Applied Mathematics & Information Sciences
*An International Journal*

http://dx.doi.org/10.12785/amis/070508

# A Maximum Lifetime Algorithm for Data Gathering Without Aggregation in Wireless Sensor Networks

*Junbin Liang*[1,2,*] *and Taoshen Li*[1]

[1]School of Computer and Electronic Information, Guangxi University, Nanning 530004, China
[2]School of Information Science and Engineering, Central South University, Changsha 410083, China

Received: 14 Dec. 2012, Revised: 21 Apr. 2013, Accepted: 23 Apr. 2013
Published online: 1 Sep. 2013

**Abstract:** Data gathering in wireless sensor network (WSN) has attracted a lot of attention in research. Data gathering can be done with or without aggregation, depending on the degree of correlation among the source data. In this paper, we study the problem of data gathering without aggregation, aiming to conserving the energy of sensor nodes so as to maximize the network lifetime. We model the problem as one of finding a *min-max-weight spanning tree (MMWST)*, which is shown to be NP-complete. In MMWST, the maximum weight of the nodes is minimized. The weight of a node in the tree equals the ratio of the number of the node's descendants to the node's energy. We propose a $\Omega(\log n/\log\log n)$-approximation centralized algorithm MITT to construct MMWST without requiring node location information. Moreover, in order to enable MITT to be used in large-scale networks, a new solution that employs the clustering technique is proposed. To the best of our knowledge, MITT is the first algorithm that constructs MMWST in wireless sensor networks. Theoretical analyses and simulation results show that MITT can achieve longer network lifetime than existing data gathering algorithms.

**Keywords:** Maximum lifetime, Data gathering, Wireless sensor networks

## 1 Introduction

A wireless sensor network (WSN) is composed of a large number of low-cost and low-energy (sensor) nodes that communicate with each other through multi-hop wireless links [1]. A basic operation in WSN is data gathering, i.e., each node periodically transmits its sensed data to the sink for further processing. Because nodes are often deployed in remote or inaccessible environments, where replenishing nodes' batteries is usually impossible, a critical issue in data gathering is to conserve nodes' energies and maximize the network lifetime. The network lifetime is defined as the time until the first node depletes its energy [2].

Data gathering can be categorized by the degree of correlation among the source data [3]. For applications of statistical queries in WSN, such as SUM, MAX, MIN, etc., correlation exists among the data sensed by nodes. Therefore, when these data are transmitted, they could be aggregated into one data packet with constant size regardless of the number of data sources. This type of data gathering is called data gathering with aggregation. Data gathering without aggregation refers to the opposite case, where no data reduction is available and all data need to be transmitted to the sink, e.g., collecting video images from distant regions of a battlefield. Both types of data gatherings have many applications, and attract a lot of research attentions. In this paper, we mainly concern with data gathering without aggregation. Our goal is to find an energy-conserving tree at each round of data gathering to maximize the network lifetime.

For data gathering with aggregation, each node consumes energy for receiving data with constant size from all of its children and transmitting the aggregated data with the same size to its parent. Therefore, the energy consumption of each node is controlled by the number of its children (or its degree in the tree). In order to maximize the network lifetime, the degrees of all nodes in the tree should be minimized. In [4][5], a $(1+\varepsilon)$-approximation centralized algorithm is proposed to construct an *Energy-Balanced Minimum Degree Data Gathering Tree* (*EBMDDGT*), where $\varepsilon > 0$. In the tree,

[*] Corresponding author e-mail: liangjb2002@163.com

nodes with higher energy have larger degree, and vice versa. Therefore, the nodes can consume their energy uniformly and maximize the network lifetime. However, if the data cannot be aggregated, a node with smaller degree in a tree may still need to receive and transmit large amounts of data from its descendants. Therefore, EBMDDGT cannot be used in data gathering without aggregation to maximize the network lifetime.

For data gathering without aggregation, each node consumes energy for receiving data from all of its descendants and transmitting these data together with its own data to its parent. Therefore, the energy consumption of each node is controlled by the number of its descendants. In order to maximize the network lifetime, the descendant numbers of all nodes in the tree should be minimized. Since a node's descendant number is often much larger than its children's number, the minimization of all nodes' descendant numbers is harder than the minimization of all nodes' degrees.

In traditional network design fields, a shortest path tree (SPT) can be used to minimize the total energy consumption of data transmission in the network. This is because each node in the SPT can transmit its data to the sink with the minimum number of hops. However, SPT cannot achieve the maximum network lifetime because it does not consider how to conserve the energy of any individual node, e.g., a node in a shortest path has to always receive and transmit large amounts of data from other nodes even if its energy is low.

In this paper, we model the maximum network lifetime problem as one of finding a *Min-Max-Weight Spanning Tree (MMWST)*, which is shown to be NP-complete. In MMWST, the maximum weight of the nodes is smaller than other maximum weights of the nodes in other trees for the network. The weight of a node is defined as the ratio of the number of the node's descendants to the node's energy. To the best of our knowledge, this paper is the first one that researches how to maximize the network lifetime by finding MMWST.

A $\Omega(\log n/\log\log n)$-approximation centralized algorithm MITT (MaxImum lifetime Tree construction for data gaThering without aggregation) is proposed for finding MMWST. MITT starts from an initial tree, which is constructed by performing a breadth first traversal on the network. The initial tree is a SPT, which can minimize the total energy consumption of data transmission in the network but may not be able to maximize the network lifetime. Then, MITT iteratively transfers descendants of the nodes with the maximum weight (which are called bottleneck nodes) to other nodes with smaller weights to decrease the maximum weight. MITT terminates when the weights of all bottleneck nodes cannot be decreased further. Compared with existing data gathering algorithms, MITT can achieve longer network lifetime.

Moreover, MITT needs the information of nodes' energies and the network topology, which is hard to be acquired in large-scale networks. By employing the clustering technique, a solution for extending MITT to large-scale networks is proposed. The network is assumed to compose of two kinds of nodes : the regular (sensor) nodes and the cluster-heads. The network is divided into several clusters by the cluster-heads. All cluster-heads form a hierarchical structure rooted at the sink. In each cluster, MITT is performed by the cluster-head and constructs a min-max-weight spanning tree for the cluster. Each node in the cluster transmits its data to its cluster-head by the tree, and the cluster-head is responsible for transmitting these data to the sink. By this way, MITT can be implemented in a large-scale network easily. Since the cost of cluster-heads is much higher than the regular nodes, the upper bound of the number of cluster-heads that should be used to achieve the maximum network lifetime is analyzed.

The remainder of this paper is organized as follows. Section II reviews related work on data gathering. Section III describes the system model and formulates the problem. Section IV describes our algorithm. Simulation results are presented in section V, and Section VI concludes this work.

## 2 Related work

Recently, the problem of efficient data gathering in WSN has been investigated extensively. Existing protocols can be classified into three categories : cluster-based protocols, chain-based protocols and tree-based protocols. For continuous monitoring applications with a periodic traffic pattern, tree-based topology is often adopted because of its simplicity [6]. Therefore, the tree-based protocols are our main concern.

(1) In cluster-based protocols (e.g., LEACH [7], HEED [8], EECJ [9], MCMC [10], CDE [11], CDC [12], and LCTSR [13], etc.), a subset of nodes in the network are selected as cluster heads by probability, and other nodes join the closest cluster head from them to form a Voronoi-based topology. Cluster-based protocols are easy to be implemented and managed, but they have some disadvantages such as asymmetry distribution of the cluster heads, heavy load on the cluster heads, etc. In addition, the sizes of the clusters are hard to be controlled.

(2) In chain-based protocols (e.g., PEGASIS [14], DRAEM [15], CHIRON [16], EECC [17], and CREEC [18], etc.), all nodes in the network are organized as a chain. One of these nodes is selected as a head to communicate with the sink directly, and other nodes transmit their data to the head through the chain. Chain-based protocols enable each node to communicate with its closest neighbor. However, the long chain causes a large delay in data gathering and the head has the highest burden of relaying data.

(3) In tree-based protocols, all nodes in the network are organized as a tree. Each node in the tree receives data from all of its children. Then, it transmits its own data and the received data to its parent. According to the way of constructing a tree, the tree-based protocols for data

gathering without aggregation can be classified into three categories : linear programming based protocols, growth based protocols and improvement based protocols.

In linear programming based protocols, the problem of finding a maximum lifetime data gathering tree is modeled as a maximum flow problem [19] with energy constraints at the nodes. Then, the problem is solved by an integer program with linear constraints. Though finding a feasible solution to the integer program is still NP-complete, it can obtain an approximation result for the problem by relaxing some integrality conditions. MLDR [20] and RSM-MLDA [21] are typical linear programming based protocols. However, they assume that the locations of the nodes and the sink are known, which are hard to be acquired because the nodes and the sink are seldom equipped with expensive GPS devices. Moreover, they assume that each node has the ability to transmit its packet to any other node in the network or directly to the sink, which is unrealistic in a large-scale network. In [22][23][24], the maximum lifetime problem in sensor-target surveillance networks is researched. The problem is to schedule the sensors to watch the targets and forward the sensed data to the sink, such that the lifetime of the surveillance network is maximized. They firstly compute the maximum lifetime of the surveillance system and a workload matrix by using the linear programming technique. Then, they decompose the workload matrix into a sequence of schedule matrices that can achieve the maximum lifetime. Finally, they construct the sensor surveillance trees based on the above obtained schedule matrices, which specify the active sensors and the routes to pass sensed data to the sink.

