source node and stores the time interval acting as an index, during which these events happened, at the hash node. When the query interval intersects with an index entry created by a source node, the query will be sent to this source node to obtain results.

Compared with a traditional DCS algorithm, our index based algorithm saves energy consumption for event transmission, but wastes energy consumption for query processing. So the index based algorithm is more suitable for the case where the event occurrence frequency is much more than the query frequency. While a traditional DCS algorithm is suitable for the case where the event occurrence frequency is much less than the query frequency. In the next section, we analyze the energy consumption of the two kinds of algorithms in detail and formally present the condition under which the index based algorithm saves more energy than a traditional DCS algorithm.

Symbol	description
t_i	starting time of an index entry
ΔT	recorded duration of an index entry
t_c	current system time of WSNs
T	users query events happening in the recent time interval $(t_c - T, t_c)$
N_e	the number of events happening during $(t_i, t_i + \Delta T)$
N_q	the number of queries related to $(t_i, t_i + \Delta T)$ when $(t_i, t_i + \Delta T) \cap (t_c - T, t_c) \neq \phi (t_c \rightarrow \infty)$
h	the number of hops between a source node and a hash node

Table 1. Symbols and their meanings

3. Energy consumption analysis

In this section we analyze the energy consumptions of our index based algorithm and a traditional DCS algorithm respectively. The number of transmitted messages is used as the metric of energy consumption. The notations used in the rest of the paper are listed in Table 1.

For the index based algorithm, the energy consumption is composed of two parts, i.e., the energy used by a source node registering an index entry $(t_i, t_i + \Delta T)$ at a hash node., and the energy used by a hash node sending a query to a source node and obtaining the result. Note that to registering an entry, two messages need to be sent, one for the starting time of an entry, one for the end time of an entry. The energy consumption for the index based algorithm is:

$$E_{inactive} = 2 \times h + N_q \times 2 \times h \quad (1)$$

For the traditional DCS algorithm, as no query is transmitted from a hash node to a source node, the energy consumption during $(t_i, t_i + \Delta T)$ is just related to the number of events happening in ΔT . The energy consumption for the DCS based algorithm is:

$$E_{active} = N_e \times h \quad (2)$$

If $E_{active} > E_{inactive}$, the index based algorithm should be adopted. Based on Formulas (1) and (2), we derive the following condition.

$$N_q < \frac{N_e}{2} - 1 \quad (3)$$

There are two variables in Formula (3), which are N_e and N_q . N_e can be counted as soon as an index entry is created. N_q , however, is unknown until the index entry $(t_i, t_i + \Delta T)$ completely moves out of $(t_c - T, t_c)$, because the index entry may receive queries during the time period $(t_i, t_i + \Delta T) \cap (t_c - T, t_c) \neq \phi$. We need to estimate the value of N_q in advance. Next we give a probability method to estimate N_q .

Before we formally define the problem of estimating N_q , we give some definitions and explain their meanings in Fig.2.

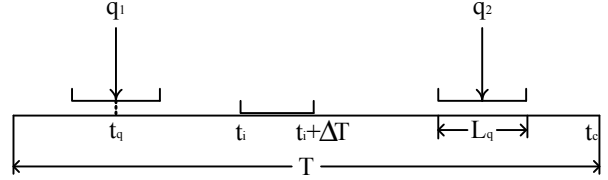


Figure 2. definition explanation

Definition 1 (query time/length of a query interval)

Suppose a query interval is (t_{qs}, t_{qe}) . $t_q = \frac{(t_{qs} + t_{qe})}{2}$ is called the query time. $L_q = (t_{qe} - t_{qs})$ is called the length of a query interval.

Definition 2 (effective query period) Let t_c be the current system time. Users only query events happening in the time period of $(t_c - T, t_c)$, which is called the effective query period.

Based on the above definitions, the problem of estimating N_q can be defined as follows. Suppose the arriving rate of queries follows a poisson process with an average arriving rate λ_q . The query time t_q of each query is uniformly distributed in the effective query period $(t_c - T, t_c)$. The length of a query interval L_q follows a probability distribution function $F(L_q)$. The problem is to calculate the expectation number of queries $E(N_q)$ related to an index entry $(t_i, t_i + \Delta T)$ from the time when it enters the effective query period to the time when it leaves the effective query period.

