Electronic Preprint for Journal of Information Processing Vol.20 No.1

Regular Paper

Autonomous Multi-Source Multi-Sink Routing in Wireless Sensor Networks

XingPing He[1] Sayaka Kamei[2] Satoshi Fujita[2,a)]

Received: April 1, 2011, Accepted: September 12, 2011

Abstract: This paper proposes a distributed algorithm to calculate a subnetwork of a given wireless sensor network (WSN) connecting a set of sources and a set of sinks, in such a way that: 1) the length of one of the shortest paths connecting from a source to a sink in the subgraph does not exceed the distance from the source to the farthest sink in the original graph, and 2) the number of links contained in the subgraph is smallest. The proposed algorithm tries to improve an initial solution generated by a heuristic scheme by repeatedly applying a local search. The result of simulations indicates that: 1) using a heuristic to generate an initial solution, the size of the initial solution is reduced by 10% compared with a simple shortest path tree; and 2) the local search reduces the size of the resultant subgraph by 20% and the cost required for such an improvement by the local search can be recovered by utilizing the resultant subgraph for a sufficiently long time such as a few days.

Keywords: wireless sensor network, autonomous routing, local search

1. Introduction

According to recent advancements of microelectronics and communication technologies, **wireless sensor networks** (WSNs) have attracted considerable attention in the fields of network computing and distributed processing [2], [9], [16]. The primary task of a WSN is to continuously monitor the surrounding environment (e.g., the temperature and the density of NO_x in the atmosphere) in order to notify the change of the status to an appropriate data aggregation point such as a meteorological observator, in either an event driven or a query-based fashion.

A typical WSN is composed of a large number of tiny devices called **sensor nodes** (or nodes), each of which is capable of conducting a simple arithmetic operation, communicating wirelessly with nearby nodes, and sensing the status of the surrounding environment. In the *multi-hop* version of WSNs [13], [15], which is the target of the current paper, each (sensor) node plays the role of a message router in addition to the role of a sensing device. Note that in such systems, all nodes should collaborate with each other, in order to report their status to an aggregation point in an efficient and timely manner. A lot of pervasive applications proposed in the literature are based on an automatic deployment of sensor nodes providing a continuous and/or periodic snapshot of an environment, such as habitat monitoring [3], target tracking [18], aquatic observations [6], and surveillance [19].

In this paper, we consider a situation in which specific nodes in a WSN consecutively detect events, and notify the fact of event detection to several data aggregation points via a notification message; i.e., several sources consecutively send messages to several sinks. Since message transmission consumes significant electric power in typical WSNs (e.g., 100 mW) compared with normal sensing and/or arithmetic operations (e.g., 1 mW), in order to extend the lifetime of the overall WSN as long as possible, we need to merge as large a number of message delivery routes as possible. Note that this is completely different from the case with a simple aggregation point [10], nor the case with a single source node [5]. In the literature, a number of routing schemes have been proposed for WSNs to enhance the efficiency and the energy-awareness of message delivery. For example, Intanagonwiwat et al. proposed a scalable and robust communication paradigm for WSNs called Directed Diffusion [9], and many researchers try to improve the quality of the delivery paths by introducing a braided multipath scheme [7], explicit consideration of life-time [17], and the robustness against unreliable nodes and fallible wireless links [20]. The notion of potential field used in the Directed Diffusion was later extended to take into account the traffic over the network [1], and Liu et al. proposed a multi-source multi-sink anycast routing framework for WSNs based on the notions of distributed, scalable estimation of the potential field and a probabilistic message forwarding [12].

In this paper, we propose a distributed algorithm to calculate a subgraph of a given WSN such that: 1) the length of one of the shortest paths connecting from a source to a sink in the subgraph does not exceed the distance from the source to the farthest sink in the original graph, and 2) the number of links contained in the subgraph is as small as possible (justification of the above constraint will be described later). Note that we can not find an optimal solution to this problem in polynomial time, since it is equivalent to the metric Steiner tree problem [8] as a special case. The proposed algorithm tries to improve an initial solution gen-

[1] Toshiba Tec Corporation, Shinagawa, Tokyo 141–8664, Japan
[2] Graduate School of Engineering, Hiroshima University, Higashi-hiroshima, Hiroshima 739–8527, Japan
[a)] fujita@se.hiroshima-u.ac.jp

erated by a heuristic message delivery scheme by repeatedly applying a local search. A technique to reduce the cost of the local search is also given in the current paper. The performance of the proposed scheme is experimentally evaluated by simulations. The result of the simulations is summarized as follows: 1) using a heuristic to generate an initial solution, the size of the initial solution is reduced by 10% compared with a simple shortest path tree [11]; and 2) a local search reduces the size of the resultant subgraph by 20% and the cost required for such an improvement can be recovered by utilizing the subgraph 600 times (e.g., if we assume that event is detected for every minute, such a time duration corresponds to 10 hours).

The remainder of this paper is organized as follows. Section 2 describes a model of WSN and the problem to be studied. Sections 3 and 4 describe our proposed scheme. The result of simulations is given in Section 5. Finally, Section 6 concludes the paper with future work.

