An Energy-Efficient Mechanism for Self-Monitoring Sensor Web

Yangfan Zhou Michael R. Lyu

Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, Hong Kong
+852-{3163-4253, 2609-8429}
{yfzhou, lyu}@cse.cuhk.edu.hk

Abstract— A Sensor Web is a network of spatially distributed sensor platforms, which is especially well suited for environmental monitoring. Although sensor nodes of a Sensor Web are critical devices that perform the monitoring work, the low-cost implementation of sensor nodes poses that they are subject to failures and permanent damage. A life-condition monitoring mechanism for sensor nodes is therefore required to ensure the function of a Sensor Web. This paper studies this sensor-node monitoring problem in the domain where in-network sensor nodes are self-monitoring, i.e., the status of each sensor node is monitored by another node. To be energy-efficient, a mechanism for implementing self-monitoring Sensor Webs should minimize the energy required. We propose a formal formulation of this problem and show that it can be solved by finding a minimum spanning tree of the graph constructed by in-network nodes. We provide some distributed algorithms in solving this problem. Simulations are conducted to study the performance of these algorithms.

TABLE OF CONTENTS

1. INTRODUCTION

In recent years, in-situ sensing with small wireless-equipped sensor devices has became a promising technique with the advances in Micro Electro-Mechanical Systems (MEMS) and wireless communication technologies [1][2]. A *Sensor Web* is a network of such spatially-distributed sensor platforms [3][4]. In sensor webs, sensor nodes perform in-situ sensing task on some physical phenomena of interest and they communicate with wireless interface. The data on the physical phenomena can thus be conveyed to data collecting nodes (the base stations) in a multi-hop manner.

The low-cost implementation and unattended operational manner make Sensor Webs especially well suited for environmental monitoring. Today, a variety of Sensor Webs have been field-deployed and field-tested in many environments. Examples include that in The Botanical Gardens at The Huntington Library [5] for monitoring botanical conditions [6], that in Antarctica to monitor microclimate conditions for extreme life detection [7], and that for in-situ exploration of gaseous biosignatures [8]. In-situ sensing with Sensor Webs is also a promising technique in aerospace applications. We can imagine that in future applications Sensor Webs will be deployed in outer space or on other planets to collect physical data that cannot easily be captured by remote sensing techniques. For example, a Sensor Web can be constructed on the Mars to collect some seismic data in order to analyze the geological features of the planet.

Although sensor nodes in a Sensor Web are critical devices to perform monitoring work, their low-cost implementation suggests that they are subject to failures and permanent damages. Moreover, sensor nodes are battery-powered devices. They cease to function when their batteries drain out. A Sensor Web is usually working in an unattended manner (*e.g.*, in future aerospace applications where Sensor Webs are deployed in outer space), an automatic life-condition monitoring mechanism for sensor nodes is therefore required to locate the failure nodes, which can help reorganize the network or facilitate manual repair of the failure nodes. Such a life-condition monitoring mechanism is important to ensure the function of a Sensor Web.

An example network in shown in Figure 1 where the black solid circles denote the sensor nodes. The network is employed to detect forrest fires. In case that a node fails (*e.g.*, the node with a cross in the figure), it cannot sense temperature of the circular area. Then, if without a sensor node life-condition monitoring mechanism, the network cannot be notified about the node failure and perform consequent repair actions. As a result, the network is not aware of a fire taking place in that area, which may cause a great disaster.

1-4244-0525-4/07/$20.00/
IEEEAC paper # 1629

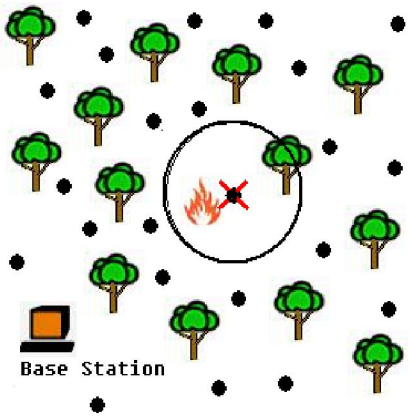

Figure 1. An example network

This paper studies the sensor-node monitoring problem in the domain where the network is self-monitoring, *i.e.*, the status of each in-network sensor node is monitored by other in-network nodes. We consider that the in-network sensor nodes perform the monitoring tasks in a passive manner. A sensor node which is in charge of the monitoring task of its peer waits for *heart-beat packets* that are sent out periodically by the peer. If it has not received any heart-beat packet from the peer for a given period of time, it regards that the peer has failed. Then, it can directly send an *alarm message* to inform the base station immediately.

Due to the unattended operational manner of Sensor Webs, recharging or replacing the batteries of sensor nodes is not convenient, or sometimes even impossible (*e.g.*, in future aerospace applications where Sensor Webs are deployed in outer space or on other planets). A mechanism for the self-monitoring of Sensor Web should be energy-efficient: the energy required to perform self-monitoring should be minimized.