There are two ways to view the relationship between the effective query period $(t_c - T, t_c)$ and the index entry $(t_i, t_i + \Delta T)$. First, t_i is constant and t_c increases. Second, t_c is constant and t_i decreases. As these two ways are equivalent, we adopt the second one. We consider $t_c - T$ as 0 and t_c as T as shown in Fig.3. At this time, with the decreasing of t_i , the index entry $(t_i, t_i + \Delta T)$ moves from right to left until it moves out of $(t_c - T, t_c)$.

To compute $E(N_q)$, we first calculate the probability that a query q which presents at t_q intersects with an index entry $(t_i, t_i + \Delta T)$. There are three possible cases for t_q , which

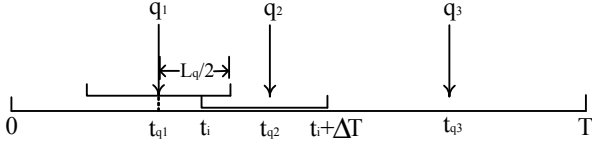


Figure 3. Relationship between query time and index time interval

are $t_q < t_i$, $t_i \leq t_q \leq t_i + \Delta T$ and $t_q > t_i + \Delta T$ as shown by q_1 , q_2 , and q_3 in Fig.3 respectively.

For the first case, as t_q is uniformly distributed in $(t_c - T, t_c)$, the probability of the query lies in a small time interval dt_q is $\frac{dt_q}{T}$. When $t_q < t_i$, if the query interval intersects with the index entry, it must satisfy the condition $\frac{L_q}{2} > (t_i - t_q)$ as shown in Fig.3, whose probability is $Pr\{\frac{L_q}{2} > (t_i - t_q)\}$. The intersection probability of a query interval and the index entry is:

$$\begin{aligned} & \int_0^{t_i} Pr\{\frac{L_q}{2} > (t_i - t_q)\} \frac{dt_q}{T} \\ &= \int_0^{t_i} \frac{F(2(t_i - t_q))}{T} dt_q \end{aligned} \quad (4)$$

For the second case, as t_q lies in the index entry, the query interval definitely intersects with the index entry. The intersection probability of a query interval and the index entry is:

$$\int_{t_i}^{t_i + \Delta T} 1 \frac{dt_q}{T} = \frac{\Delta T}{T} \quad (5)$$

For the last case, the intersection probability can be calculated the same way as for the first case as following:

$$\begin{aligned} & \int_{t_i + \Delta T}^T Pr\{\frac{L_q}{2} > (t_q - t_i - \Delta T)\} \frac{dt_q}{T} \\ &= \int_{t_i + \Delta T}^T \frac{F(2(t_q - t_i - \Delta T))}{T} dt_q \end{aligned} \quad (6)$$

Suppose the probability distribution function for the length of a query interval is

$$Pr\{L_q = 2l\} = 1$$

which means the length of a query interval is always a constant value. l is half of the length of a query interval. According to formulas (4), (5) and (6), we can calculate the probability that a query interval intersects with an index entry from the time when it enters the effective query period to

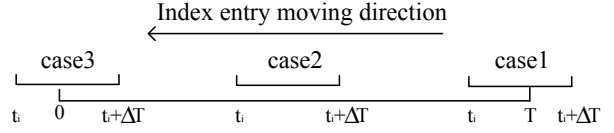


Figure 4. An index entry passing through the effective query period

the time when it leaves the effective query period, as shown in Fig.4.

Case 1: $T - \Delta T - l < t_i$, which means an index entry enters the effective query period. If the query time $t_q < t_i$, according to formula (4), we have

$$\int_{t_i - l}^{t_i} \frac{dt_q}{T} = \frac{l}{T}$$

Otherwise, if $t_i \leq t_q \leq T$, according to formulas (5) and (6), we have

$$\int_{t_i}^{t_i + \Delta T} \frac{dt_q}{T} + \int_{t_i + \Delta T}^T \frac{dt_q}{T} = \int_{t_i}^T \frac{dt_q}{T} = \frac{T - t_i}{T}$$

Now we derive the intersection probability of a query interval with the index entry when the entry just enters the effective query period.