2. Preliminaries

2.1 Model

Figure 1 illustrates an overhead view of a WSN, and its corresponding graph representation. As shown in Fig. 1 (a), we assume that each node is located at a point in a two-dimensional coordinate space, and by considering circles centered at the location of the nodes with a radius which is a half of the transmission radius of each sensor node, we can naturally define a graph structure such that: 1) each node corresponds to a circle, and 2) two nodes are connected by a link if and only if the corresponding circles have a non-empty intersection. See Fig. 1 (b) for illustration. In the following, we use symbol V to denote a set of sensor nodes, and $N[u]$ ($\subseteq V$) to represent a set of **neighbors** of u. Node u is capable of broadcasting a message to all nodes in $N[u]$ in a single step, where we assume that a collision of messages transmitted by nearby nodes is resolved by an appropriate link layer protocol. The graph defined by V and $\{(u, v) \in V \times V : v \in N[u]\}$ is assumed to be connected. Note that by definition, the binary relation defined by $N[\cdot]$ is symmetric and reflexive, i.e., 1) $u \in N[v]$ implies $v \in N[u]$ for any $u, v \in V$ and 2) $u \in N[u]$ for any u.

Given nodes u and v, a message delivery from u to v is realized by repeating message transmissions along a path connecting those two nodes [*1]. Two end nodes of such a path are called the originator and the destination of the message, respectively [*2], and the number of transmissions conducted before v receiving a message originated from u is called the **hop count** of the message delivery. The minimum hop count from u to v is denoted as $hop(u, v)$. In this paper, we assume that a message transmission consumes a unit of electric power. Thus, the power consumption due to message delivery is proportional to the hop count of the delivery path.

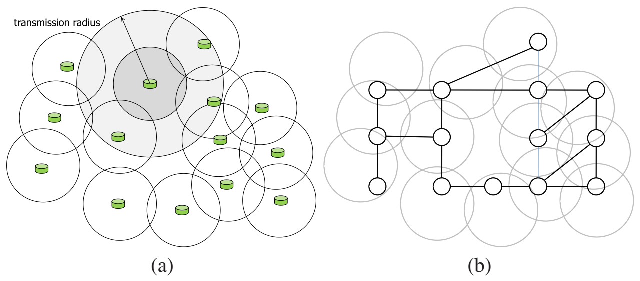


Fig. 1 Model of a WSN and the corresponding graph representation.

2.2 Problem

Suppose that we are given a set of sources S ($\subseteq V$) and a set of sinks D ($\subseteq V$). Each source models a sensor node who detects an event happened in the environment, and each sink models a data aggregation point which is used to extract the detected information from the outside of the network. In the following, for simplicity, we assume that each source in S must notify the detection of an event to "all" nodes in D, while we will make no assumptions on the timing of event detections observed by the source nodes.

In this paper, we consider the problem of minimizing power consumptions of message delivery from S to D. In general, the power consumption level is affected by the timing of event detections in addition to the locations of sets S and D. If $|S| = |D| = 1$, we can minimize the amount of power consumptions by simply forwarding a message along one of the shortest paths connecting to the (unique) sink, which is calculated in polynomial time even in a distributed environment [4]. If the number of destinations becomes greater than one, we can reduce the number of message transmissions by appropriately *merging* delivery paths connecting to different sinks; i.e., by minimizing the number of edges contained in the delivery tree. Such a minimization problem is NP-hard for general D since it is equivalent to the Steiner tree problem [8] which is well known to be NP-hard (as for an approximate calculation, we claim that the minimization problem is 2-approximable since the Steiner tree problem is known to be 2-approximable [14]). For constant number of sinks, however, we can find the optimal solution in polynomial time through an exhaustive search.

If $|S| > 2$, on the other hand, a merge of delivery paths originating from different sources does not guarantee a reduction in power consumption in general. In fact, even if the total number of links is minimized by merging two delivery paths, if such a minimization doubles the length of a delivery path from $s_1 \in S$ to a sink (i.e., if it increases the "stretch" of the delivery path to two), the power consumption of the overall network will become almost *twice* when 99% of the events are detected by source s_1. This indicates that, from a practical point of view, we should merge delivery paths originating from different sources only when it satisfies a constraint concerned with the "stretch" of each delivery path (or we should give up with reducing the total number of message transmissions).

In order to clarify this point, in this paper, we will focus on the following constraint on the maximum stretch of the resultant

[*1] Although each node has the capability to broadcast a message to its neighbors, it can simulate unicast by designating the name of the receiver in each message.

[*2] In the following, we distinguish the sender and receiver of a message as the originator and destination of the message.

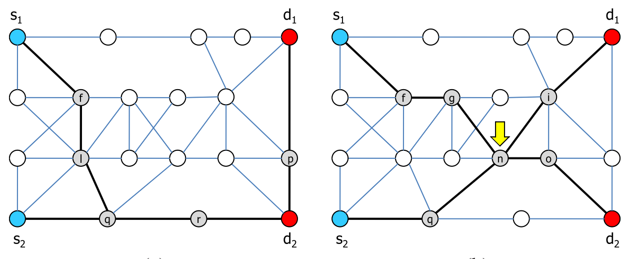


Fig. 2 Solutions with (a) eight links, and (b) nine links.