Sending heart-beat packets is energy-consuming. In this paper, we consider each sensor node can adjust the power level of its wireless transmitters, which is the usual case in typical sensor node implementations. For example, the Berkeley Mica Mote [9] provides such program interfaces. To save energy, a sensor node adjusts the power level of its wireless transmitter to a minimum value under the constraint that the heart-beat packets sent by this node can just reach its intended destination of the packets. The energy consumption of transmitting the heart-beat packets can thus be reduced.

With transmitter power control, obviously, the energy required for performing self-monitoring depends on the distances between sensor nodes to their peers that monitor them. In order to achieve energy-efficiency, it is necessary to investigate how to organize sensor nodes in performing the self-monitoring tasks (*i.e.*, how to assign which nodes to monitor which peers of the nodes).

In this paper, we first formulate this specific self-monitoring problem and provide a theoretical solution for the problem. Because the scale of a Sensor Web can be very large, often containing hundreds of sensor nodes, exchanging information within the whole network is very expensive. A practical solution therefore should be implemented in a fully distributed manner. Moreover, the topology of a Sensor Web may change frequently if a sleeping/working scheduling mechanism is employed to exploit the redundancy of a dense network. The self-monitoring problem should be solved each time the network topology changes. So, the process to find a solution of this problem should also by itself be energy-efficient. The above considerations motivate us to further study several distributed algorithms in implementing the solution of the self-monitoring problem.

Our study differs from what was conducted previously [10] in the objective of the monitoring problem. In [10], Wang *et al* consider the sensor nodes which are monitoring their peers consume energy while the sensor nodes which are being monitored do not consume energy. This assumes that no packet-exchanging is required in conducting the self-monitoring tasks. They formulate the problem as how to minimize the number of sensor nodes that are monitoring their peers. We, on the other hand, consider that the monitoring tasks are performed in a passive manner, which require the monitored sensor nodes to send heart-beat packets. We formulate the problem as how to minimize the energy required to send heart-beat packets.

The rest of the paper is organized as follows. In Section 2, we formulate the self-monitoring problem according to the features of Sensor Webs. Section 3 discusses the theoretical solution of this problem and provide some properties of the resulting topology. Section 4 elaborates several distributed algorithms in solving the self-monitoring sensor web problem. In Section 5, we present our simulation results. Conclusion remarks are provided in Section 6.

2. Problem Formulation

2.1. Communication models

In Sensor Webs, packets transmitted from node u can be successfully received by the destination node v if the transmitter power setting of node u satisfies the following condition [11]:

$$Pr(u) \geq c \cdot (D(u,v))^n. \tag{1}$$

Here c is a constant whose value is related to the system parameters such as the wavelength of the wireless signal, the antenna gains, and the threshold that a signal can be successfully detected in the destination node. n is the signal fading factor whose value is typically in the interval $(2, 5)$ in an application environment. $D(u, v)$ is the Euclidian distance between node u and node v, namely,

$$D(u,v) = \|X(v) - X(u)\|, \tag{2}$$

in which $X(\cdot)$ denotes the physical location of a node.

We consider the optimal transmitter power level for node u to send a packet to node v is:

$$Pr(u) = c \cdot (D(u,v))^n = c \cdot \|X(v) - X(u)\|^n. \quad (3)$$

We assume that each sensor node can know its approximate physical location. This is true in most application cases as the location information is usually required to identify where the phenomena of interest take place. The approximate location information is obtainable if each sensor node carries a GPS receiver or if some localization algorithms (*e.g.*, [12]) are employed.

2.2. Self-monitoring Sensor Webs

Sensor nodes perform self-monitoring tasks in a passive manner. Each sensor node u actively sends out heart-beat packets periodically to another node v (note that node v can be the base station) which is in charge of monitoring node v. If node v has not received any heart-beat packet for a given period of time, it regards that node u has failed. Then, if node v is not the base station, it can send an alarm message to inform the base station[1].

We define a Sensor Web is *self-monitoring* if the base station can be aware of the failure of any sensor nodes (*i.e.*, failures of any subset of nodes which does not contain the base station) with the above monitoring mechanism.

In the following discussion, $v \rightarrow u$ denotes the monitoring relation between v and u in which v is monitoring u, *i.e.*, u should periodically send heart-beat packets to v. Because packet transmission costs energy, to achieve energy-efficiency, we consider a node is monitored by exactly one node. That is, each node should send heart-beat packets to exactly one node, consequently,

$$v_1 \rightarrow u \ \& \ v_2 \rightarrow u \Longrightarrow v_1 = v_2 \quad (4)$$

Let us introduce another relation $\rightsquigarrow$: $v \rightsquigarrow u$ denotes the relation between v and u which is defined recursively as follows:

$$\begin{aligned} &\text{if } v \rightarrow u, \text{ then } v \rightsquigarrow u; \\ &\text{if } v \rightarrow w \text{ and } w \rightsquigarrow u, \text{ then } v \rightsquigarrow u. \end{aligned} \quad (5)$$

It is easy to see that $v \rightsquigarrow u$ implies that $v \rightarrow w_1, w_1 \rightarrow w_2, ..., w_i \rightarrow u$, where the number of intermediate nodes w_1, w_2, ..., w_i can be larger or equal to zero.