For growth based protocols, their basic ideas are similar to the Prim's algorithm for *minimum spanning tree (MST)*. Each edge in the network is assigned a weight. The data gathering tree is initialized by containing only the sink node. Then, the tree is grown by adding selected edges one by one until it spans all nodes. The selected edge is from a node in the tree to another node not in the tree. The key difference in these protocols is how to define each edge's weight and how to add a proper edge to the tree at each time. In PEDAP [25], the weight of each edge is defined as the transmission cost between the edge's two end nodes. At each time, PEDAP adds an edge with the minimum weight to the tree. However, PEDAP does not consider nodes' energies and cannot achieve energy-awareness. PEDAP-PA [25] improves PEDAP by considering each node's energy during the tree construction. The weight of each edge is defined as the ratio of the transmission cost between the two end nodes to the residual energy of the sending node. PEDAP-PA achieves an approximation ratio of $\Omega(\log n)$, where $n$ is the number of nodes in the network. MLDGA [26] and MNL [27] improve PEDAP-PA by designing different edge weights. In MLDGA, the weight of each edge is defined as the ratio of the minimum lifetime of the edge's two end nodes to the transmission cost between the two nodes. At each time, an edge with the maximum weight is added to the tree. In MNL, an edge is added to the tree if it can maximize the minimum residual energy of the nodes of the new tree. MLDGA and MNL also achieve the approximation ratio of $\Omega(\log n)$.

Improvement based protocols start with an initial feasible solution, and then make improvements by adding and deleting edges as long as improvements are possible. To the best of our knowledge, LOCAL-OPT [28] is the only one improvement based protocol that constructs a tree for data gathering without aggregation. It starts from an arbitrary tree $T_0$ with lifetime $L_0$. Then, it tries to change each node's parent in the tree to one of the node's neighbors in the network to find a new tree $T_1$, which has lifetime $L_1 > L_0$. If $T_1$ is found, another new tree $T_2$ that has lifetime $L_2 > L_1$ is found by the same way. The above process of finding new trees is executed iteratively, until a tree $T_i$ with lifetime $L_i$ is found and there is no a tree $T_j$ with lifetime $L_j > L_i$ can be found. LOCAL-OPT achieves an approximation ratio of $\Omega(\log n/\log\log n)$, which is the current best result for tree construction in data gathering without aggregation. However, LOCAL-OPT does not consider the situation that $T_j$ can be found by changing parents of multiple nodes in $T_i$ at the same time. Moreover, in above optimization process of changing each node's parent to one of the node's neighbors to find a new tree, LOCAL-OPT does not use the nodes that are not the changed node's neighbors to enlarge the algorithm's optimization range.

In this paper, a new improvement based protocol MITT that can achieve better performance of finding a maximum lifetime data gathering tree than LOCAL-OPT is proposed. The system model and problem statement that MITT considers are described as follows.

## 3 System model and problem statement

### 3.1 Network model

Assume that $v_1, v_2, \ldots, v_n$ are sensor nodes (or nodes) in a WSN and $v_0$ is the sink. All nodes are randomly deployed over an $M \times M$ field. The network forms a connective undirected graph $G(V,E)$, where $V$ is the set of the nodes and the sink, and $E$ is the set of edges. $|E| = m$ is the number of edges. There is an edge $(v_i, v_j)$ in $E$ if the nodes $v_i$ and $v_j$ are within each other's communication range. Each node has different initial energy. The network has following characteristics :

(1) The network is static, i.e., all nodes and the sink are stationary after deployment.

(2) The sink has infinite power supply and powerful computation ability.

(3) All nodes have the same transmission range.

Each node mainly consumes energy in communication, and the amount of energy required to transmit, receive one bit of data is $E_{tx}$, $E_{rx}$ respectively. The energy consumed by nodes in computing and sensing is negligible [10]. At each round, each node generates $l$ bits data.

## 3.2 Definitions

Before diving into the problem of finding a maximum lifetime tree for data gathering without aggregation, some fundamental definitions and notations are given, which will be used throughout this paper .

*Definition* 1 : A round is defined as the process of gathering all the data from nodes to the sink, regardless of how much time it takes [25].

*Definition* 2 : For each node $v_i$ that has $S(T,v_i)$ descendants in a tree $T$, the size of data that it receives at each round is $S(T,v_i)l$ and the size of data that it needs to transmit is $(S(T,v_i)+1)l$.

*Definition* 3 : At each round, the amount of energy $C(T,v_i)$ that each node $v_i$ consumes in a tree $T$ is :

$$\begin{aligned} C(T,v_i) &= S(T,v_i)lE_{rx} + (S(T,v_i)+1)lE_{tx} \\ &= S(T,v_i)l(E_{rx}+E_{tx}) + lE_{tx} \end{aligned} \tag{1}$$

*Definition* 4 : The lifetime of a node $v_i$ in a tree $T$ is defined as the number of rounds that the node can receive data from all of its descendants and transmit these data together with its own data to its parent :

$$L_{node}(T,v_i) = \lfloor \frac{E(v_i)}{S(T,v_i)l(E_{rx}+E_{tx})+lE_{tx}} \rfloor \tag{2}$$

*Definition* 5 : The lifetime of a tree $T$ is defined as the number of rounds that the tree can sustain until the first node in the tree depletes its energy :

$$L_{tree}(T) = \min_{i=1,\dots,n} \{L_{node}(T,v_i)\} \tag{3}$$

*Definition* 6 : The network lifetime is defined as the number of rounds that the network can perform data gathering until the first node in the network depletes its energy.

In fact, an alternative definition of the network lifetime that can be used is the number of rounds that the network can perform data gathering until a certain percentage of nodes in the network deplete their energies. That is because the nodes are often deployed in a network densely, the quality of the network is not affected until a significant amount of nodes die. In [30], it is shown that this definition is actually quite similar in nature to Definition 6. Therefore, Definition 6 is used in this paper for simplicity.

*Definition* 7 : Let $T_S(G) = \{T_1,\dots,T_z\}$ be the set of spanning trees in the network and $w(T_i)$ be the weight of a tree $T_i$ in $T_S(G)$, where $w(T_i)$ equals the maximum weight of the nodes in the tree $T_i$, i.e., $w(T_i)= \max_{\substack{i=1,\dots,z\\ j=1,\dots,n}} w(T_i,v_j)$. A spanning tree $T_i$ is called a *min-max-weight spanning tree* if its weight is the minimum among all trees in $T_S(G)$, i.e., $w(T_i)= \min_{T\in T_S(G)} w(T)$.

## 3.3 Problem statement

### *(a) The problem of finding a min-max-weight spanning tree (PMMWST)*

In tree-based protocols, a tree can be used for a period of time or be reconstructed at each round, depending on specific applications. If a tree is used until the first node depletes its energy, the network lifetime is equal to the lifetime of the tree. If a tree is reconstructed at each round or after it is used for several rounds, the network lifetime is decided by the trees that are constructed.

No matter whether the tree is reconstructed or not, it is expected that the tree can effectively conserve nodes' energies at each round so as to maximize the network lifetime. A tree with maximum lifetime can achieve above expectation, so it is our main concern.

For a network $G$, there exists multiple possible trees. Our goal is to find a tree with the maximum lifetime :

$$\max_{T\in T_S(G)} L_{tree}(T) \tag{4}$$

According to Definitions 4 and 5, Formula (4) can be defined as :

$$\max_{T\in T_S(G)} \min_{i=1,\dots,n} \lfloor \frac{E(v_i)}{S(T,v_i)l(E_{rx}+E_{tx})+lE_{tx}} \rfloor \tag{5}$$

In Formula (5), since $E_{rx}$ and $E_{tx}$ are constant, $S(T,v_i)$ is the main optimization objective. Being extracted the constant $E_{rx}$ and $E_{tx}$ from the denominator, the problem for solving Formula (5) is transformed to a new problem :

$$\max_{T\in T_S(G)} \min_{i=1,\dots,n} \frac{E(v_i)}{S(T,v_i)+c} \tag{6}$$

where $c = E_{tx}/(E_{rx}+E_{tx})$. In Formula (6), the goal is to maximize the minimum of $E(v_i)/(S(T,v_i)+c)$, where $i = 1,\dots,n$. In order to achieve this goal, a node with larger energy (larger $E(v_i)$) should take more responsibilities by serving more descendants (larger $S(T,v_i)$) in a tree, and vice versa.

In Formula (6), the variable $S(T,v_i)$ is in the denominator, which is not convenient to be analyzed. Hence, the problem for solving Formula (6) is transformed to another problem by changing $E(v_i)/(S(T,v_i)+c)$ to $(S(T,v_i)+c)/E(v_i)$ as follows :

$$\min_{T\in T_S(G)} \max_{i=1,\dots,n} \frac{S(T,v_i)+c}{E(v_i)} \tag{7}$$

Each node $v_i$ in $T$ is given a weight $w(T,v_i)$=$(S(T,v_i)+c)/E(v_i)$, then Formula (7) is transformed into :

$$\min_{T\in T_S(G)} \max_{i=1,\dots,n} w(T,v_i) \tag{8}$$

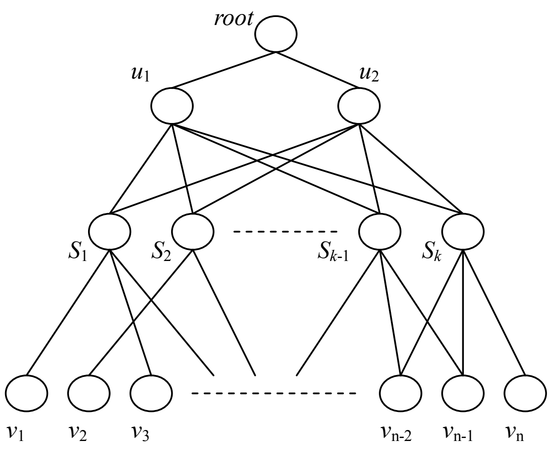


**FIGURE 1** Reduction from SET-COVER to our problem

Note that Formula (8) has the same goal with the initial problem of solving Formula (4). From Formula (8), the initial problem of finding a maximum lifetime tree is transformed to the problem of finding a min-max-weight spanning tree, i.e., PMMWST. Next, we will establish the hardness of PMMWST.