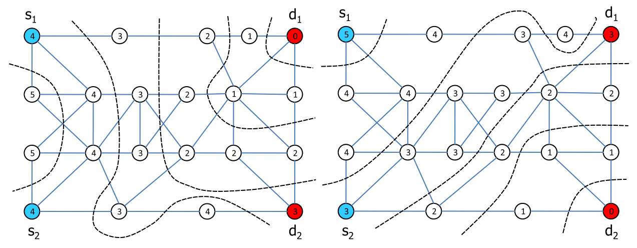


Fig. 3 Distance field for sinks d_1 and d_2.

message delivery paths: *For each $s_i \in S$, let h_i be the (minimum) hop count to the furthest sink in D. A merge of delivery paths is allowed only when the length of the resultant path does not exceed h_i for any sink $d_j \in D$.* By this constraint, the power consumption (i.e., the total number of message transmissions) may increase compared with cases without such a constraint. However, it enables us to bound the maximum number of links by $h_i \times |D|$ even if only a single source s_i detects an event. In addition, it is meaningful to bound the latency of event notifications to each sink in D; i.e., it guarantees that every sink receives a notification from s_i within h_i hops after an event detection.

To see this in more detail, let us consider an example shown in **Fig. 2**. Two solutions are illustrated in the figure. The first solution shown in Fig. 2 (a) consists of eight links, and the second solution shown in Fig. 2 (b) consists of nine links. Although the first solution uses less number of links than the second one, it does not satisfy the above constraint since the hop count from s_1 to d_1 is seven, which is larger than the length of the shortest path connecting s_1 and d_2.

Finally, in this paper, we do not consider the merge of several messages transmitted by different nodes at some rendezvous point, although the overall power consumption will further be reduced by such a technique.

3. Basic Procedure for Single Source

Before describing the details of the proposed scheme, we first provide a procedure to calculate a message delivery tree for the case of $|S| = 1$. This procedure is used as a basic module in the proposed scheme for the case of several sources, which will be described in Section 4.

3.1 Overview

Let $D = \{d_1, d_2, \ldots, d_{|D|}\}$ be a given set of sinks. The procedure tries to merge shortest paths which are independently established from the (unique) source to each sink by "delaying" the point of splitting (note that independent shortest paths split at the source). Such a merge process proceeds in the following two steps:

Step 1 (Preprocessing): Each sink d_j broadcasts a message to all nodes in the network the hop count to the sink. Let $d(u, v)$ be a variable representing the (minimum) distance from u to v in terms of hop count. Variable $d(u, v)$ is locally kept by node u, i.e., u keeps a list of variables $\langle d(u, d_1), d(u, d_2), \ldots, d(u, d_{|D|})\rangle$ in its local memory, which serves as a routing table. **Figure 3** illustrates the calculated value of $d(u, d_i)$ for each i. The concrete way of calculating those values is described in Section 3.2.

Step 2 (Event notification): Upon detecting an event, the source transmits a message to notify the detection of an event to all sinks in D. This message is copied at each intermediate node on the delivery path, and is forwarded to the next node on the path towards the direction of the sinks. Each copy is attached a set of sinks relevant with it. Initially, the source has a copy of the message associated with set D. Suppose that node u receives a message with sink set $D' \subseteq D$ from its neighbor. u calculates the next node for each sink in D' by referring to the $d(*, *)$ values held by its neighboring nodes (concrete procedure for the calculation is described in Section 3.3). Let N' ($\subseteq N[u]$) be the set of adjacent nodes calculated by the procedure, where each node $w \in N'$ is associated with a set of sinks D_w ($\subseteq D'$). Finally, for each $w \in N'$, u forwards a copy of the received message to w by attaching D_w as the set of corresponding sinks.

3.2 First Step

This subsection describes the details of the first step. In the first step, each sink broadcasts a message to all nodes in V to notify the hop count from those nodes to the sink. Each message broadcast by sink $d_j \in D$ has a field $\mathsf{mhc}(d_j)$ representing the hop count to d_j, which is initialized to zero when it is transmitted by the sink. The behavior of intermediate node u is as follows: 1) It initializes variable $d(u, d_j)$ to ∞ for all $d_j \in D$; 2) After receiving a message with $\mathsf{mhc}(d_j) = i$ from a neighbor, it compares value i with $d(u, d_j)$, and executes the following only when $i < d(u, d_j)$: 2a) $d(u, d_j) := i + 1$, and 2b) transmits a copy of the message after incrementing $\mathsf{mhc}(d_j)$ by one.

Note that the above scheme is a naive (distributed) implementation of Dijkstra's algorithm. The final result is the calculation of the shortest path from d_j to all nodes in V.

3.3 Second Step

As a concrete procedure for the second step, we propose two different strategies called SIMPLE and SLACK. SIMPLE constructs a minimum shortest path tree (MSPT) from a given source to all sinks in D, and SLACK applies a heuristic optimization a simple MSPT to reduce the number of links contained in the delivery tree.

3.3.1 Strategy SIMPLE

Figure 4 illustrates an MSPT rooted at source s_2. In SIMPLE, each node u conducts the following procedure to construct an MSPT in a distributed manner:

(1) Suppose that u receives a message with a set of sinks D' from its neighbor. For each $d_j \in D'$, let $N_j (\subset N[u])$ be the set of neighbors w of u such that

$d(w, d_j) = d(u, d_j) - 1$.

Note that $d(w, d_j) < d(u, d_j)$ implies $d(w, d_j) = d(u, d_j) - 1$ since $d(w, d_j) \geq d(w, d_j) - 1$ by construction. Let $N' = \bigcup_{d_j \in D'} N_j$ be the set of neighbors relevant with D'.