$$Pr_1 = \frac{T - t_i + l}{T} \quad (7)$$

Case 2: $l < t_i < T - \Delta T - l$, which means an index entry has entirely entered the effective query period. If the query time $t_q < t_i$, according to formula (4), we have

$$\int_{t_i - l}^{t_i} \frac{dt_q}{T} = \frac{l}{T}$$

If $t_i \leq t_q \leq t_i + \Delta T$, according to formula (5), we have

$$\int_{t_i}^{t_i + \Delta T} \frac{dt_q}{T} = \frac{\Delta T}{T}$$

Otherwise, according to formula (6) we have

$$\int_{t_i + \Delta T}^{t_i + \Delta T + l} \frac{dt_q}{T} = \frac{l}{T}$$

Now we derive the intersection probability of a query interval with the index entry when the entry has entirely entered the effective query period.

$$Pr_2 = \frac{2l + \Delta T}{T} \quad (8)$$

Case 3: $-\Delta T < t_i < l$, which means an index entry is going to leave the effective query period. If the query time $t_q \leq t_i + \Delta T$, according to formulas (4) and (5), we have

$$\int_0^{t_i} \frac{dt_q}{T} + \int_{t_i}^{t_i + \Delta T} \frac{dt_q}{T} = \int_0^{t_i + \Delta T} \frac{dt_q}{T} = \frac{t_i + \Delta T}{T}$$

Otherwise, if $t_q > t_i + \Delta T$, according to formula (6), we have

$$\int_{t_i + \Delta T}^{t_i + \Delta T + l} \frac{dt_q}{T} = \frac{l}{T}$$

Now we derive the intersection probability of a query interval with the index entry when the entry is going to leave the effective query period.

$$Pr_3 = \frac{t_i + \Delta T + l}{T} \quad (9)$$

As the arriving rate of queries follows a poisson process with average arriving rate λ_q , in any small time interval dt_i , the probability of a source node receiving a query is $\lambda_q dt_i$. According to formulas (7), (8) and (9), the expectation of N_q can be calculated as follows:

$$\begin{aligned} E(N_q) &= \int_{T-l-\Delta T}^T \frac{T-t_i+l}{T} \lambda_q dt_i + \\ &\int_l^{T-l-\Delta T} \frac{2l+\Delta T}{T} \lambda_q dt_i + \\ &\int_{-\Delta T}^l \frac{t_i+\Delta T+l}{T} \lambda_q dt_i \end{aligned}$$

which is

$$E(N_q) = \lambda_q (2l + \Delta T - \frac{l^2}{T}) \quad (10)$$

Recall formula (3), which gives the condition of the index based algorithm saving more energy than the traditional DCS algorithm. Taking the expectation of formula (3), we have

$$\begin{aligned} E(N_q) &< \frac{N_e}{2} - 1 \\ \lambda_q (2l + \Delta T - \frac{l^2}{T}) &< \frac{\bar{\lambda}_e \Delta T}{2} - 1 \end{aligned}$$

where $\bar{\lambda}_e$ represents the average event occurrence rate during ΔT . Furthermore, suppose $\bar{\lambda}_e = i\lambda_q$ where i is some constant number. Then we have

$$\Delta T > \frac{2(2l + \frac{l^2}{\lambda_q} - \frac{l^2}{T})}{i-2} \quad (11)$$

Here we note that i must be greater than 2. Now we identify two requirements for the index based algorithm to save more energy than the traditional DCS algorithm:

- During a time interval ΔT , the average event occurrence rate $\bar{\lambda}_e$ should be larger than two times of the query rate λ_q .
- When $\bar{\lambda}_e$ is i ($i > 2$) times of λ_q , ΔT must be larger than $\frac{2(2l + \frac{l^2}{\lambda_q} - \frac{l^2}{T})}{i-2}$.

4. The time based index management algorithm

In this section we propose a time based index management algorithm according to the two conclusions in Section 3. The main idea is as follows. When the relationship between the average event occurrence rate and the query rate satisfies the two conditions, maintaining an index entry at a hash node can save energy. Otherwise, the index based algorithm consumes more energy. In this case, a source node transmits events to a hash node instead of storing event information locally. By using the index management algorithm, a source node can adopt a proper query processing algorithm at any time. The working procedure of our index management algorithm is as follows.