*(b) Hardness of PMMWST*

The decision version of PMMWST is : Given a network $G(V,E)$ and node energies, does there exist a spanning tree in which the maximum weight of the nodes is $w$?

**Theorem 1 :** The decision version of PMMWST is NP-complete.

*Proof* : The proof employs the minimum set cover problem [29] (SET-COVER for short), which is known to be NP-complete. PMMWST's decision version is shown to be NP-complete as follows.

PMMWST's decision version is clearly in NP. Given a tree and node energies, it is easy to verify whether the maximum weight in the tree reaches $w$. In order to prove that PMMWST's decision version is NP-Complete, a reduction from SET-COVER to it is shown.

Given an element set $S = \{v_1, v_2, \ldots, v_n\}$ and a collection of subsets $C = \{S_1, S_2, \ldots, S_k\}$ of $S$, the decision version of SET-COVER is to determine whether there is a selection of $K$ subsets from $C$ such that the union of $K$ subsets cover all the $n$ elements, where $1 \leq K \leq k$. Given an instance of SET-COVER, an instance of the network can be constructed in polynomial time as follows.

The nodes in the network are arranged in four rows, as shown in Figure 1. The first row just has a root node (the sink), which has unlimited energy supply. The second row consists of two nodes : $u_1$ with energy $(n+K)+c$ and $u_2$ with energy $(k-K)+c$. The two nodes are used to decide which subsets will be in the set cover and which will not. Both of the two nodes are connected to the root node. There are $k$ nodes corresponding to $S_1, S_2, \ldots, S_k$ in the third row, and each node corresponding to $S_j$ has $2(|S_j|+c)$ units of energy, where $1 \leq j \leq k$ and $|S_j|$ is the size of $S_j$. The $k$ nodes are connected to both $u_1$ and $u_2$. In the fourth row, $n$ nodes with energy $2c$ are corresponding to the $n$ elements of $S$. For each node corresponding to the element $v_i$, there is an edge from it to the node corresponding to $S_j$ if $v_i$ is in $S_j$, where $1 \leq i \leq n$.

In Figure 1, each node corresponding to $v_i$ or each node corresponding to $S_j$ has a weight at most 1/2. The nodes that can achieve larger weights in the instance are nodes $u_1$ and $u_2$. Now, it can be claimed that there exists a spanning tree in which the maximum weight of the nodes is 1 if and only if a set cover of size $K$ exists.

Suppose that there exists a set cover of size $K$. Then the tree can be constructed as follows. The data from the $K$ third row nodes and the $n$ fourth row nodes can be routed through $u_1$ to the root. The data from rest of the $k-K$ third row nodes can be routed through $u_2$. Only by this way, both $u_1$'s and $u_2$'s weights achieve 1, i.e., the maximum weight of the nodes in the tree is 1.

Conversely, if there is a spanning tree in which the maximum weight of the nodes is 1, then it is easy to construct a set cover for the SET-COVER instance. That is the union of the subsets corresponding to the $K$ third row nodes, which contains all the elements in $S$. □

Since PMMWST is hard, an approximate solution needs to be found. In this paper, an approximation algorithm MITT is proposed to solve PMMWST. Details of the algorithm are shown as follows.

## 4 MITT algorithm

### 4.1 Algorithm description

Firstly, an initial tree $T$ is constructed in the network $G$ as follows :

(1) Initially, there is only one node $v_0$ in the tree $T$, and it is in level 1 of $T$.

(2) All neighbors of $v_0$ are added to $T$ as its children, and these nodes are in level 2 of $T$.

(3) The nodes in level $h$ of $T$ are sorted in descending order according to their energies, where $h \geq 2$. According to the order, each node in level $h$ picks all of its neighbors that are not in the tree as its children in turn. When all neighbors of the nodes in level $h$ are added into the tree $T$, the level $h+1$ of the tree is formed.

Step (3) is executed iteratively and terminates when all nodes in the network are added to the tree. When step (3) terminates, $T$ is got. Based on the initial tree, a *min-max-weight spanning tree* is constructed as follows.

Given an arbitrary variable $\varphi$ >0, all nodes in $T$ are classified into three disjoint subsets $V_1, V_2$ and $V_3$ :

(1) $V_1 = \{v_i | w(T) - \varphi < w(T, v_i) \leq w(T), v_i \in V\}$. $V_1$ contains the nodes whose weights are equal to or close to

```
Function Capacity($T,x$)
1. { if ($x$ is the sink) $x.capacity$=$n$ ;
        // $n$ is the number of nodes in sensor network
2.   else{ case $x \in V_1$ $x.capacity$=-1 ;
3.         case $x \in V_2$ $x.capacity = \min\{0, x.parent.capacity\}$ ;
4.         case $x \in V_3$ $x.capacity = \min\{\lfloor (w(T) - \varphi - w(T,x))\ E(x) - c \rfloor, x.parent.capacity\}$ ;
5.       }
6.   For each child $y$ of $x$ in tree $T$
7.       Capacity($T$, $y$) ;
8. }
```

**FIGURE 2** Function Capacity($T$, $x$)

the weight $w(T)$ of the tree $T$ (or the maximum weight of the nodes in the tree). These nodes are called "*bottleneck nodes*".

(2) $V_2 = \{v_i | w(T) - \varphi - 1/E(v_i) < w(T,v_i) \leq w(T) - \varphi, v_i \in V\}$. Each node in $V_2$ will become a *bottleneck node* if the number of its descendants is increased by one. The nodes in $V_2$ are called "*sub-bottleneck nodes*".

(3) $V_3 = V - V_1 - V_2$. $V_3$ contains all remaining nodes, and these nodes are called "*rich nodes*". Each node in $V_3$ will not become a *bottleneck node* even if the number of its descendants is increased by one.

However, a node in $V_3$ does not mean that it can be added more descendants. That is because some of the node's ancestors may be in $V_1$ or $V_2$, and they would become new bottleneck nodes if the number of the node's descendants is increased. To represent this property, an attribute *capacity* is used to denote a node's ability for serving more descendants according to its weight and its ancestors' weights.

For a *bottleneck node*, it has larger weight than other nodes. Therefore, it and its descendants should not be added more descendants. As a result, if a node is a *bottleneck node* or a bottleneck node's descendant, its *capacity* is set to -1.

For a *sub-bottleneck node*, its descendant number should not be increased too. Therefore, if a node is a sub-bottleneck node, its *capacity* is set to the minimum between 0 and its ancestors' *capacities*.

If a node $v_i$ is a *rich node*, its *capacity* is equal to the minimum between $\lfloor (w(T) - \varphi - w(T,v_i))E(v_i) - c \rfloor$ and its ancestors' *capacities*. Note that $\lfloor (w(T) - \varphi - w(T,v_i))E(v_i) - c \rfloor$ is the number of descendants that $v_i$ can further afford.

MITT can compute all nodes' *capacities* by traversing the tree in depth first order once. A Function Capacity($T,x$) is defined to compute the *capacities* of nodes in a tree $T$ rooted at a node $x$, which is shown in Figure 2.

After all nodes' *capacities* are got, a transferring operation is performed on each *bottleneck node* $v_i$ to try to transfer some of $v_i$'s descendants (the number of these descendants is $s$) to another node $v_j$ that is not $v_i$'s descendant to decrease $v_i$'s weight. In the transferring operation, there are 3 cases that may happen as follows :

(1)if $v_j$'s *capacity* is larger than $s$, $v_i$ can transfer its $s$ descendants to $v_j$ directly, i.e., $v_j$ becomes the $s$ descendants' new ancestor.

(2)If $v_j$'s *capacity* equals 0, it means that $v_j$ is in a sub-tree rooted at a *sub-bottleneck node* $v_k$, where $v_k$ and $v_i$ may be the same node and $v_k$ has the characteristics that $v_k.capacity$==$v_i.capacity$ and $v_k.parent.capacity > v_i.capacity$. Since $v_k$'s *capacity* decides $v_j$'s *capacity*, $v_k$'s *capacity* is increased firstly. By performing the transferring operation on $v_k$ recursively, $v_k$'s *capacity* can be increased. Therefore, if $v_j$'s *capacity* becomes larger than $s$ after $v_k$'s *capacity* is increased, $v_i$ can transfer its $s$ descendants to $v_j$.

(3)If $v_j$'s *capacity* is not larger than $s$, $v_i$ does not transfer its $s$ descendants to $v_j$.

A Function Transfer($T,x,k_1$) is defined to implement the transferring operation on any node $x$ in the tree $T$, as shown in Figure 3, where the variable $k_1 \geq 0$ is an integer used to limit the depth of the recursion that the transferring operation can be performed. If $k_1$ is set to a large value, e.g., $k_1$=$\infty$, the transferring operation can be performed on any *sub-bottleneck node* that needs to increase its *capacitie* and the probability that the transferring operation can transfer some of a *bottleneck node*'s descendants to a *sub-bottleneck node* is high. However, the transferring operation may need more time to terminate if $k_1$ is a large value. If $k_1$ is set to a small value, e.g., $k_1$=1 or $k_1$=0, the transferring operation just can be performed on a few or no *sub-bottleneck nodes* and the probability that a *bottleneck node* can transfer some of its descendants to a *sub-bottleneck node* becomes small. However, the transferring operation can terminate in a short time if $k_1$ is a small value.

In Figure 3, Function Transfer($T,x,k_1$) is performed by traversing the tree rooted at $x$ in breadth first order. If one or more descendants of $x$ are transferred to another node, the function returns *TRUE*. Otherwise, the function returns *FALSE*.