(2) For each $w \in N'$, let $c(w) \overset{\text{def}}{=} |\{d_j \in D' : w \in N_j\}|$.

(3) After initializing X to $\emptyset$, repeat the following until $N' = \emptyset$: 1) let w be an element in N' with the largest $c(w)$; 2) add w to X with relevant sinks $D_w = \{d_j \in D' : w \in N_j\}$; 3) $N' := N' - \{w\}$ and $D' := D' - D_w$; and 4) recalculate function c for the updated N' and D'.

(4) Node u transmits copies of the received message to each neighbor w in X with set D_w of relevant sinks.

Note that under this strategy, each intermediate node that receives a message from its neighbor will partition the set of corresponding destinations into subsets in a greedy manner. Thus, it is guaranteed that every sink is connected with the source through a shortest path in the resultant delivery tree.

3.3.2 Strategy SLACK

In contrast to SIMPLE, the second strategy SLACK tries to reduce the number of links by merging as large a number of delivery paths as possible, while keeping the hop count to each sink from the source to being no larger than the hop count to the furthest sink from the source (thus, each sink may not be connected with the source through a shortest path). In order to realize such a behavior in a distributed manner, for each sink d_j, we assign a *slack* to d_j at the source s in such a way that the summation of $d(s, d_j)$ and the slack is equal to the hop count to the furthest sink. The slack is propagated to nodes in the network with the message delivered to sinks, where the slack of a sink decreases when it passes a link which does not reduce the hop count to the sink. More concretely,

(1) it decrements by one if it passes a link connecting to a neighbor with the same hop count to the sink, and

(2) it decrements by two if the selection of the link increases the hop count to the sink by one.

Recall that the definition of hop count ensures that a single message transmission increases the hop count by two or more.

In the second strategy, a delivery path towards sink d_j will be merged with other paths as long as the slack of d_j does not become negative after the merge, even if the resulting path to d_j is not the shortest. Such a (conditional) merge is realized by simply following the decision made by a *critical* path connecting to a sink with zero slack at the time of partitioning D' using SIMPLE; i.e., by simply being a member of the same subset with a critical sink even if the selected neighbor is not contained in subset N_j (if there are several critical sinks, it may select a critical sink such that the reduction of the slack becomes smallest).

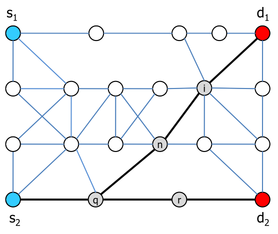


Fig. 4 An MSPT starting from source s_2.

Note that in both strategies, the number of transmitted messages is at most $\sum_{s \in S} \sum_{d \in D} d(s, d)$, and the amount of reduction from this upper bound is dependent on the instance.

4. Local Search

4.1 Proxy-Controlled Merge Scheme

Now let us consider the case of several sources. Recall that the goal of our scheme is to find a subgraph of G which connects S and D with as small a number of communication links as possible, without violating the constraint on the stretch of each delivery path. In order to attain such a goal in a distributed manner, we associate a **token** t_i to each sink $d_i \in D$ designating the merge point to the sink, and try to move it in the direction of the sources, so as to control the merge of delivery paths coming from different sources.

More concretely, in our scheme, a subgraph connecting S and D is a combination of the following two parts:

(1) In the first part, each source independently establishes a delivery tree with nodes containing tokens as the leaves by using SIMPLE or SLACK.

(2) In the second part, leaves of the trees are connected with the corresponding sinks by a collection of shortest paths connecting to those sinks; i.e., subpaths connecting a leaf to a sink are merged into a single path.

An outline of the procedure is described as follows (**Figure 5** illustrates using snapshots of the merge steps). Let x_i be a variable representing a node having token t_i, and X be the set of nodes represented by variables $x_1, x_2, \ldots, x_{|D|}$. Nodes in X are called "proxies." Merge processes for different sinks are conducted sequentially, where the synchronization among proxies is realized by circulating X. Note that X is a multiset initialized to D, and the content of the multiset is changed according to the progress of the merge process. In Fig. 5, token t_1 is initially placed at node d_1, and is handed over to nodes i and n in a sequential manner. Given

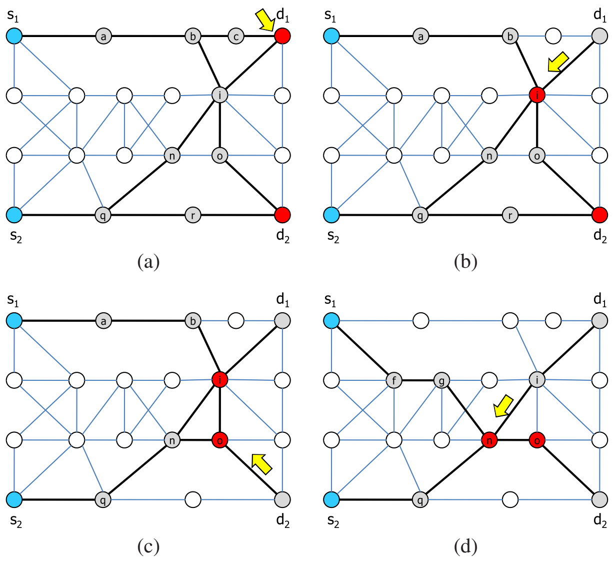


Fig. 5 Snapshots of the merge process; (a) initial configuration; (b) after the first move; (c) after the second move; and (d) after the third move.