1. Each sensor has a timer, whose trigger time is half of the time interval between two consecutive queries, that is, $\frac{1}{2\lambda_q}$. When a sensor detects an event for the first time, it transmits the event to a hash node by using a traditional DCS algorithm and starts the timer.
2. If a sensor does not detect any event until trigger time, which means the event occurrence rate is smaller than two times of the query rate, the first condition for the index based algorithm to save energy is violated. The node stops the timer and goes back to the first step.
3. If a sensor detects an event before the trigger time, which means the event occurrence rate is bigger than $2\lambda_q$, the source node adopts the index based algorithm. It sends a register message to a hash node to construct an index entry, which contains the happening time t_s of the event just detected, and restarts the timer again.
4. Afterward, if an event happens before the trigger time, no message needs to be transmitted and the timer needs to be restarted. The new event belongs to the current index entry.
5. When the timer is out before a new event happens, the source node sends a message to the hash node to indicate the end of the current index entry, which contains the occurrence time t_e of the last event.

- A source node judges whether it is beneficial to store the events happened during (t_s, t_e) in its local storage according to formula (11). If the answer is no, the source node needs to transmit all the information about events happened during (t_s, t_e) to the hash node where it registered an index entry.

The algorithm to judge whether it is beneficial to store events locally at a source node is as follows.

- Calculate the average event occurrence rate $\overline{\lambda_e}$ during (t_s, t_e) .
- Given λ_q , calculate ΔT according to formula (11).
- If $(t_e - t_s) \geq \Delta T$, which means the length is long enough for the index based algorithm to save more energy than the traditional DCS algorithm, all the information about the events happened during (t_s, t_e) can be stored locally at the source node.
- Otherwise, the source node transmits all the information about the events happened during (t_s, t_e) to the hash node.

Given a query whose query interval is (t_{qs}, t_{qe}) for a kind of events, the query is first sent to the hash node dedicated to this kind of events. Two types of information are stored at the hash node, which are the event information and the index information. With event information, the hash node can answer the query directly. Index information is useful when the index entry satisfies $(t_s, t_e) \cap (t_{qs}, t_{qe}) \neq \phi$, which indicates the source node stores the answer to the query. In this case, the hash node needs to send the query to the source node. The hash node h also needs to compare the distance from itself to the source node s with the distance from the node q where the query is issued to s . If s is closer to q , h transmits q 's position to s together with the query so that s may directly send the result to q . Otherwise, s sends the result to h and the result will be relayed to q .

5. Performance evaluation

Four factors affect the energy consumption of the index management algorithm: (1) the ratio i between the event occurrence rate and the query rate (2) the length of the time duration in an index entry ΔT (3) the length of the query interval l (4) the length of the effective query period T . Three groups of experiments are conducted to compare the energy consumption of our index management algorithm with that of a traditional DCS algorithm by varying these four factors. The number of messages transmitted is used as the metric of energy consumption.

Fig.5 shows the relationship between i and ΔT according to formula (11) where $l = 5$, $T = 200$, $\lambda_q = 1$. It

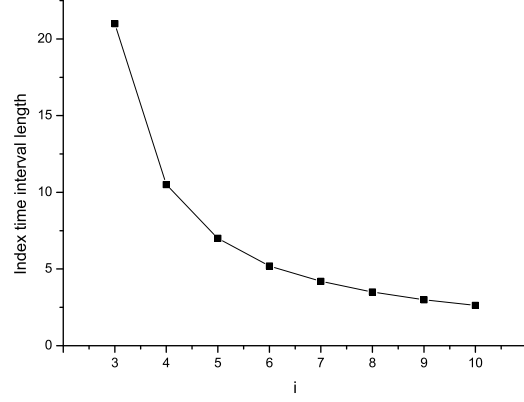


Figure 5. Relationship between i and ΔT

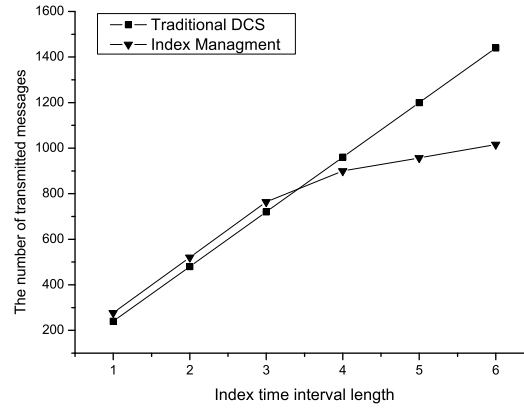


Figure 6. Energy consumption comparison ($i = 8$)