In a self-monitoring Sensor Web, at least one node in any subset of nodes without the base station should be monitored by a node that is not in the subset. Otherwise, the base station cannot be aware of the failure of the whole subset, resulting in a Sensor Web which is not self-monitoring.

Therefore, the monitoring relations cannot form *loops*. The reason is that if a loop exists, according to the monitored-by-only-one feature described in Equation (4), no sensor nodes in such a loop can be monitored by any node outside the loop, which poses that the Sensor Web is not self-monitoring.

Hence, relation $\rightsquigarrow$ is a *strict partial order* relation, which formally satisfies:

$$\begin{aligned} &\diamond \ \textit{irreflexivity}: \ \neg\exists u, \text{ such that, } u \rightsquigarrow u; \\ &\diamond \ \textit{antisymmetry}: \ \neg\exists u, v, \text{ such that, } u \rightsquigarrow v \text{ and } v \rightsquigarrow u; \\ &\diamond \ \textit{transitivity}: \ \text{if } v \rightsquigarrow w \text{ and } w \rightsquigarrow u, \text{ then } v \rightsquigarrow u; \end{aligned} \quad (6)$$

where $\neg$ means "not" and $\exists$ means "exist".

The irreflexivity property means that a sensor node cannot monitor itself. The antisymmetry property means that no monitor-relation loops can be formed. The transitivity is directly obtained from the definition of the $\rightsquigarrow$ relation.

2.3. Energy consumption model

$Energy(v \rightarrow u)$ denotes the energy required for maintaining $v \rightarrow u$ per second. Because the energy consumption for receiving a heart-beat packet and processing a heart-beat packet is constant, we consider that $Energy(v \rightarrow u)$ is the energy consumption for a node to send heart-beat packets per second. With transmitter power control, it is:

$$Energy(v \rightarrow u) = \gamma Pr(u) \quad (7)$$

where γ is a constant related to the packet size and the packet-rate of the heart-beat packets.

2.4. The self-monitoring problem

According to the above discussion, the self-monitoring Sensor Web problem is formulated as follows.

Problem 1: Given a set of sensor nodes S, a base station bs, order the set $S \cup \{bs\}$ with the relation $\rightsquigarrow$ so that $bs \rightsquigarrow u$ ($\forall u \in S$) and the total energy required to maintain the $\rightsquigarrow$ relations among the nodes is minimized.■

In this formulation, we require $bs \rightsquigarrow u$ ($\forall u \in S$) to ensure that the failure of any subset of sensor nodes can be detected by a node that is not in the subset. This is because the failure of any subset of sensor nodes would cut at least one $\rightarrow$ relation (*e.g.*, that denoted by $w \rightarrow v$) whereas w is still working. As a result, w can detect the failure of v and inform the base station. The objective in this problem formulation is to minimize the energy consumption required in a self-monitoring Sensor Web.

[1] In the rest of our discussion, we assume a node can always send the alarm message to the base station. How the alarm message is sent to the base station is *not* the focus of this paper. It can be sent in a multi-hop manner via the paths that route the sensor-reporting data packets on the phenomena of interest, or it can be sent directly to the base station if the transmitter power level of the sensor node can be large enough.

3. THEORETICAL SOLUTION

3.1. Graph model of the relations

It is convenient to denote strict partial order relations among the members in a set with a graph, in which the vertices denote the members and directed edges in the graph denote the relations.

We model a Sensor Web as a weighted graph $G(V, E)$. Here V is the set of vertices denoting the sensor nodes and the base station bs (*i.e.*, $V = S \cup \{bs\}$). E is the set of edges denoting the wireless links from a node to another if the node can directly communicate with the other node at the maximum transmitter power setting[2]. The weight of an edge $e(u, v)$ is defined by:

$$weight(e(u, v)) = c \cdot (D(u, v))^n \tag{8}$$

Obviously, $v \rightarrow u$ can be denoted by an edge with direction from u to v in $G(V, E)$, and $v \rightsquigarrow u$ can be denoted by a path from u to v in $G(V, E)$. The relations $bs \rightsquigarrow u$ ($\forall u \in S$) required in Problem 1 can be denoted by a number of paths in graph $G(V, E)$. The union of these paths forms a subgraph of $G(V, E)$.