X, each source independently establishes a tree towards nodes in X, using a procedure described in the last subsection; i.e., when a node becomes a new proxy, it conducts the first step of the procedure to notify the distance from each node to the proxy. Let $\Psi(X)$ be the summation of the number of links in such trees and the length of the shortest paths connecting proxies and their corresponding sinks (in what follows, we call $\Psi(X)$ the "cost" of X).

After establishing such trees, proxies conduct a *local search* to find a better X' such that $\Psi(X') < \Psi(X)$, and update X to X' if such X' exists. Such a local search is repeated until no further improvement can be obtained. More concretely, improvement of X is sequentially attempted by proxies in a cyclic order which is given to sinks in D beforehand. The search process stops if no proxy in X can find an appropriate candidate in its neighborhood. In Fig. 5, the cost of initial configuration $X = \{d_1, d_2\}$ is 13, and is improved to 11 by moving token t_1 from d_1 to node i (Fig. 5 (b)); improved to 10 by moving token t_2 from d_2 to node o (Fig. 5 (c)); and improved to 9 by moving token t_1 from node i to node n (Fig. 5 (d)).

In the subsequent subsections, we describe a detailed way to realize such a local search; i.e., how to find a successor of a given proxy which reduces the cost of the resultant solution, and how to hand over the role of proxy to the selected candidate.

4.2 Delivery Tree with Two Sinks

This subsection further clarifies our discussion by assuming that the number of destinations is restricted to two (i.e., $|D| = 2$) and each delivery tree is established using the SIMPLE strategy. A generalization for the other cases is described in Section 4.3.

4.2.1 Overview

Let $X = \{x_1, x_2\}$ be a set of proxies, and let us consider a situation in which proxy x_1 tries to find its successor x'_1 from $N[x_1]$. Recall that S denotes the set of sources. Suppose that the paths from source s_i to proxies x_1 and x_2 are split at node r_i. Then, the value of function $\Psi(X)$ can be represented as follows:

$$\Psi(X) \; = \; \sum_{s_i \in S} hop(s_i, x_2) + \sum_{s_i \in S} hop(r_i, x_1) + \alpha, \qquad (1)$$

where $\alpha = hop(x_1, d_1) + hop(x_2, d_2)$. Since we are assuming that S and x_2 are fixed, the amount of change of $\Psi(X)$ according to the change of x_1 to its neighbor v, is equivalent to the change of the second term in the above equation, where we have to notice that the branch node may differ for x_1 and v. In other words, proxy x_1 can check the goodness of its neighbor v as a new proxy, by evaluating *the hop count from v to a branch node on the shortest path connecting s_j and x_2 for every source s_j*, and by taking the summation of them (we should, of course, take into account the distance to the corresponding sink from x_1 and v, respectively, which is easily checked by referring to variables $d(x_1, d_1)$ and $d(v, d_1)$).

Let $r(v)$ denote the branch node corresponding to node $v \in N[x_1]$. In order to evaluate the distance to the shortest path, in the proposed scheme, we use two types of messages called REQ and Signal, in the following manner [*3]:

[*3] We assume that each message is attached to the name of the originator, the name of the sender, and the hop count from the originator to the receiver of the message.

- Message REQ is disseminated by node v to notify the hop count to the node. Note that such messages have already been used in the procedure given in Section 3.2. In the following, we assume that the result of the notifications is locally stored by each node u in the form of array hcv, such that hcv$[v] = hop(u, v)$.
- Message Signal(next,Dest,count) is transmitted by a source which detects an event, where Dest ($\subseteq X$) is a (sub)set of proxies corresponding to the message, next is the designated receiver of the message, and count is a field to enumerate the hop count from the last branch node.

4.2.2 A Counting Scheme

With the above notations, a procedure to calculate $hop(r(x_1), x_1)$ is now described as follows. In the following, we assume that each node in V has already calculated hcv$[x]$ for each $x \in S \cup X$, and stores it in its local memory (an efficient way to disseminate REQ is described in the next subsection). Upon detecting an event, a source transmits Signal(next,Dest,0) to its neighbors by appropriately setting the next and Dest fields. After receiving the message Signal(u, X', count) from its neighbor, node u, which is the designated receiver of this message, conducts the following operation (if it is not the designated receiver, it simply discards the received message):

Step 1: Let $P := N[u]$ and $Q := X'$; i.e., P is initialized to the set of neighbors of u, and Q is initialized to the set of proxies associated with the received message. The following two steps are repeated until $Q = \emptyset$.

Step 2: Let $Q_x \overset{\text{def}}{=} \{y \in Q\colon hop(x, y) < hop(u, y)\}$ for $x \in P$, and let $w \in P$ be a node with the largest Q_w. Note that Q_x is a subset of proxies such that x is on a shortest path connecting from u to the proxy. Node u transmits Signal(w, Q_w, c) to node w, where $c = 0$ if $|Q_w| > 1$ and $c =$ count$+1$ otherwise.

Step 3: Update local variables as $P := P - \{v\}$ and $Q := Q - Q_w$.