First, each child $y$ of $x$ in the tree $T$ is enqueued in a queue $Q_{children}$. While the queue $Q_c$ is not empty, a node is dequeued from $Q_c$ into $v$. Then, each child $y'$ of $v$ in the enqueued in the queue $Q_c$ to ensure the breadth first search on the tree rooted at $x$. If the node $v$ has the characteristics that $v.capacity$==$x.capacity$ and $v.visited$== "*FALSE*", the transferring operation tries to transfer $v$ to a node that is not $x$'s descendant, where the attribute *visited* is used to show whether the transferring operation has been performed on a node. Limiting that $v.capacity$==$x.capacity$ is because $v.capacity$ may be smaller than $x.capacity$ in the tree and $v$ may being performed the transferring operation before the transferring operation is performed on $x$ recursively.

In order to increase the probability that $v$ can be transferred to a node that has a *capacity* larger than the number of nodes including $v$ and $v$'s descendants, all of $v$'s neighbors in the network are sorted in descending

```
Function Transfer($T, x, k_1$)
1.  for(each child $y$ of node $x$ in the tree $T$)
2.     enqueue($Q_c$, $y$) ; //enqueue the node $y$ in a queue $Q_c$
3.  while ($Length(Q_c) > 0$)
4.  {  dequeue($Q_c$, $v$) ;  //dequeue a node from $Q_c$ into $v$
5.     for(each child $y'$ of node $v$ in the tree $T$)
6.        enqueue($Q_c$, $y'$) ;
7.     if ($v.capacity$==$x.capacity$ and $v.visited$==“FALSE”)
8.     {  all neighbors of $v$ in the network are first sorted in
          descending order according to their capacities, and then
          they are enqueued in a queue $Q_n$ ;
9.        while ($Length(Q_n) > 0$)
10.       {  dequeue($Q_n$, $z$) ;
11.          if(node $z$ is not a descendant of node $x$ in $T$)
12.          { case $z.capacity > S(T, v)$
13.                ChangeParent($v$, $z$) ;
14.                return TRUE ;
15.            case ($z.capacity$==0)&& ($k_1$ >0)
16.                find the ancestor $v_k$ of $z$ that has
                   the characteristics $v_k.capacity$==$z.capacity$ and
                   $v_k.parent.capacity > z.capacity$ ;
17.                if($v_k.visited$==“FALSE”)
18.                { $v_k.visited$=“TRUE” ;
19.                  if ( Transfer($T, v_k, k_1$-1) && $z.capacity >$
                     $S(T, v)$)
20.                  { ChangeParent($v, z$) ;
21.                    return TRUE ;
22.                  }
23.                }
24.            case ($z.capacity \le S(T, v)$) break ;
25.          } //endif
26.       } //endwhile
27.    } //endif
28. } //endwhile
29. return FALSE ;

Function ChangeParent($v, z$)
1.  $w = v.parent$ ;
2.  $v.parent = z$ ;
3.  update $S(T, v_i)$ for each node $v_i$ that is $v$’s old ancestor
    or $v$’s new ancestor, and let $v_i.visited$=“FALSE” ;
4.  Capacity ($T, v_0$) ;
```

**FIGURE 3** Function Transfer($T, x, k_1$)

order according to their *capacities* and then they are enqueued in a queue $Q_n$. While the queue $Q_n$ is not empty, a node is dequeued from $Q_n$ into $z$. If $z$ is not a descendant of node $x$ in the tree $T$, there are three cases that may happen as follows :

(1) If $z.capacity > S(T, v)$, $v$ can be transferred to $z$ directly, i.e., $v$ takes $z$ as its new parent. Since the tree is changed, each node that is $v$’s old ancestor or $v$’s new ancestor sets its *visited* attribute to “*FALSE*”. Then, the *capacities* of all nodes in the tree are re-computed. Finally, the function returns *TRUE*.

(2) If $z.capacity$==0, $v$ cannot be transferred to $z$ directly. Therefore, the root node $v_k$ of the sub-tree containing $z$ should be found, which has the characteristics that $v_k.capacity$==$z.capacity$ and $v_k.parent.capacity > z.capacity$. If $v_k.visited$==“*FALSE*”, the transferring operation is performed on $v_k$ recursively, i.e., Function Transfer($T, v_k, k_1$) is executed. If some of $v_k$’s descendants are transferred and $z$’s *capacity* is larger than $S(T, v)$, $v$ can be transferred to $z$. Otherwise, $v$ does not transfer to $z$. If the transferring operation succeeds and the tree is changed, each node that is $v$’s old ancestor or $v$’s new ancestor sets its *visited* attribute to “*FALSE*”. Then, the *capacities* of all nodes in the tree are re-computed. Finally, the function returns *TRUE*.

(3) If $z.capacity \le S(T, v)$, $v$ cannot be transferred to any of its neighbors. Therefore, the function stops the current transferring operation on $v$ and continues to dequeue another node from $Q_c$ into $v$ to re-perform above operations.

If $z$ is a descendant of node $x$ in the tree $T$, $v$ does not transferred to $z$ and another node is dequeued from $Q_n$ into $z$. Then, $v$ tries to be transferred to the new $z$. If the queue $Q_n$ is empty and $v$ cannot be transferred to any one of its neighbors, another node is dequeued from $Q_c$ into $v$ to re-perform above above operations. If the queue $Q_c$ is empty and there is no a node in the tree rooted at $x$ can be transferred, the function return *FALSE*.

An algorithm MITT, which uses Function Capacity($T, x$) and Transfer($T, x, k_1$) to construct a *min-max-weight spanning tree*, is shown in Figure 4. The inputs of MITT are $G$, $k_1$, and $k_2$, where $k_2 > 0$ is an integer controlling the number of iterations that the algorithm can be performed. If $k_2$ is large value, e.g., $k_2$=∞, MITT can be performed until a tree $T_i$ is constructed and there is no a tree that has larger lifetime than $T_i$’s lifetime can be found. However, MITT may need more time to terminate if $k_2$ is a large value. If $k_2$ is a small value, e.g., $k_2$=0, MITT will terminate in a short time. However, MITT just can find a tree with small lifetime if $k_2$ is a small value.

In Figure 4, MITT firstly constructs an initial tree $T$ in step 1. Then, MITT enters the iteration process of transferring descendants of bottleneck nodes in step 3. In each iteration, it first traverses the tree $T$ to compute $w(T)$ and sets each node’s *visited* attribute to “*FALSE*”. Then, it computes the *capacities* of all nodes in $T$. Next, it sorts the nodes in $V_1$ in decreasing order according to their weights and then enqueues these nodes in a queue $Q$. Finally, it performs the transferring operation on each node $x$ in $V_1$ according to the order, i.e., while the queue $Q$ is not empty, a node is dequeued from $Q$ into $x$ and the transferring operation is performed on $x$ at each time.

If the transferring operation succeeds, i.e., Transfer($T$, $x$, $k_1$)==*TRUE*, MITT breaks out of current iteration and enters the next iteration. If the transferring operation fails, i.e., Transfer($T, x, k_1$)==*FALSE*, MITT performs the transferring operation on the next node that is dequeued

```
Algorithm MITT
    $Input$ : Network $G(V,E)$, the parameters $k_1$ and $k_2$
    $Output$ : a min-max-weigh spanning tree for data
    gathering without aggregation
 1. construct an initial tree $T$ ;
 2. Ischanged="TRUE" ;
 3. while((Ischanged=="TRUE" ) && ($k_2 >= 1$) )
 4. { Ischanged="FALSE" ; $k_2 = k_2 - 1$ ;
 5.     traverse the tree $T$ to compute $w(T)$, and let each
    node v.visited="FALSE" ;
 6.     Capacity $(T, v_0)$ ;
 7.     all the nodes in $V_1$ are first sorted in descending order
    according to their weights, and then they are enqueued
    in a queue $Q$ ;
 8.     while ($Length(Q) > 0$)
 9.     { dequeue($Q$, $x$) ;
10.         if(Transfer($T, x, k_1$))
11.         { Ischanged="TRUE" ;
12.             break ;
13.         } //endif
14.     } //endwhile
15. } //endwhile
```

**FIGURE 4** MITT Algorithm

from $Q$, and so on. If the queue $Q$ is empty, it means that the weights of all bottleneck nodes in $V_1$ cannot be decreased and the algorithm MITT terminates.

In MITT, each node's *visited* attribute is firstly set to "*FALSE*" when the algorithm is executed. If a node $v_i$ is performed the transferring operation, i.e., Function Transfer$(T, v_i, k_1)$ is executed, its *visited* attribute is set to "*TRUE*". When the number of the node's descendants is changed, i.e., the transferring operation succeeds, the node's *visited* attribute becomes "*FALSE*" again. For each node, it would not be performed the transferring operation again if its *visited* attribute equals "*TRUE*". Therefore, the *visited* attribute prevents each node from being performed the transferring operation repeatedly, which can eliminate the appearance of dead lock.

**Proposition 1 :** For MITT, each iteration will be finished in polynomial time. When MITT terminates, it is performed at most $O(n^2(1+1/(\varphi E_{min})))$ iterations.

*Proof* : As shown in Figure 4, step 1 is to construct an initial tree $T$. It starts with a tree that contains only the sink node, and then selects a node that is not in the tree to join the tree at each time iteratively. Hence, after $n$ times of selecting, all nodes are added into the tree.

Steps 3-15 describe the process of transferring descendants of bottleneck nodes in the tree $T$ by iterations. In each iteration, MITT needs to traverse the sub-trees rooted at the *bottleneck nodes* (may be include other sub-trees rooted at *sub-bottleneck nodes*) and check neighbors of the nodes in these sub-trees, which costs $O(nm)$ time. Computing the *capacities* of all nodes in the tree $T$ cost $O(n) < O(nm)$ time. Therefore, each iteration costs $O(nm)$ time.