According to Equation (7), the energy required to maintain $v \rightarrow u$ is linearly related to the weight of edge defined in Equation (8). The objective of Problem 1, *i.e.*, minimizing the energy consumed to preserve all the $\rightarrow$ relations that are required for maintaining $bs \rightsquigarrow u$ ($\forall u \in V$ and $u \neq bs$), is therefore equivalent to minimizing the weight of the subgraph formed by the union of paths denoting relations $bs \rightsquigarrow u$ ($\forall u \in V$ and $u \neq bs$).

3.2. Theoretical solution

According to the antisymmetry property of $\rightsquigarrow$, there is no loop in the union of paths in $G(V, E)$ denoting the $\rightsquigarrow$ relations between nodes. So the resulting subgraph should be a tree. Because $bs \rightsquigarrow u$ ($\forall u \in V$ and $u \neq bs$), this tree is a spanning tree rooted at the base station bs.

Therefore, the solution of the self-monitoring problem is to find a spanning tree rooted at the base station bs and the weight of the tree is minimized. Obvious, this is a Minimum Spanning Tree (MST) of graph $G(V, E)$. Classic algorithms to find an MST of a graph include Prim's algorithm [13], Kruskal's algorithm [14], and Boruvka's algorithm, *etc* [15].

3.3. Properties of MSTs

To facilitate our later discussions, in the rest of this section, a theorem that describes some properties of the MST is provided. This serves as the foundation of the distributed algorithm described in the following section.

[2]We consider that the base station does not have out-link edges in this model.

Let us define the *distance* between two subgraphs of graph $G(V, E)$ as the minimum weights among the weights of the shortest paths in $G(V, E)$ between any vertex in a subgraph and any vertex in anther subgraph.

For any proper subtree T_0 in the MST of a graph, suppose the subtree has n edges ($n \geq 1$), denoted by e'_1, e'_2, ..., e'_n, connecting to other parts of the MST. Cut these n edges, then the MST is divided into exactly $n + 1$ trees according to the fact that the MST is a tree. Let T_0, T_1, ..., T_n denote these trees.

Lemma 1: Suppose with the above notations, the edge e'_i ($1 \leq i \leq n$) is with minimum weight among e'_1, e'_2, ..., e'_n. $weight(e'_i)$ is also the minimum distance among all distances between T_0 and each tree of T_1, ..., T_n.

Proof: If $weight(e'_i)$ is not the minimum distance among all distances between T_0 and each tree of T_1, ..., T_n, suppose the minimum distance is d which is the weight of a path from a node in T_j (T_j is some tree in T_1, ..., T_n) to a node in T_0. Add this path to the MST and cut e'_j to form a new subgraph. Purge any edges that result in loops in this subgraph to avoid loops and form a new spanning tree. Because $weight(e'_j) \geq weight(e'_i) > d$, the weight of the new spanning tree is smaller than the MST, which contradict the definition of an MST. ■

Theorem 2: For any subtree in the MST of G(V, E), suppose the subtree has m edges ($m \geq 1$), denoted by e_1, e_2, ..., e_m, connecting to other parts of G(V, E) and the edge e_k ($1 \leq k \leq m$) is with minimum weight among e_1, e_2, ..., e_n. Then, e_k is also in an MST of G(V, E).

Proof: We prove this by contradiction again. Consider that the subtree is the T_0 in Lemma 1. Assume e_k is not in an MST. Because an MST is a subgraph of G(V, E), $\{e'_1, e'_2, ..., e'_n\} \subseteq \{e_1, e_2, ..., e_m\}$. As e_k connects a vertex u in T_0 to a vertex v that is not in T_0, vertex v must be in one of the trees T_1, ..., T_n. Denote the tree T_p ($1 \leq p \leq n$). Then the edge connecting T_p and T_0 is e'_p and $weight(e'_p) \geq weight(e'_i)$.

According to Lemma 1, $weight(e'_i)$ is less than or equal to the distance between T_0 and T_p. This distance is less than or equal to $weight(e_k)$ because edge e_k connects T_0 and T_p. As $weight(e_k)$ is the minimum among all weights of $e_1, e_2, ..., e_m$, $weight(e_k) = weight(e'_i) \leq weight(e'_p)$. Remove e'_p from the MST and add e_k to the MST, the weight of resulting spanning tree is less than or equal to the MST, which is also an MST. It contradicts that e_k is not in an MST. ■

An example is demonstrated in Figure 2. In this example, suppose that the edges (v_1, v_3), (v_2, v_4), (v_3, v_4), and (v_4, v_5) form a subtree of an MST in a graph. This subtree has eleven edges connecting to the other parts of the graph. Note that we