Note that in the above procedure, count enumerates the number of hops from the last branch $r(x_1)$ to the proxy x_1; i.e., it starts to increment the counter when the number of proxies corresponding to the message becomes one. Thus, by taking the summation of such counter values over all sources, proxy x_1 can obtain a value corresponding to the second term in Eq. (1). A similar observation holds for any node in $N[x_1]$. Thus, by sequentially conducting the above operation for all nodes in $N[x_1]$, and by comparing the resultant values to the value for x_1, we can select the best candidate as the successor to proxy x_1.

The reader should note that the above local search scheme is scalable since it merely tries to find a neighboring node which reduces the total cost of the resultant trees and the selection of a neighbor is not affected by the nodes which do not exist around the current delivery trees.

4.2.3 Efficient Propagation of REQ

This subsection describes several heuristics to reduce the number of message transmissions in the proposed local search scheme. Recall that the objective of flooding REQ from node v is to notify the hop count from each node to node v. However, such information is necessary only for the nodes around a delivery tree connecting S and X, since the other nodes have no chance to use such information. In other words, we can significantly reduce the propagation cost for REQ by not forwarding to such non-relevant nodes.

In this paper, we propose the following three heuristics to attain such reductions:

(1) The first idea is to control the range of message delivery by introducing TTL (Time-To-Live). Message REQ issued by node v is useful only to nodes at distance $\max_{x \in S} hop(x, v)$ or less from v, since the other nodes never become a member of the resulting delivery trees. Such a behavior can easily be realized by providing a TTL to messages transmitted by v.

(2) Suppose that node u receives REQ issued by a neighbor v of the current proxy x_i. Our second idea is to allow u to stop the propagation of the message if it observes $hop(u, x_i) = hop(u, v)$. In fact, since the above equality implies that the same property holds at any node u' closer to a source than u (i.e., to deliver Signal to v), node u' may simply use $hop(u', x_1)$ instead of $hop(u', v)$.

(3) The last heuristic is applicable only to the first strategy SIMPLE. Suppose that u receives REQ from neighbor u'. If a part of vector hcv concerned with nodes in S held by u, is greater than or equal to a part of the vector held by u' (in terms of element-wise comparison), u can stop the forwarding of the message, since it implies that u does not exist on any shortest path originating from the source node.

4.3 Generalization

If the underlying strategy is SIMPLE, we can directly apply the above idea to the case of $|D| \geq 3$, since in any delivery tree under SIMPLE, each pair of source and proxy is connected by a shortest path between them, and the change of the position of a proxy (to its neighbor) does not affect to the configuration of the remaining subtree. Thus, for any D, we may simply evaluate the length of the path to the proxy from the last branch, in order to calculate the value corresponding to the second term of Eq. (1).

On the other hand, if the underlying strategy is SLACK, we cannot evaluate the reduction of the second term without recalculating the cost for the whole network, since in general, the move of a proxy affects to the delivery paths to the other proxies. Thus, if we wish to estimate the amount of reduction very accurately, one step of the local search under SLACK becomes much more expensive than SIMPLE, although it would significantly reduce the size of the resulting subgraph. In the simulation described in the next section, we use a variant of such local search, such that the hop count from the last branch is used as an *approximated* cost. As will be shown later, such a heuristic approach is still effective to improve the quality of the solution generated by SLACK (in fact, SLACK with a modified local search provides the best solution among schemes examined in the experiments).

5. Simulation

In this section, we evaluate the performance of the proposed schemes by simulation. In the simulation, we compare the performance of four schemes (i.e., either SIMPLE or SLACK, and either with or without local search) against the performance of a naive scheme in which each source independently establishes the shortest path to each sink (in what follows, we call such a naive scheme NAIVE). In the following, "+" indicates that the scheme refines the initial solution by applying local search (e.g., SIMPLE+ means a scheme which generates an initial solution by SIMPLE, and then improves it by repeatedly applying a local search).

5.1 Instances

Instances used in the simulation are generated as follows. Given a two-dimensional space of size $n \times m$, we select 3000 points at random with a uniform probability, and associate them to the set of nodes V. We then connect each pair of nodes by a link if and only if the Euclidean distance between them is smaller than or equal to 1.5 (i.e., we assume that the transmission radius of each node is 1.5). As for determining the location of S and D, as well as the size of the given two-dimensional space, we consider the five patterns listed in **Table 1**, and generate 100 random instances for each pattern (i.e., results described below are averaged over those 100 instances). Note that the above five patterns can be classified into two groups; i.e., in the first group (i.e., $P1$, $P2$, and $P3$), the distance among sources and the distance among sinks are both fixed to 20, and in the second group (i.e., $P1$, $P4$, and $P5$), the distance between S and D is fixed to 40. In all patterns, the average degree of each node is 13.2, and the number of hops between a source and a sink is almost proportional to the Euclidean distance between them. For example, if the Euclidean distance is 40, then 60 hops is needed in typical examples.

5.2 Size of Delivery Trees

A comparison of the schemes with respect to the size of the resultant subgraph is shown in **Table 2**. For each pattern, SLACK+ generates the smallest subgraph. In particular, it improves the solution of NAIVE by 44% for pattern $P3$, which is apparently due to the effect of merge process starting from nodes in $S \cup D$. In addition, we can observe that SLACK beats SIMPLE for every pattern, which is because SLACK tries to reduce the number of branches generated at the intermediate nodes compared with SIMPLE.