On the other hand, the attribute *capacity* ensures that the transferring operation just transfers descendants of the nodes with larger weights to the nodes with smaller weights and there is no new bottleneck nodes will be generated after the transferring. Therefore, the maximum weight of the nodes in the network can only be decreased. When the weights of all bottleneck nodes cannot be decreased, MITT terminates. The largest number of iterations that MITT is performed when it terminates is analyzed as follows.

Let $w^*$ denote the weight of the optimal tree, which equals the maximum weight of the nodes in the optimal tree. In MITT, a bottleneck node can be changed to a sub-bottleneck node or a rich node, if its weight is decreased to a value that is smaller than $w(T)-\varphi$. When the weights of all bottleneck nodes are smaller than $w(T)-\varphi$, $w(T)$ is decreased by at least $\varphi$. When MITT terminates, it is performed at most $\lceil (w(T)-w^*)/\varphi \rceil Z < \lceil ((n+c)/E_{min} - 0)/\varphi \rceil Z = \lceil (n+c)/(\varphi E_{min}) \rceil Z$ iterations, where $Z$ is the number of iterations that MITT needs to decrease the weights of all bottleneck nodes to values that are smaller than $w(T)-\varphi$ and $E_{min} = \min_{i=0,\dots,n} E(v_i)$.

For a bottleneck node $v_i$, its weight will be decreased by $1/E(v_i)$ if one of its descendants is transferred. Therefore, after at most $\lceil \varphi/(1/E(v_i)) \rceil = \lceil \varphi E(v_i) \rceil$ iterations, its weight will be smaller than $w(T)-\varphi$. In order to decrease the weights of all bottleneck nodes to values that are smaller than $w(T)-\varphi$, the largest number of iterations $Z'$ that MITT needs to be performed is :

$$\begin{aligned} Z' = \sum_{v_i \in V_1} \lceil \varphi E(v_i) \rceil &\le \sum_{v_i \in V_1} (\varphi E(v_i) + 1) \\ &\le \varphi \sum_{v_i \in V_1} E(v_i) + |V_1| \end{aligned} \tag{9}$$

For any bottleneck node $v_i$, there is $w(T)-\varphi < w(T, v_i) \le w(T)$. Therefore, we have $w(T)-\varphi < (S(T,v_i)+c)/E(v_i) \Rightarrow (w(T)-\varphi)E(v_i) < S(T,v_i)+c$. Since $S(T,v_i) < n$, for all bottleneck nodes we have :

$$\begin{aligned} (w(T)-\varphi) \sum_{v_i \in V_1} E(v_i) &< \sum_{v_i \in V_1} (S(T,v_i)+c) \\ &< n^2 + cn \\ \Rightarrow \sum_{v_i \in V_1} E(v_i) &< (n^2+cn)/(w(T)-\varphi) \end{aligned} \tag{10}$$

Since $|V_1| < n$ and $c = E_{tx}/(E_{rx}+E_{tx}) < 1$, Formula (9) is transformed into a new form as follows by combining with Formula (10) :

$$\begin{aligned} Z' &\le \varphi \sum_{v_i \in V_1} E(v_i) + |V_1| \\ &< \varphi(n^2+cn)/(w(T)-\varphi)+n \\ &< (\varphi n^2 + nw(T))/(w(T)-\varphi) \end{aligned} \tag{11}$$

Notice that $w(T) \le \lceil (n+c)/E_{min} \rceil$ and $Z \le Z'$, there is $Z = O(n(\varphi E_{min}+1))$. Therefore, when MITT terminates, it is performed at most $\lceil (n+c)/(\varphi E_{min}) \rceil Z = O(n^2(1+1/(\varphi E_{min})))$ iterations. $\square$

## 4.2 Time complexity analysis

**Theorem 2 :** The time complexity of MITT is $O(n^3 m(1+1/(\varphi E_{min})))$.

*Proof* : In Figure 4, step 1 costs $O(n)$ time to construct the initial tree by traversing the network in breadth first order. In step 3-15, the algorithm is performed at most $O(n^2(1+1/(\varphi E_{min})))$ iterations, and each iteration costs $O(nm)$ time. Therefore, the total time spent in step 3-15 is $O(n^2(1+1/(\varphi E_{min})))O(nm)=O(n^3 m(1+1/(\varphi E_{min})))$. Based upon the above analysis, the time complexity of the whole algorithm is $O(n^3 m(1+1/(\varphi E_{min})))$. $\square$

## 4.3 Approximation ratio

In this subsection, the approximation ratio of MITT is analyzed by a network instance as follows.

**Theorem 3 :** The MITT algorithm can achieve an approximation ratio of at least $\Omega(\log n/\log\log n)$.

*Proof* : Consider a network $G$ that is shown in Figure 5(a). Assume that each node has $e$ units of energy and generates one bit of data at each round. Moreover, each node will consume $a$, $b$ units of energy in receiving, sending one bit of data. An optimal tree is shown in Figure 5(b), in which the bottleneck nodes have just one child. The lifetime of the optimal tree is $L_{opt} = e/(a+2b)$. Figure 5(c) shows the initial tree constructed by MITT, in which each bottleneck node has 4 children.

For MITT, when $k_1$=0, the function Transfer$(T,x,k_1)$ cannot be performed on any *sub-bottleneck node*, i.e., MITT cannot transfer a *bottleneck node*'s descendants to a *sub-bottleneck node* by firstly transferring some descendants of the *sub-bottleneck node* to a third node to increase the *sub-bottleneck node*'s *capcity*. Therefore, in one of the worst cases, the initial tree is changed to a locally optimal tree, as shown in Figure 5(d), where the locally optimal tree is a tree that MITT constructs and MITT cannot find another tree that has larger lifetime than the tree.

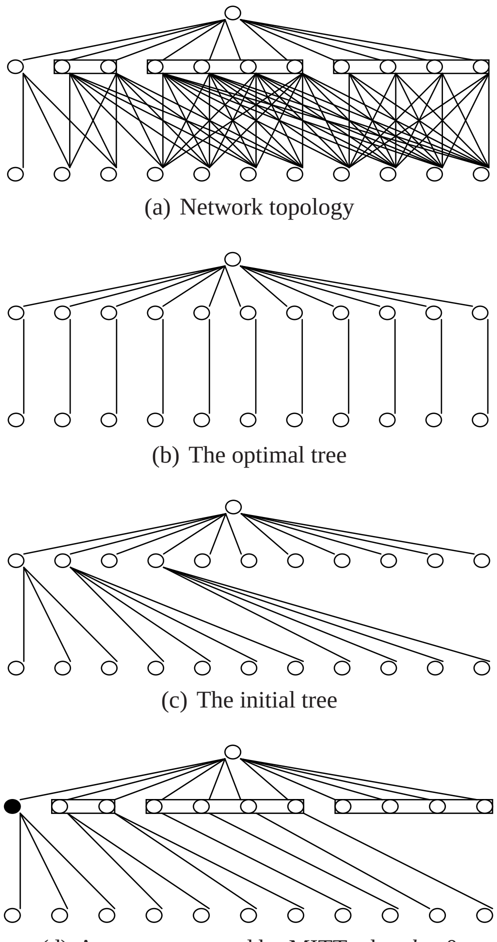


**FIGURE 5** A network topology and some of its trees

In Figure 5(d), the bottleneck node on the leftmost of the middle row has $q$=3 children. Therefore, the lifetime of the locally optimal tree is $L_{MITT}$=$e/(q(a+b)+b)$. In this situation, MITT achieves an approximation ratio $A$=$L_{opt}/L_{MITT}$=$(q(a+b)+b)/(a+2b)$.

Clearly, the construction of the locally optimal tree can be extended to arbitrary $q$. Consider the following recurrence. Let $N(f)$ be the number of nodes that have $0 \le f \le q$ children. From Figure 5(d), there are $N(3)$=1, $N(2)$=2, $N(1)$=4 and $N(0)$=4. Therefore, a recurrent inference is formed as follows : $N(q)$=1, $N(q-1)$=$q-1$, $N(q-2)$=$(q-1)N(q-1)$ and $N(q-3)$=$(q-2)N(q-2)$. From the above inference, the functional form of $N(f)$ is got :

$$N(f) = \begin{cases} (f+1)N(f+1), & \text{if } 0 \le f \le q-2, \\ q-1, & \text{if } f = q-1, \\ 1, & \text{if } f = q. \end{cases} \tag{12}$$

```
For the sink $v_0$ :
1.  $v_0$ is assigned a level $Level_0$=0 ;
2.  $v_0$ broadcasts a packet $packet(v_0)$=$(v_0, Level_0)$ that
    contains its ID and its level to its neighbors ;

For each node $v_i$ :
1.  if ($v_i$ receives a packet $packet(v_j)$=$(v_j, Level_j)$ and $v_i$
    does not set a node as its parent)
2.  { $v_i$ sets $v_j$ as its parent ;
3.     $Level_i$=$Level_j$+1 ;
4.     $v_i$ broadcasts a packet $packet(v_i)$=$(v_i, Level_i)$ to its
    neighbors ;
5.  }
```

**FIGURE 6** The scheme of constructing a hierarchical structure with the sink being the root for all nodes in the network

Since $n > \sum_{f=1}^{q} fN(f) = O(q^q)$, $q=\Omega(\log n/\log\log n)$. Therefore, the approximation ratio of MITT is $A = \Omega(\log n/\log\log n)$.