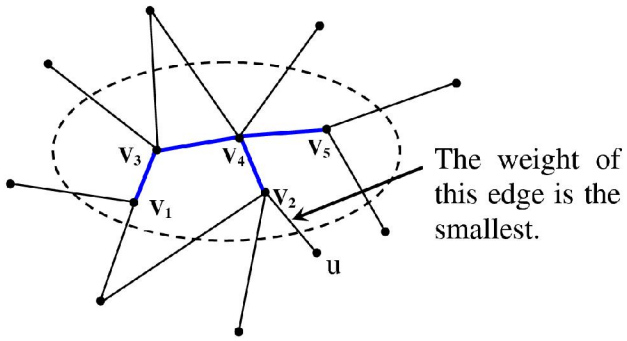

Figure 2. An example showing how to apply Theorem 2

just show the vertices connected to the subtree by an edge in this figure. Among all these eleven edges, the edge (v_2, u) is with the minimum weight. Theorem 2 tells us that this edge is also in an MST of the graph. That is to say, the edges (v_1, v_3), (v_2, v_4), (v_3, v_4), (v_4, v_5), together with (v_2, u), form a subtree of an MST.

4. Solutions in a Distributed Environment

Theoretically, finding an MST in $G(V, E)$ is simple in terms of computational complexity of the algorithm. However, in most applications of Sensor Webs, the scale of a Sensor Web can be very large containing hundreds of sensor nodes. Distributed algorithms should be designed to attack this Sensor Web self-monitoring problem. This section provides some example algorithms we design.

For distributed networks like Sensor Webs, localized algorithms, in which each node determines which edge is in the MST with only the information of the nodes within some constant hops, is highly desired for constructing the MST. Unfortunately, such an algorithm does not exist [16]. Therefore, in practical applications, it is inevitable for a distributed algorithm to exchange a large number of packets to construct an MST as each in-network node does not have a global picture of the whole network due to the large scale of the network. The performance in terms of the number of packets that should be exchanged is the main consideration in our algorithm design.

We first design an distributed algorithm based on Theorem 2 to find the optimal solution of Problem 1, *i.e.*, to find an MST of G(V, E). Since exchanging information within the whole network is very expensive, we then provide several approximation algorithms which exchange less packets to find suboptimal solution of Problem 1, *i.e.*, to find a spanning tree of G(V, E) of which the weight is comparable to that of an MST.

Note that although constructing such spanning trees for finding the monitoring relations (*i.e.*, the $\rightarrow$ relations between nodes) costs less energy, maintaining a self-monitoring Sensor Web costs more as the weights of the corresponding spanning tree is larger than that of an MST. Such a trade-off will be shown in our experimental study in Section 5.

4.1. Algorithm 1: Finding an MST

This algorithm contains two processes. In the first process, a set of disjoint subtrees of an MST are found. These subtrees contain all in-network nodes. In the second process, these subtrees join together to form the MST.

In the first process, a subtree, say T_i, is found as follows. First, a node informs its nearest neighbor to form subtree T_i. The newly-added node then takes over the work and informs its nearest neighbor to join T_i. This step continues and in this way T_i grows. It stops when a newly-added node, say node u, finds that its nearest neighbor is already in a subtree. If this subtree is not T_i, the neighbor is informed and the two subtrees are merged. Then node u hands over to another neighbor which is not in any subtree. And this neighbor begins to form another subtree T_{i+1} of the MST.

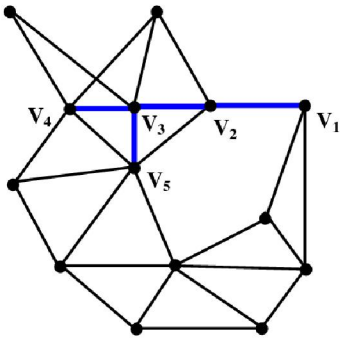

Figure 3. An example showing how the first process of Algorithm 1 works

Let us demonstrate this process by an example. Consider the graph shown in Figure 3. At first, v_1 adds itself to a subtree, say T_1. It knows its nearest neighbor is v_2, and informs v_2 to join T_1. v_2 then takes over the work and knows that its nearest neighbor is v_3. It informs v_3 to join T_1. v_3, similarly, informs its nearest neighbor v_4 to join T_1. Now v_4 takes over the work and knows that its nearest neighbor is v_3. As v_3 is already in T_1, v_4 hands over to a neighbor (*i.e.*, v_5 in this case) to find a new subtree. v_5 adds itself to a new subtree, say T_2. And v_5 knows it nearest neighbor is v_3, which is already in T_1. Therefore, v_3 is informed that T_2 and T_1 should be merged. The resulting subtree of an MST is then composed by the *thick* edges shown in Figure 3. v_5 then hands over to a neighbor that does not belong to any subtree to continue the process. In this way, the other subtrees of the MST can be found.

In the second process, nodes in a subtree should exchange packets and find out the edge with minimum weight among all the edges connecting the subtree to other subtrees of G(V, E). Then the corresponding two subtrees connected by this edge are merged. This merge procedure continues till all subtrees are merged into one single tree. The resulting tree is the MST.