The impact of the location of S and D to the size of the resultant subgraph is summarized as follows: In the first group, as the distance between S and D increases, the number of reductions in size (compared with NAIVE) increases, since a long path connecting S and D has a bigger chance of being merged with

Table 1 Five patterns of examined instances.

pattern	size	D	S
$P1$	40×40	$(10,0),(30,0)$	$(10,40),(30,40)$
$P2$	25×64	$(2,0),(22,0)$	$(2,64),(22,64)$
$P3$	20×80	$(0,0),(20,0)$	$(0,80),(20,80)$
$P4$	40×40	$(5,0),(35,0)$	$(5,40),(35,40)$
$P5$	40×40	$(0,0),(40,0)$	$(0,40),(40,40)$

Table 2 Size of the resultant delivery trees.

	P1	P2	P3	P4	P5
NAIVE	141	217	273	150	161
SIMPLE	125	183	204	138	151
SIMPLE+	114	164	183	131	141
SLACK	110	169	194	125	139
SLACK+	101	144	154	121	131

other paths than short paths. On the other hand, the result on the second group indicates that a similar phenomenon can be observed when the distance among nodes in S and D decrease. In fact, by decreasing the distance among sources, the location of the split point becomes closer to the sources, which implies that a large portion of the delivery path is merged with other paths. The amount of such reduction increases by adopting SLACK instead of SIMPLE (compare the second and the fourth rows of the table), and of course, by applying local search.

5.3 Total Cost including Local Search

In the proposed scheme, a reduction in the size of the solution is attained at the cost of a local search. Thus, we next evaluate the "total cost" of the schemes which is defined as the summation of the size of the resultant subgraph and the cost of the local search.

In the following, without loss of generality, we assume that each source detects an event several times (otherwise, it is not necessary to reduce the size of the initial network by applying local search). The total cost of a scheme is defined as the summation of the cost for propagating Signal messages and the cost for propagating REQ messages. In general, a transmission of Signal is more expensive than a transmission of REQ since Signal should contain additional information such as the set of destinations. In the following, we assume that the cost for a Signal message is α (> 1) times larger than the cost for a REQ message.

During the calculation of an initial solution, each node v in $S \cup D$ broadcasts a REQ message to all nodes in V exactly once to calculate hcv[v]. Let c_f denote the cost for broadcasting REQ. On the other hand, during a local search, each candidate of a successor of the proxy must send REQ to a limited region of the network (see Section 4.2 for the details). Let c_e be the cost for a covering such a limited region with REQ messages. Thus, the total cost for transmitting REQ is $c_e + c_f$. Suppose that a repetitive application of local search monotonically decreases the size of the resultant subgraph from l_1 to l_2, and it transmits w_1 Signal messages before obtaining the final solution, and w_2 Signal messages after obtaining the final solution. Then, the total cost for transmitting Signal is at most $(l_1 \times w_1 + l_2 \times w_2) \times \alpha$.

By combining the above values, an upper bound on the total cost of a scheme is described as follows:

$$c_f + c_e + (l_1 \times w_1 + l_2 \times w_2) \times \alpha. \qquad (2)$$

This value linearly increases as w_2 increases. Thus, if a scheme generates a smaller subgraph than another one, two lines representing the total cost of those schemes should *cross* at a specific value for w_2, even if the first scheme requires a higher cost than the second one to generate the final solution.

Table 3 summarizes some values for w_2 as determined by SLACK+ and the other three schemes, where the value of parameter α is fixed to 50, and the values of other parameters are determined as shown in **Table 4** (those values are acquired through simulations, and in the experiments, we confirmed that the technique shown in Section 4.2 reduces the cost c_e for the SLACK+ scheme). The first row in the table indicates that for every pattern, the total cost of SLACK+ becomes lower than the cost of NAIVE if $w_2 > 171$, and similarly, we can observe that SLACK+ beats SIMPLE if $w_2 > 393$, and SLACK+ beats SIMPLE+ if $w_2 > 699$. The above results indicate that although additional cost is needed to calculate an improved solution, the local search used in the proposed scheme ultimately reduces the total cost of the overall scheme if the lifetime of the given WSN is sufficiently long; e.g., if each source reports its status to sinks every minute, the above values correspond to about 10 hours.

Table 3 Value for w_2.

	P1	P2	P3	P4	P5
NAIVE	120	163	168	128	171
SIMPLE	211	308	393	216	268
SLACK	473	481	485	699	561

Table 4 Total cost of SLACK+.

	$P1$	$P2$	$P3$	$P4$	$P5$
c_f	12000	12000	12000	12000	12000
c_e	234232	581931	991364	193938	270747
l_1	141	218	272	148	160
l_2	101	145	155	120	132
w_1	204	357	649	131	149

6. Concluding Remarks

This paper proposed a distributed algorithm to calculate a subgraph of a given WSN connecting a set of sources and a set of sinks with as small number of links as possible. The proposed algorithm tries to improve an initial solution generated by a heuristic scheme by repeatedly applying a local search.

Future works include:

- We need to reduce the cost required for a local search by performing the search in parallel.
- We have to implement the proposed scheme in an actual WSN (we are now constructing an infrastructure consisting of several SunSPOT devices).

Reference

[1] Basu, A., Lin, A. and Ramanathan, S.: Routing Using Potentials: A Dynamic Traffic-Aware Routing Algorithm, *Proc. 2003 ACM SIGCOMM*, pp.37–48 (2003).