When $k_1 > 0$, MITT can achieve higher performance than MITT with $k_1$=0. For example, in Figure 5, the locally optimal tree can continue to be optimized to a tree with higher lifetime or the optimal tree by MITT if $k_1 > 1$. Therefore, MITT can achieve an approximation ratio of at least $\Omega(\log n/\log\log n)$. □

## 4.4 Implementation of MITT

Since some nodes may die for depleting their energy or physical damage, and some other nodes may be supplemented to the network at some time, the set of the nodes may be changed at each round. In order to adapt to the dynamic characteristic of the network, the tree is reconstructed at each round. The implementation of MITT is similar to LEACH and IAA, where each round begins with a set-up phase, and then is followed by a steady-state phase. MITT is performed to compute a *min-max-weight spanning tree* in the set-up phase, and data gathering based on the tree is carried through in the steady-state phase.

In the set-up phase, the sink firstly collects the information of all nodes' energies and neighbors. Since each node does not know the path from it to the sink, it cannot transmit its information to the sink. In order to solve this problem, a scheme is proposed to construct a hierarchical structure with the sink being the root for all nodes in the network, as shown in Figure 6.

In Figure 6, the sink is firstly assigned a level $Level_0$=0. Then, the sink broadcasts a packet $packet(v_0)$=$(v_0, Level_0)$ that contains its ID and its level to its neighbors. For each node $v_i$, if it receives a packet $packet(v_j)$=$(v_j, Level_j)$ and it does not set a node as its parent, it sets $v_j$ as its parent. Then, $v_i$ is assigned a level $Level_i$=$Level_j$+1. Finally, $v_i$ broadcasts a packet $packet(v_i)$=$(v_i, Level_i)$ to its neighbors. When the above process terminates, a hierarchical structure with the sink being the root is constructed.

By the hierarchical structure, each node can transmit the information of its energy and its neighbors to the sink. To guarantee that all the information can be received by the sink, reliable data delivery mechanisms like hop-by-hop acknowledgments are used [4]. After the sink receives the information of all nodes, it computes a *min-max weight spanning tree* by using MITT for the network. Finally, it informs each node the information of the tree, including each node's parent and children. At this time, the set-up phase terminates and the steady-state phase begins.

In the steady-state phase, each node firstly receives the data from its children. Then, it transmits its data and all of its children's data to its parent.

## 4.5 Extension to large-scale networks

In a large-scale network, it is not easy for the sink to collect the information of all nodes. Therefore, the implementation of MITT is hard to achieve. In order to enable MITT to be executed easily in a large-scale network, a clustering-based solution is proposed as follows.

Clustering is a promising technique for algorithm's implementation in large-scale sensor networks because of its high scalability and efficiency [31]. By dividing the whole network into small clusters, each cluster-head can collect the information of nodes in its cluster easily. Then, the cluster-head can execute MITT to construct a *min-max weight spanning tree* for the cluster. In the process of data gathering, each node only needs to transmit its data to its cluster-head in short distance. Moreover, each node just needs to relay a few or no data from other nodes. Therefore, each node can conserve its energy effectively.

Similar to [32], the network is assumed to compose of two kinds of nodes that are deployed in the field randomly : regular (sensor) nodes and cluster-heads :

(1)The regular nodes have limited energy and limited transmission range, and they perform operations such as sensing as well as data relaying. In the network, each regular node joins the closest cluster-head to form a Voronoi cell. Since the regular node's energy is limited, how to conserve each regular node's energy to maximize the network lifetime is our main concern.

(2)The cluster-heads are equipped with enough energy, so they can work for a long time. The transmission range of the cluster-heads is much longer than that of the regular nodes. Each cluster-head is responsible for collecting data from nodes within its cluster and transmitting these data to the sink. Note that the sink can act as a cluster-head too. In order to implement the data transmission from each cluster-head to the sink, all cluster-heads forms a hierarchical structure with the sink being the root.

Through the clusters, data gathering is performed as follows :

(1) Intra-Cluster. In each cluster, the cluster-head collects the information of regular nodes' energies and their neighbors. Then, MITT is performed at the cluster-head to compute a *min-max-weight spanning tree* for the cluster. At each round, the regular nodes will transmit their data to the cluster-head by the tree.

(2) Inter-Cluster. When each cluster-head gathers the data from all the regular nodes in its cluster, it transmits these data to its parent (which may be another cluster-head or the sink).

However, since the cluster-heads have higher energy and longer communication range than the regular nodes, they have more complex hardware than the regular nodes. As a result, the cost of a cluster-head is much higher than that of a regular node, where the cost of a cluster-head (or regular node) is defined as the manufacturing cost of the hardware as well as the battery of the cluster-head (or the node). Therefore, it is unrealistic to deploy large number of cluster-heads in the network.

**Theorem 4 :** In order to maximize the lifetime of a network that is composed of $n$ regular nodes and $N_c$ cluster-heads with the lowest cost, the upper bound of $N_c$ is $log(1-2^{logP/n})/log(1-\pi r^2/M^2)$, where $P$ is the probability that each regular node can communicate with at least one cluster-head directly.

*Proof* : From Formula (6), the network lifetime is maximized if the descendant number $S(T,v_i)$ of each regular node $v_i$ in a tree $T$ is minimized. Therefore, if each regular node's descendant number equals 0, the network lifetime achieves its maximum.

On the other hand, if the number $N_c$ of the cluster-heads is large enough, each regular node can transmit its data to at least one of the cluster-heads directly and has no descendants in the tree constructed in its cluster. Therefore, there is a connection between the value of $N_c$ and the value of $P$. Next, the relationship between $N_c$ and $P$ is analyzed.

Since each regular node's communication range is $r$, the size of its coverage area is $\pi r^2$. Therefore, the probability that there is a cluster-head in a regular node's communication range is $\pi r^2/M^2$. As a result, the probability that each regular node can communicate with at least one cluster-head directly is :

$$P=(1-(1-\pi r^2/M^2)^{N_c})^n \tag{13}$$

From Formula (13), we have :

$$\begin{aligned}
&logP = nlog(1-(1-\pi r^2/M^2)^{N_c})\\
&\Longrightarrow 1-(1-\pi r^2/M^2)^{N_c} = 2^{logP/n}\\
&\Longrightarrow (1-\pi r^2/M^2)^{N_c} = 1-2^{logP/n}\\
&\Longrightarrow N_c log(1-\pi r^2/M^2) = log(1-2^{logP/n})\\
&\Longrightarrow N_c = log(1-2^{logP/n})/log(1-\pi r^2/M^2)
\end{aligned} \tag{14}$$

Therefore, if $N_c$ is set to be equal to or larger than $log(1-2^{logP/n})/log(1-\pi r^2/M^2)$, each regular node can communicate with at least one cluster-head directly and the network lifetime is maximized, where $0<P<1$. Considering the high cost of each cluster-head, in order to maximize the network lifetime with the lowest cost, $log(1-2^{logP/n})/log(1-\pi r^2/M^2)$ is the upper bound of the number of the cluster-heads. □

**Corollary 1 :** In a network with $n$ regular nodes and $N_c$ cluster-heads, if $N_c < log(1-2^{logP/n})/log(1-\pi r^2/M^2)$, the extended implementation of MITT achieves an approximate ratio of $\Omega(\log(n/N_c)/\log\log(n/N_c))$.

*Proof* : Since the network contains $N_c$ cluster-heads, there are $N_c$ clusters in the network and the number of regular nodes in each cluster scales approximately as $n/N_c$. In each cluster, MITT is performed to construct a *min-max-weight spanning tree*. Combining with Theorem 3, MITT achieves an approximate ratio of $\Omega(\log(n/N_c)/\log\log(n/N_c))$ in each cluster. □

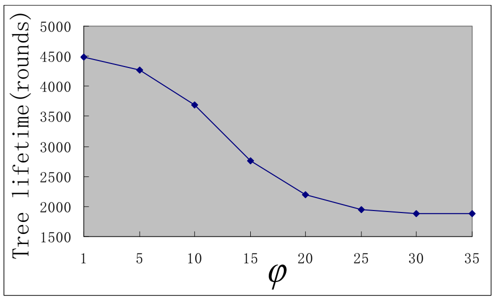
## 5 Simulations

The simulations are assumed to be performed in a square field of 100m×100m, in which nodes are randomly dispersed. Each node in the field is assigned an initial energy level, which is randomly selected from the set of [0.5, 0.6, 0.7, 0.8, 0.9, 1] Joules(J). Each node produces 16 bytes of data at each round. The transmission range of nodes is set to 25m. According to previous measurements [33], the transmission power is about two times the reception power, i.e., $E_{tx}$=2$E_{rx}$. Therefore, $E_{rx}$ is set to 50nJ/bit and $E_{tx}$ is 100nJ/bit. We mainly concern about the problem of finding a maximum lifetime tree, so the parameters $k_1$ and $k_2$ are set to large enough values to enable MITT to be performed without any constraints, e.g., $k_1$=$n$ and $k_2$=$n^3$. In data gathering without aggregation, large amounts of data should be transmitted to the sink and the nodes close to the sink will suffer from heavy loads of data transmissions. In order to avoid congestion and retransmission among the nodes, we assume that there are effective congestion control mechanisms in the network.

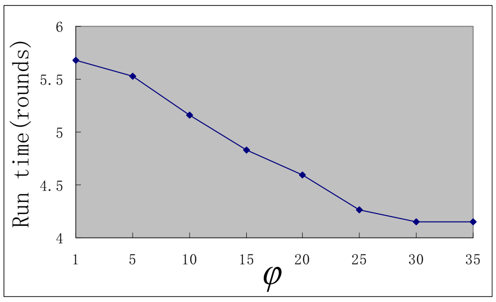
The simulations are performed on a personal computer (PC) with a Pentium 4, 2.8 GHz processor and 1 GB RAM. The effect of parameter $\varphi$ on MITT is first evaluated. Then, the network lifetime achieved by MITT is examined. Finally, the effect of the cluster-heads' number on the extended implementation of MITT is evaluated. All simulations are performed 20 times and the average values of their results are took as the final results.