shows when i is close to 2, the index based algorithm needs a long index time interval so that it can save more energy than the traditional DCS algorithm. But with the increasing of i , the length of the index time interval decreases quickly and approaches a reasonable value. The larger the i is, the smaller the ΔT is to satisfy the energy saving requirements.

5.1. The influence of ΔT and i to the energy consumption

In this experiment, we test the influence of ΔT and i to our algorithm. The parameters are set as follows: $T = 200$, $\lambda_q = 1$ and $l = 5$. We set $i = 8$ and 10 respectively and varying ΔT in each case. The results are shown in Fig.6 and Fig.7. When ΔT is small, the index management al-

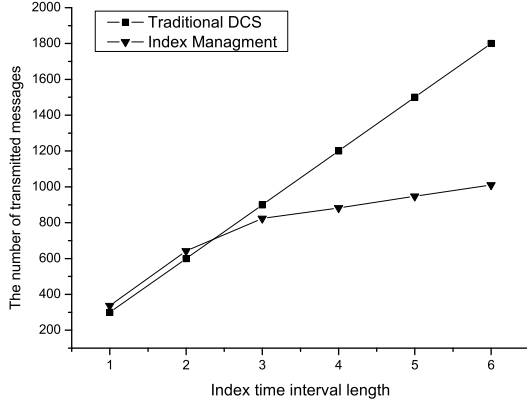


Figure 7. Energy consumption comparison ($i = 10$)

gorithm consumes more energy than the traditional DCS algorithm, because additional register messages are transmitted. On average, the index management algorithm only consumes 8% more energy than the traditional DCS algorithm, because once the index management algorithm identifies such inefficiency, it transmits all event information to a hash node in an adaptive manner. When ΔT is larger than a threshold, the index management algorithm saves much more energy than the traditional DCS algorithm. On average, it saves 28% energy than the traditional DCS algorithm. With the increasing of ΔT , the index management algorithm can save even more energy than the traditional DCS algorithm. The result also shows that when i becomes big, our algorithm needs a small ΔT to save energy.

5.2. The influence of l to the energy consumption

In this experiment, we test the influence of l to our index management algorithm. The parameters are set as follows: $i = 8, 10$, $T = 200$, $\lambda_q = 1$ and $l = 4, 6$. Fig.8 shows the comparison result when $i = 8$. It shows that for the index management algorithm, the larger the l , the more energy it consumes. Meanwhile, it needs a longer ΔT to save more energy than the traditional DCS algorithm. This is because if T does not change, a longer l makes an index entry get involved in more queries. The same explanation applies to the comparison result shown in Fig.9 where $i = 10$.

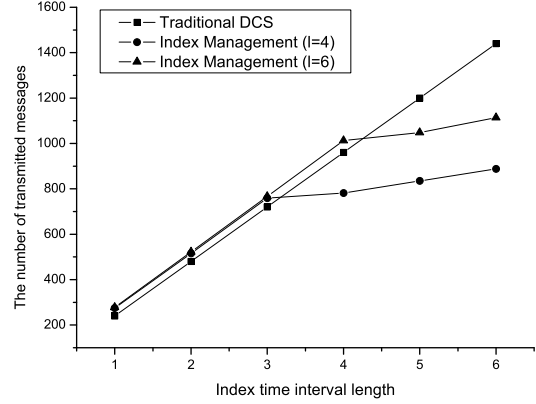


Figure 8. Influence of l ($i = 8$)