[2] Bonnet, P., Gehrke, J.E. and Seshadri, P.: Querying the Physical World, *IEEE Personal Communications*, Vol.7, No.5, pp.10–15 (2000).

[3] Cerpa, A., Elson, J., Estrin, D., Girod, L., Hamilton, M. and Zhao, J.: Habitat Monitoring: Application Driver for Wireless Communications Technology, *Proc. 2001 ACM SIGCOMM Workshop Data Communication in Latin America and the Caribbean*, pp.20–41 (2001).

[4] Chen, C.C.: A Distributed Algorithm for Shortest Paths, *IEEE Trans. Comput.*, Vol.31, No.9, pp.898–899 (1982).

[5] Chuang, S., Chen, C. and Jiang, C.: Minimum-delay energy-efficient source to multisink routing in wireless sensor networks, *Signal Processing*, Vol.87, No.12, pp.2934–2948 (2007).

[6] Dhariwal, A., Zhang, B., Stauffer, B., Oberg, C., Sukhatme, G.S., Caron, D.A. and Requicha, A.A.: Networked Aquatic Microbial Observing System, *Proc. IEEE International Conference on Robotics and Automation*, pp.4285–4287 (2006).

[7] Ganesan, D., Govindan, R., Shenker, S. and Estrin, D.: Highly-Resilient Energy-Efficient Multipath Routing in Wireless Sensor Networks, *ACM Mobile Computing and Communication Review*, Vol.5, No.4, pp.11–25 (2001).

[8] Garey, M.R. and Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness.*, W.H. Freeman and Company, San Francisco (1979).

[9] Intanagonwiwat, C., Govindan, R. and Estrin, D.: Directed Diffusion: A scalable and robust communication paradigm for sensor networks, *Proc. 6th Annual International Conference on Mobile Computing and Networking (MobiCOM '00)*, pp.56–67 (2000).

[10] Intanagonwiwat, C., Estrin, D., Govindan, R. and Heidemann, J.: Impact of network Density On Data Aggregation in Wireless sensor Networks, *Proc. 22th ICDCS'02*, p.457 (2002).

[11] Krishnamachari, B., Estrin, D. and Wicker, S.: Modeling data-centric routing in wireless sensor networks, *Proc. IEEE INFCOM* (2002).

[12] Liu, H., Zhang, Z.-L., Srivastava, J. and Firoiu, V.: PWave: A Multi-source Multi-sink Anycast Routing Framework for Wireless Sensor Networks, *NETWORKING 2007, Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet, 6th International IFIP-TC6 Networking Conference*, LNCS 4479, pp.179–190 (2007).

[13] Marcy, H.O. and Kaiser, W.J.: Wireless Integrated Network Sensors: Low power systems on a chip, *Proc. 24th European Solid State Circuits Conference* (1998).

[14] Mehlhorn, K.: A faster approximation algorithm for the Steiner problem in graphs, *Inf. Process. Lett.*, Vol.27, No.2, pp.125–128 (1988).

[15] Rahman, R., Alanyali, M. and Saligrama, V.: Distributed tracking in multihop sensor networks with communication delays, *IEEE Trans. Signal Processing*, Vol.55, pp.4656–4668 (2007).

[16] Rosencrantz, M., Gordon, G. and Thrun, S.: Decentralized sensor fusion with distributed particle filters, *Proc. UAI* (2003).

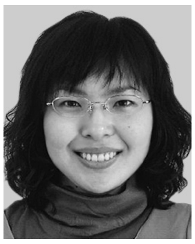
[17] Shah, R.C. and Rabaey, J.: Energy Aware Routing for Low Energy Ad Hoc Sensor Networks, *Proc. 2002 IEEE Wireless Communications and Networking Conference* (*WCNC2002*), Vol.1, pp.350–355 (2002).

[18] Shin, J., Guibas, L. and Zhao, F.: A Distributed Algorithm for Managing Multi-target Identities in Wireless Ad-hoc Sensor Networks, *Proc. IPSN 2003*, LNCS 2634, pp.223–238 (2003).

[19] Werner-Allen, G., Johnson, J., Ruiz, M., Lees, J. and Welsh, M.: Monitoring volcanic eruptions with a wireless sensor network, *Proc. 2nd EWSN* (2005).

[20] Ye, F., Zhong, G., Lu, S. and Zhang, L.: GRAdient Broadcast: A Robust Data Delivery Protocol for Large Scale Sensor Networks, *J. Wireless Networks*, Vol.11, No.3, pp.285–298 (2005).

XingPing He received his B.E. degree from Hiroshima Shudo University in 2006, and M.E. degree from Hiroshima University in 2009. He has been working in Toshiba TEC corporation since 2009.



Sayaka Kamei received her B.E., M.E. and D.E. degrees in electronics engineering in 2001, 2003, and 2006 respectively from Hiroshima University. She is currently an assistant professor in the Department of Information Engineering, Graduate School of Engineering, Hiroshima University. Her research interests include distributed algorithms. She is a member of the IEEE, ACM, IEICE, and Information Processing Society of Japan.

Satoshi Fujita received his B.E., M.E., and a Ph.D. degrees in Information Engineering from Hiroshima University in 1985, 1987, and 2000, respectively. He has been a professor at Hiroshima University since 2007. His research interests include parallel algorithms, graph algorithms, communication networks, and resource assignment problem in interconnection networks. He is a member of IEEE Computer Society, SIAM, IPS Japan, and IEICE.