### 5.1 Effect of $\varphi$

In MITT, $\varphi$ is an important parameter that affects the classification of nodes. Therefore, the performance of



(a) Effect of $\varphi$ on tree lifetime



(b) Effect of $\varphi$ on run time

**FIGURE 7** Effect of $\varphi$

MITT is examined under different values of $\varphi$. Assume that there are 100 nodes in the field, and the sink is located at the center of the field. The tree lifetime achieved by MITT under different values of $\varphi$ is shown in Figure 7(a), and the corresponding run time of MITT is shown in Figure 7(b).

In Figure 7(a), there is a trend that the tree lifetime achieved by MITT decreases as $\varphi$ increases. In Figure 7(b), the run time of MITT also decreases with the increase of $\varphi$. That is because the numbers of nodes in $V_1$ and $V_2$ increase with the increase of $\varphi$, and the number of nodes in $V_3$ decreases at the same time. Therefore, the nodes in $V_1$ are hard to find the nodes in $V_3$ to transfer their descendants. As a result, when $\varphi$ is increased, the algorithm terminates quickly and just constructs a tree with short lifetime.

## 5.2 Network lifetime

In this subsection, four typical algorithms : PEDAP, PEDAP-AP, MNL and LOCAL-OPT are selected to compare with MITT. Assume that the network instances comprise 100, 150, 200, 250, 300, 350, and 400 nodes, respectively. In order to examine the scalability of MITT, two scenarios are considered : (1) The sink is located at the center of the field (its coordinate is (50, 50)) ; (2) The sink is located at the edge of the field (its coordinate is (100, 50)).

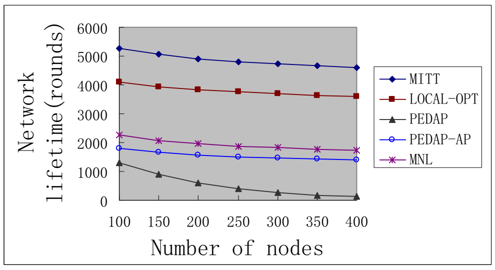
Since MITT and the four algorithms are all performed at the sink, their implementations are the same at each round, e.g., in the set-up phase, the sink collects the information of nodes' energies and the network topology to compute a tree, and then informs the tree information to all nodes ; in the steady-state phase, all nodes transmit their data to the sink by the tree.

In the set-up phase of each round, each node will consume some extra energy in aspects such as transmitting its energy's and its neighbors' information to the sink and receiving the tree information from the sink. This part of energy consumption is almost the same at each round, and it is independent of the algorithms. Moreover, it is insignificant compared with the energy consumption in the steady-state phase[4][30]. Therefore, we ignore this part of energy consumption and mainly concern with the energy consumption in the steady-state phase. The network lifetime and run time of the algorithms in different network instances are shown in Figure 8.

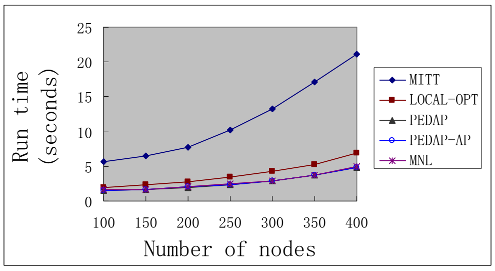
In Figure 8(a), the network lifetime achieved by the algorithms in scenario 1 is shown. Since MITT can construct a *min-max-weight spanning tree* to effectively conserve the energies of nodes at each round, it achieves longer network lifetime than other algorithms. LOCAL-OPT achieves longer network lifetime than PEDAP, PEDAP-AP and MNL. However, the network lifetime achieved by LOCAL-OPT is lower than that of MITT. The network lifetime achieved by PEDAP is the lowest, because it is not energy-aware. PEDAP-AP improves PEDAP and achieves longer network lifetime than PEDAP. The network lifetime achieved by MNL is longer than that of PEDAP-AP and PEDAP, but it is lower than that of LOCAL-OPT and MITT.

In Figure 8(c), the network lifetime achieved by the algorithms in scenario 2 is shown. Compared with Figure 8(a), the network lifetime achieved by MITT decreases by about 48%, and the network lifetime achieved by PEDAP, PEDAP-AP, MNL and LOCAL-OPT decreases by about 35%, 24%, 23% and 47% in average, respectively. This is because the number of the sink's neighbors decreases, when the sink is located at the edge of the field. Therefore, the sink's neighbors have to relay more data from other nodes further away from the sink and die sooner. As a result, the network lifetime achieved by all algorithms decreases. However, though the network lifetime achieved by MITT decreases the largest, it is still longer than that of other algorithms.

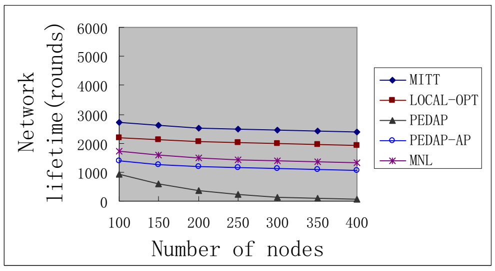
In Figure 8(b) and Figure 8(d), wherever the sink is located, MITT needs more time than other algorithms to terminate. However, since all of above algorithms are performed at the sink and the sink has powerful computation ability, the run time of the algorithms is not our main concern. Therefore, though MITT has higher time complexity than other algorithms, it is still a better choice for data gathering without aggregation because it can achieve longer network lifetime than existing algorithms.



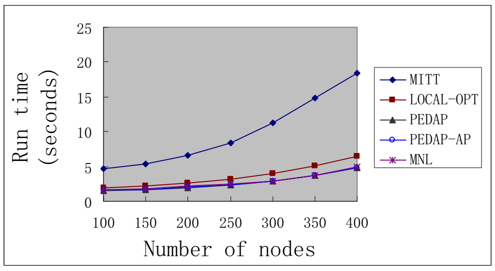
(a) Network lifetime in scenario 1



(b) Run time in scenario 1



(c) Network lifetime in scenario 2

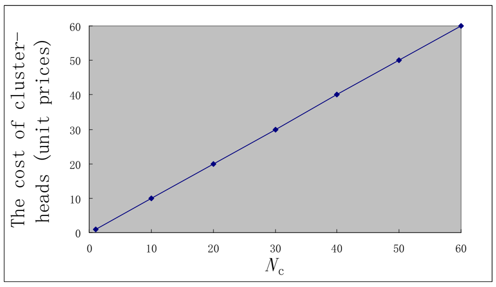


(d) Run time in scenario 2

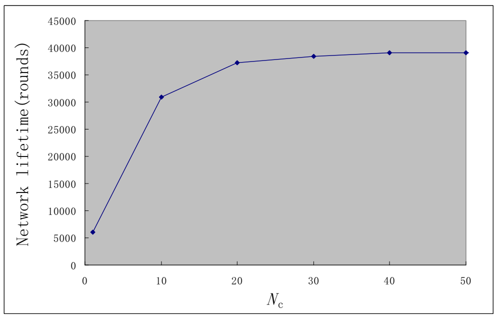
**FIGURE 8** Network lifetime and run time of algorithms

## 5.3 Effect of $N_c$

In the extended implementation of MITT in large-scale networks, $N_c$ cluster-heads are deployed in the network to divide the network into small clusters. According to Corollary 1, the extended implementation of MITT has an approximate ratio of $\Omega(\log(n/N_c)/\log\log(n/N_c))$. Therefore, the number $N_c$ of the cluster-heads would affect the performance of the algorithm, i.e., if $N_c$ is a large value, the approximate ratio



(a) Effect of $N_c$ on the cost of the cluster-heads



(b) Effect of $N_c$ on the network lifetime

**FIGURE 9** Effect of $N_c$

is a small value, and vice versa. However, the costs of the cluster-heads are high, it is unrealistic to deploy large number of cluster-heads in the network. In this subsection, the relationship between the number of cluster-heads and the network lifetime achieved by MITT is examined.

Assume that there are $N_c$ cluster-heads and 100 regular nodes in the network. The cost of a cluster-head is set to 1 unit price. The communication range of the regular nodes is $r$=25m. The effects of $N_c$'s different values on the cost of the cluster-heads and the network lifetime are shown in Figure 9.

In Figure 9(a), the cost of the cluster-heads increases proportionally with the increase of the number of the cluster-heads. That is because that the cost of a cluster-head is set to 1 unit price, and $N_c$ cluster-heads cost $N_c$ unit prices accordingly. Since the cost of the cluster-heads increases with the increase of the number of the cluster-heads, large number of cluster-heads will bring high cost for the deployment of the network. According to the low-cost characteristic of sensor network, the number of cluster-heads should not be a large value.

In Figure 9(b), when $N_c$ <40, the network lifetime increases with the increase of $N_c$. That is because the increase of $N_c$ helps that more nodes can communicate with at least one cluster -head directly, and these part of nodes do not need other nodes relay their data and save the other nodes' energy. Therefore, the increase of $N_c$ will benefit to increase the network lifetime. However, when

$N_c \geq 40$, the network lifetime achieves its maximum and would not be increased. On the other hand, when $N_c \leq 20$, the network lifetime increases quickly with the increase of $N_c$. When $N_c > 20$, the effect of the increase of $N_c$ on the network lifetime becomes smaller. Therefore, the increasing of the number of the cluster-heads will not always bring proportional increasing of the network lifetime. As a result, the number of the cluster-heads $N_c$ should not larger than its upper bound.