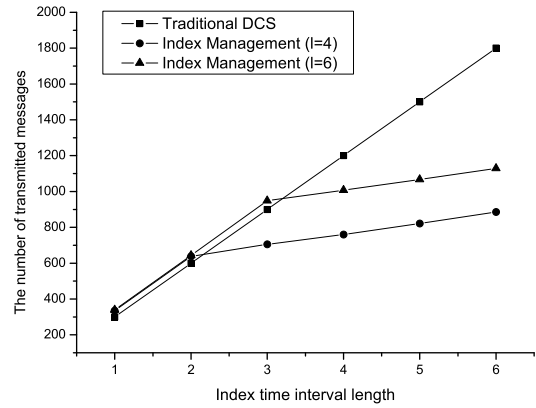


Figure 9. Influence of l ($i = 10$)

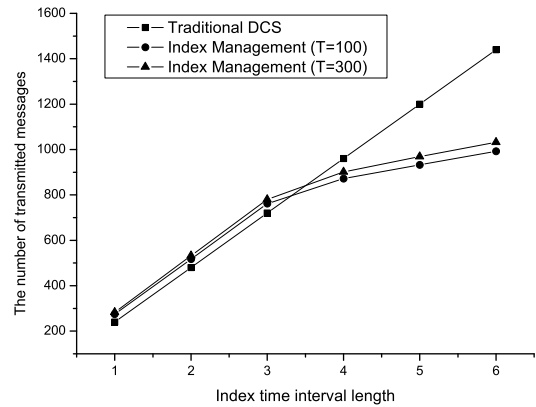


Figure 10. Influence of T ($i = 8$)

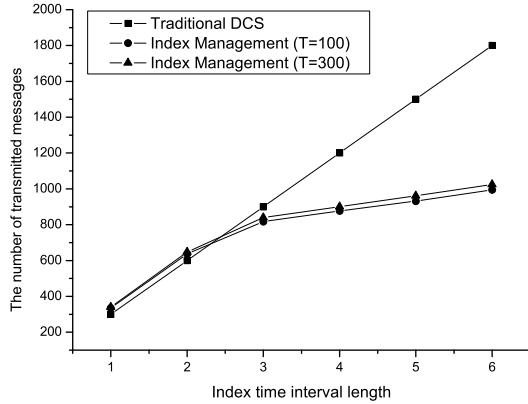


Figure 11. Influence of T ($i = 10$)

5.3. The influence of T to the energy consumption

In this experiment, we test the influence of T to our index management algorithm. The parameters are set as follows: $i = 8, 10$, $l = 5$, $\lambda_q = 1$ and $T = 100, 300$. Fig.10 shows the comparison result when $i = 8$. It shows that for different T , there is not a big difference of the energy consumptions of the index management algorithm. This is because even though a long T increases the time that an index interval stays at the hash node, as l does not change, the opportunity that the index entry being involved in a query decreases. The two factors make the increment of energy consumption small. The same explanation applies to the comparison result shown in Fig.11 where $i = 10$.

6. Conclusion

In this paper, an index based query processing algorithm is proposed and its energy consumption is analyzed. We formally define the problem of energy saving for the index based algorithm and illustrates the conditions under which the index based algorithm can save more energy than a traditional DCS algorithm. Finally a time based index management algorithm is presented. It can adaptively decide a solution in different scenarios. Experiment results show that the index management algorithm can save more energy than the traditional DCS algorithm.

Acknowledgements

This work is supported in part by the US NSF under grants CCF-0545667 and CCF-0844829; the NSFC under grants No.60533110 and No.60703012; the National

Grand Fundamental Research 973 Program of China under grant No. 2006CB303000; Program for New Century Excellent Talents in University under grant No.NCET-05-0333; the NSF of Heilongjiang Province of China under grant No.ZJG03-05; Heilongjiang Province Fund for Young Scholars under grant No.QC06C033.

References

- [1] M. Li and Y.H. Liu, Underground Structure Monitoring with Wireless Sensor Networks, *IPSN*, 2007.
- [2] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An analysis of a large scale habitat monitoring application. *SenSys*, 2004.
- [3] C. Intanagonwivat, R. Govindanj, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. *MobiCOM*, 2000.
- [4] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS*, 2002.
- [5] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin and F. Yu. Data-Centric storage in sensornets with GHT, a geographic hash table. *MONET*, 2003.
- [6] R. Sarkar, X.J. Zhu and J. Gao, Double Rulings for Information Brokerage in Sensor Networks, *Mobicom*, 2006.