This process is also demonstrated by an example, which is shown in Figure 4. Suppose T_1, T_2, T_3, and T_4 are subtrees of an MST, which are constructed in the first process. In the second process, through exchanging packets and performing comparisons, T_1 knows that e_3 is the edge with minimum weight among all the edges connecting T_1 to the other subtrees. T_1 and T_4 are then merged into one subtree, *i.e.*, $T_1 \cup T_4 \cup \{e_3\}$. This new subtree, similarly, can know that e_1 is the edge with minimum weight among all the edges connecting the subtree to the other subtrees. Then T_3 joins the subtree via e_1 and the subtree becomes $T_1 \cup T_4 \cup \{e_3\} \cup T_3 \cup \{e_1\}$. Finally, T_2 joins the subtree via e_2 in a similar way and the resulting tree is now $T_1 \cup T_4 \cup \{e_3\} \cup T_3 \cup \{e_1\} \cup T_2 \cup \{e_2\}$. This is the MST.

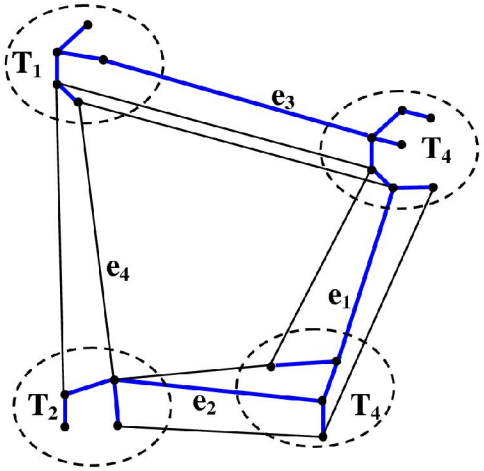

Figure 4. An example showing how the second process of Algorithm 1 works

Algorithm 1 is based on Theorem 2. Because a vertex is always a subtree of an MST, for each node, the edge to its nearest neighbor is always an edge of an MST of $G(V, E)$. Therefore, subtrees constructed in the first process are subtrees of an MST. Also, as a subtree, say T_i, merges into another subtree on the other side of the edge which is with the minimum weight among all the edges connecting T_i to other subtrees of G(V, E), the newly-formed subtree is also a subtree of an MST according to Theorem 2. Therefore, the resulting tree in the second process is an MST eventually.

Nodes should exchange $O(|V|)$ packets ($|\cdot|$ denotes the number of members in a set) totally in the first process. In the second process $O(|V|^2)$ packets should be exchanged.

4.2. Algorithm 2: A variation of Algorithm 1 to find a spanning tree

Because the second process of Algorithm 1 requires much more packets to be exchanged, we provide a variation of Algorithm 1 in which only $O(|V|)$ packets should be exchanged in the second process. But note that the resulting spanning tree in this algorithm is not an MST.

In the first process, a node also informs its nearest neighbor the weight of the edge to its *second* nearest neighbor that is not in any subtree (say, the weight is b). Nodes in a subtree therefore can know which node in the subtree has the lowest b. Then in the second procedure, we do not exchange packets to find out the edge with minimum weight among all the edges connecting the subtree to other subtrees of G(V, E). Instead, the node that has the lowest b merges its subtree to another subtree which contains the node's nearest neighbor that is not in the same subtree as itself. In this way, the packets exchanged in the second procedure is reduced to $O(|V|)$.

4.3. Algorithm 3: Finding MSTs of subnetworks and merging them

In this algorithm, the network area is divided into several subareas and G(V, E) is divided into several subgraphs according to which subarea each node is located in. For each subgraph, Algorithm 1 is performed and an MST of each subgraph are found. Then these MSTs merge into one spanning tree.

4.4. Algorithm 4: A simple greedy algorithm

In this algorithm, a spanning tree is found with a variation of the first procedure in Algorithm 1. Instead of stopping when a node finds that its nearest neighbor belongs to a subtree, the node informs its nearest neighbor that does not belong to the same subtree to join the subtree.

5. Simulation Results

We study the performance of the algorithms discussed in Section 4 with simulations. We randomly deploy n sensor nodes and the base station in a uniform manner. The network area is $400m \times 400m$. In our simulation study, we change n in the range of $(80, 150)$. For each setting of n, different random seeds are selected to generate 10 randomized networks. The algorithms are performed for each network and the results of these 10 networks are averaged. When conducting simulations of Algorithm 3, we partition the network into 2×2 and 3×3 grids.

We first study the performance of the algorithms in terms of the weights of the spanning trees found. Figure 5 demonstrates the results. Note that in these results, c in Equation (8) is set 1, as the value of c is not important for comparison purpose.