## 6 Conclusions

In this paper, the problem of constructing a *min-max-weight spanning tree* for the data gathering without aggregation is studied. The problem is proved to be NP-complete, and our goal is to maximize the network lifetime. A novel approximation algorithm MITT is proposed for solving the problem. MITT achieves an approximation ratio of $\Omega(\log n/\log\log n)$. Moreover, a solution for extending MITT to large-scale networks is presented. Simulation results show that MITT can achieve longer network lifetime than existing algorithms. In the future, we will research a new scheme with low time complexity and a distributed scheme.

## Acknowledgement

This paper is an extended and revised version of our previous paper ”An Efficient Algorithm for Constructing Maximum lifetime Tree for Data Gathering Without Aggregation in Wireless Sensor Networks”, in proceeding of the IEEE 29th Conference on Computer Communications (INFOCOM2010), mini conference, San Diego, CA, USA, March 15-19, 2010, pp.356-360. Our work is sponsored by the National Natural Science Foundation of China under Grant Nos.61103245, 60963022, 61262003; the Guangxi Natural Science Foundation under Grant Nos. 2012GXNSFBA053163, 2013GXNSFGA019006.

The authors are grateful to the anonymous referee for a careful checking of the details and for helpful comments that improved this paper.

## Références

[1] Lan F. Akyildiz, Welljan Su, Yogesh Sankarasubramaniam, Erdal Cayirci, Wireless sensor networks : a survey, Journal of computer networks, **38**, 393-422 (2002).

[2] Johannes Gehrke, Samuel Madden, Query processing in sensor networks, Pervasive Computing, **3**, 46-55 (2004).

[3] Soonmok Kwon, Jeong-gyu Kim, Cheeha Kim, An efficient tree structure for delay sensitive data gathering in wireless sensor networks, In : Proc. of the 22nd International Conference on Advanced Information Networking and Applications (AINAW 2008), 25-28, 738-743 (2008).

[4] Yan Wu, Sonia Fahmy, Ness B. shroff, On the Construction of a Maximum-Lifetime Data Gathering Tree in Sensor Networks : NP-Completeness and Approximation Algorithm. In : Proceedings Of The IEEE 27th Conference on Computer Communications (INFOCOM2008), 356-360 (2008) .

[5] Tung-Wei Kuo, Ming-Jer Tsai, On the construction of data aggregation tree with minimum energy cost in wireless sensor networks : NP-completeness and approximation algorithms, In : Proceedings Of The IEEE 31th Conference on Computer Communications (INFOCOM2012), 2591-2595 (2012).

[6] Maleq Khan, Gopal Pandurangan, Anil Vullikanti, Distributed Algorithms for Constructing Approximate Minimum Spanning Trees in Wireless Sensor Networks, IEEE Transactions on Parallel and Distributed Systems, **20**, 124-139 (2009).

[7] Wendi Rabiner Heinzelman, Anantha Chandrakasan, Hari Balakrishnan, Energy-efficient communication protocol for wireless microsensor networks. In : Proc. of the Hawaii Int’l Conf. on System Sciences. San Francisco : IEEE Computer Society, 3005-3014 (2000).

[8] Ossama Younis, Sonia Fahmy, HEED : A hybrid energy efficient distributed clustering approach for ad hoc sensor networks. IEEE Transactions on Mobile Computing, **3**, 366-379 (2004).

[9] Chamam Ali, Pierre Samuel, On the Planning of Wireless Sensor Networks : Energy-Efficient Clustering under the Joint Routing and Coverage Constraint, IEEE Transactions on Mobile Computing, **8**, 1077-1086 (2009).

[10] Hongbin Chen, Chi K Tse, Jiuchao Feng, Minimizing effective energy consumption in multi-cluster sensor networks for source extraction, IEEE Transactions on Wireless Communications, **8**, 1480-1489 (2009).

[11] Jun Fang, Hongbin Li, Power constrained distributed estimation with cluster-based sensor collaboration, IEEE Transactions on Wireless Communications, **8**, 3822-3832 (2009).

[12] Wang Pu, Dai Rui, Akyildiz Ian F., Collaborative Data Compression Using Clustered Source Coding for Wireless Multimedia Sensor Networks, in Proc.of The 29th IEEE Conference on Computer Communications (INFOCOM 2010), 327-336 (2010).

[13] Roseline R. A., Sumathi P., Local clustering and threshold sensitive routing algorithm for Wireless Sensor Networks, in Proc.of the 2012 International Conference on Devices, Circuits and Systems (ICDCS 2012), 365-369 (2012).

[14] Stephanie Lindsey, Cauligi S. Raghavendra, PEGASIS : Power efficient gathering in sensor information systems. In : Proc. of the IEEE Aerospace Conf. San Francisco : IEEE Computer Society, 1-6 (2002).

[15] Sung-Min Jung, Young-Ju Han, Tai-Myoung Chung, The Concentric Clustering Scheme for Efficient Energy Consumption in the PEGASIS9th IEEE International Conference on Advanced Communication Technology, Phoenix Park, Gangwon-Do, Republic of Korea, 260-265 (2007).

[16] Chen Kuong-Ho, Huang Jyh-Ming, Hsiao Chieh-Chuan, CHIRON : An energy-efficient chain-based hierarchical

routing protocol in wireless sensor networks, Wireless Telecommunications Symposium(WTS2009), 1-5 (2009).

[17] Yu Jae Duck, Kim Kyung Tae, Jung Bo Yle, Youn Hee Yong, An Energy Efficient Chain-Based Clustering Routing Protocol for Wireless Sensor Networks, International Conference on Advanced Information Networking and Applications Workshops(WAINA2009), 383-388 (2009).

[18] Jisoo Shin, Changjin Sun, CREEC : Chain routing with even energy consumption, Journal of Communications and Networks, **13**, 17-25 (2011).

[19] Palazzo R.P., A network flow approach to convolutional codes, IEEE Transactions on Communications, **43**, 1429-1440 (1995).

[20] Konstantinos Kalpakis, Koustuv Dasgupta, Parag Namjoshi, Maximum lifetime data gathering and aggregation in wireless sensor networks, in Proc. of IEEE International Conference on Networking, (2002).

[21] Kalpakis K., Shilang Tang, A combinatorial algorithm for the Maximum Lifetime Data Gathering And Aggregation problem in sensor networks, 2008 International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2008), 1-8 (2008).

[22] Hai Liu, Pengjun Wan, Xiaohua Jia, Maximal Lifetime Scheduling for Sensor Surveillance Systems with K Sensors to One Target, IEEE Transactions on Parallel and Distributed Systems, **17**, 1526-1536 (2006).

[23] Hai Liu, Xiaohua Jia, Peng-Jun Wan, Chih-Wei Yi, Makki S. K., Pissinou N., Maximizing Lifetime of Sensor Surveillance Systems, IEEE/ACM Transactions on Networking, **15**, 334-345 (2007).

[24] Qun Zhao, Gurusamy M., Lifetime Maximization for Connected Target Coverage in Wireless Sensor Networks, IEEE/ACM Transactions on Networking, **16**, 1378-1391 (2008).

[25] Tan HO, Korpeoglu I., Power efficient data gathering and aggregation in wireless sensor networks. SIGMOD Record, **32**, 66-71 (2003).

[26] Qing Zhang, Zhipeng Xie, Weiwei Sun, Baile Shi, Tree Structure Based Data Gathering for Maximum Lifetime in Wireless Sensor Networks, 7th Asia-Pacific Web Conference (APWeb 2005), Shanghai, China, 513-522 (2005).

[27] Weifa Liang, Yuzhen Liu, Online data gathering for maximizing network lifetime in sensor networks, IEEE Transaction on mobile computing, **6**, 2-11 (2007).

[28] Chiranjeeb Buragohain, Divyakant Agrawal, Subhash Suri, Power Aware Routing for Sensor Databases, In : Proceedings of the IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005), 13-17, 1747-1757 (2005) .

[29] Zissimopoulos V., Paschos V. T., Pekergin F., On the approximation of NP-complete problems by using the Boltzmann machine method : the cases of some covering and packing problems, IEEE Transactions on Computers, **40**, 1413-1418 (1991).

[30] Zhao Cheng, Perillo M., Heinzelman W. B., General network lifetime and cost models for evaluating sensor network deployment strategies, IEEE Transactions on Mobile Computing, **7**, 484-497 (2008).

[31] Mhatre V.P., Rosenberg C., Kofman D., Mazumdar R., Shroff N., A minimum cost heterogeneous sensor network with a lifetime constraint, IEEE Transactions on Mobile Computing, **4**, 4-15 (2005).

[32] Haibo Zhang, Hong Shen, Balancing Energy Consumption to Maximize Network Lifetime in Data-gathering Sensor Networks, IEEE Transactions on Parallel and Distributed Systems, (2009).

[33] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, Directed Diffusion : A Scalable and Robust Communication Paradigm for Sensor Networks, In : Proceedings of the sixth Annual ACM International Conference on Mobile Computing and Networking (MOBICOM 2000), 56-67 (2000).

**Junbin Liang** received the BS and MS degrees in computer science and technology from Guangxi University, Nanning, Guangxi, China, in 2000 and 2005, respectively. He received the Ph.D. degree in computer science and technology from Central South University, Changsha, Hunan, China. He is currently an associate professor at the school of Computer and Electronics Information of Guangxi University. His research interests include wireless sensor network, mobile and pervasive computing, and parallel and distributed computing.

**Taoshen Li** received his Ph.D. degree in Computer Application Technology from Central South University of China in 2008. He is a professor at the school of Computer and Electronics Information of Guangxi University, and he is a senior member of China Computer Federation(CCF). His research interests include distributed database, network computing, and CAD theory and application, etc.