We can see that the weights of the resulting spanning trees found by Algorithms 2 and 3 are much larger than the weights of the trees found by Algorithms 1 and 4. This is not strange as Algorithms 2 and 3 are approximation algorithms, which are designed mainly to achieve a smaller number of packets required to find a spanning tree. Algorithm 4, although also an approximate algorithms, finds MSTs of subnetworks. We can see its performance, in terms of the weights of the spanning trees found, is comparable to Algorithm 1.

We show the energy required to perform the algorithms in

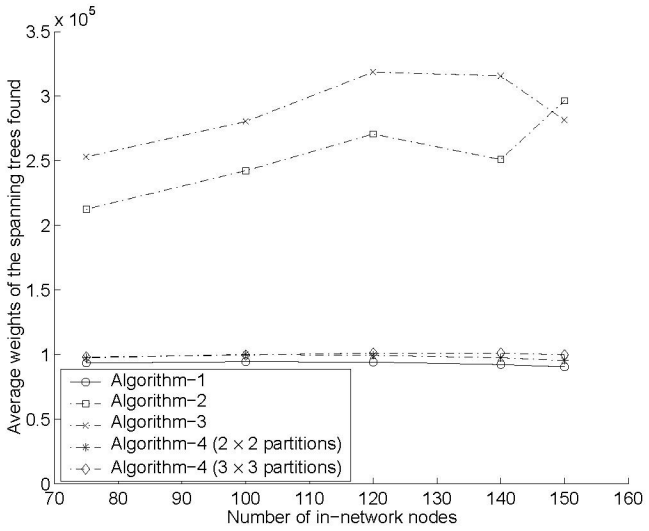

Figure 5. Weights of the spanning trees found

terms of the number of packets exchanged. Figure 6 demonstrates the results. It can be found that Algorithms 2 and 3 require much less packets to be exchanged in finding the spanning tree comparing to the other algorithms. Algorithm 1 requires quite a large number of packets exchanged.

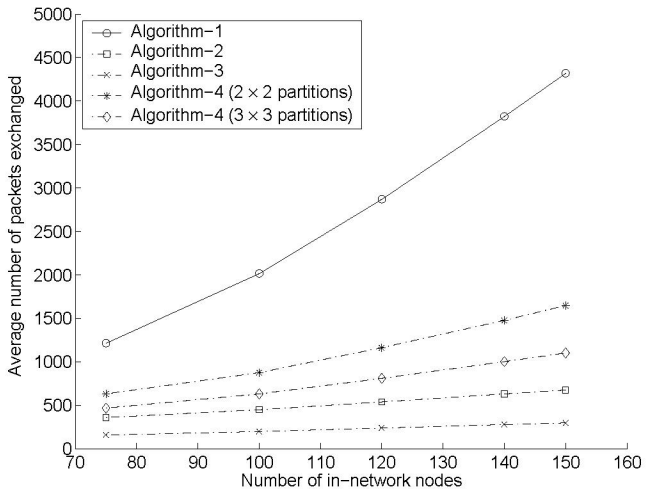

Figure 6. Number of packets that should be exchanged

As usually a Sensor Web contains a large number of redundant nodes to achieve fault tolerance, a node's sleeping/working scheduling mechanism is usually employed to exploit such redundancy. As a result, the topology of a Sensor Web may change frequently (we consider the topology is determined only by the active nodes) and the self-monitoring problem should be solved each time the network topology changes. The overhead (*i.e.*, the number of packets exchanged) in solving the problem with distributed algorithms should be considered.

On the other hand, note that the rate of the heart-beat packets can be very low. As a result, the energy consumption required for self-monitoring a Sensor Web based on the resulting spanning trees found by different algorithms does not differ too much, although the weights of the spanning trees are different. This should be noted especially when the time to perform self-monitoring is not long in case that the network topology changes frequently.

The results in Figure 5 and 6 show that there is a trade-off between the energy cost to find a spanning tree and the energy cost to self-monitoring sensor web, which is related to how frequently the network topology is changing. We cannot draw simple conclusions that one algorithm is superior to the others. Such a trade-off should be carefully taken into account when designing a distributed algorithm to address the self-monitoring problem for Sensor Webs.

6. Conclusions

This paper investigates the sensor-node monitoring problem for Sensor Webs. We notice that a life-condition monitoring mechanism for sensor nodes is necessary to ensure the function of a Sensor Web as sensor nodes are subject to failures and permanent damage. We consider in-network sensor nodes are self-monitoring, i.e., the status of each sensor node is monitored by another node. To be energy-efficient, a mechanism for the implementation of the self-monitoring of Sensor Web should minimize the total energy consumption required to provide self-monitoring of the Sensor Web. We provide a formulation of this problem specifically. It is shown that this problem can be solved by finding a minimum spanning tree of the graph that models a Sensor Web. We provide some example distributed algorithms to address this problem and conduct simulations to investigate the performance of these algorithms. Finally, a trade-off between the energy cost to find a spanning tree with a distributed algorithm and the energy cost to self-monitoring Sensor Web is identified.

References

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on wireless sensor networks," *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102–114, 2002.

[2] F. L. Lewis, "Wireless sensor networks," in *Smart Environments: Technologies, Protocols, and Applications*, D. J. Cook and S. K. Das, Eds. New York: John Wiley, 2004.

[3] K. A. Delin, "The sensor web: Distributed sensing for collective action," *Sensors Magazine*, July 2006.

[4] ——, "The sensor web: A distributed, wireless monitoring system," *Sensors Magazine*, April 2004.

[5] The Huntington Library, "The Botanical Gardens," http://www.huntington.org/BotanicalDiv/HEHBotanicalHome.html.

[6] SensorWare Systems, Inc., "Sensor webs deployments

- Huntington Botanical Gardens,"
http://www.sensorwaresystems.com/historical/resources/huntington_sw5.shtml.

[7] K. A. Delin, R. Harvey, N. A. Chabot, S. Jackson, M. Adams, D. Johnson, and J. Britton, "Sensor web in antarctica: Developing an intelligent, autonomous platform for locating biological flourishes in cryogenic environments," in *Proc. of the 34th Lunar and Planetary Science Conference*, Houston, TX, March 2003.

[8] K. A. Delin and S. P. Jackson, "Sensor web for in situ exploration of gaseous biosignatures," in *Proc. of the IEEE Aerospace Conference*, vol. 7, March 2000, pp. 465 – 472.

[9] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," in *Proc. of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, November 2000.

[10] D. Wang, Q. Zhang, and J. Liu, "Self-protection for wireless sensor networks," in *Proc. of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, Lisboa, Portugal, July 2006.

[11] T. Rappaport, *Wireless Communications: Principles and Practices (2nd Edition)*. Upper Saddle River: Prentice Hall, 2002.

[12] N. Bulusu, J. Heidemann, and D. Estrin, "GPS-less low-cost outdoor localization for very small devices," *IEEE Personal Communication*, October 2000.

[13] R. C. Prim, "Shortest connection networks and some generalisations," *Bell System Technical Journal*, vol. 36, pp. 1389–1401, 1957.

[14] J. B. Kruskal, "On the shortest spanning subtree and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, pp. 48–50, 1956.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Minimum spanning trees," *Chapter 23, Introduction to Algorithms (2nd Edition)*, pp. 561–579, MIT Press and McGraw-Hill, 2001.

[16] X.-Y. Li, Y. Wang, P.-J. Wan, W.-Z. Song, and O. Frieder., "Localized low-weight graph and its applications in wireless ad hoc networks," in *Proc. of the 2004 IEEE Infocom*, Hong Kong, March 2004.

Yangfan Zhou *(S'04) is currently a PhD student in the Computer Science and Engineering Department at the Chinese University of Hong Kong. He received the B.Sc. (2000) in electronics from Peking University in 2000 and the M.Phil (2005) in computer science and engineering from the Chinese University of Hong Kong. He also worked in industrial area as hardware engineer and later software engineer in China from 2000 to 2003. His research interests are in wireless ad hoc and sensor networks.*

Michael R. Lyu *(S'84-M'88-SM'97-F'04) received the B.S. (1981) in electrical engineering from National Taiwan University, the M.S. (1985) in computer engineering from University of California, Santa Barbara, and the Ph.D. (1988) in computer science from University of California, Los Angeles. He is a Professor in the Computer Science and Engineering Department of the Chinese University of Hong Kong. He worked at the Jet Propulsion Laboratory, Bellcore, and Bell Labs; and taught at the University of Iowa. His research interests include software reliability engineering, software fault tolerance, distributed systems, image and video processing, multimedia technologies, and mobile networks. He has published over 200 papers in these areas. He has participated in more than 30 industrial projects, and helped to develop many commercial systems and software tools. Professor Lyu was frequently invited as a keynote or tutorial speaker to conferences and workshops in U.S., Europe, and Asia. He initiated the International Symposium on Software Reliability Engineering (ISSRE), and was Program Chair for ISSRE'1996, Program Co-Chair for WWW10 and SRDS'2005, and General Chair for ISSRE'2001 and PRDC'2005. He also received Best Paper Awards in ISSRE'98 and in ISSRE'2003. He is the editor-in-chief for two book volumes: Software Fault Tolerance (Wiley, 1995), and the Handbook of Software Reliability Engineering (IEEE and McGraw-Hill, 1996). He has been an Associate Editor of IEEE Transactions on Reliability, IEEE Transactions on Knowledge and Data Engineering, and Journal of Information Science and Engineering. Professor Lyu is an IEEE Fellow and an AAAS Fellow.*