Wireless Netw (2007) 13:583–595
DOI 10.1007/s11276-006-6254-6

Dynamic proxy tree-based data dissemination schemes for wireless sensor networks

Wensheng Zhang · Guohong Cao · Tom La Porta

Published online: 8 May 2006

Abstract In wireless sensor networks, efficiently disseminating data from a dynamic source to multiple mobile sinks is important for the applications such as mobile target detection and tracking. The tree-based multicasting scheme can be used. However, because of the short communication range of each sensor node and the frequent movement of sources and sinks, a sink may fail to receive data due to broken paths, and the tree should be frequently reconfigured to reconnect sources and sinks. To address the problem, we propose a *dynamic proxy tree-based framework* in this paper. A big challenge in implementing the framework is how to efficiently reconfigure the *proxy tree* as sources and sinks change. We model the problem as on-line constructing a minimum Steiner tree in an Euclidean plane, and propose centralized schemes to solve it. Considering the strict energy constraints in wireless sensor networks, we further propose two distributed on-line schemes, the *shortest path-based (SP) scheme* and the *spanning range-based (SR) scheme*. Extensive simulations are conducted to evaluate the schemes. The results show that the distributed schemes have similar performance as the centralized ones, and among the distributed schemes, the SR scheme outperforms the SP scheme.

Keywords Data dissemination · Wireless sensor networks

W. Zhang (✉)
Department of Computer Science, Iowa State University, Ames, IA 50011
e-mail: wzhang@cs.iastate.edu

G. Cao · T. L. Porta
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802

1. Introduction

A wireless sensor network [1] consists of many tiny and inexpensive sensor nodes that are distributed over a vast field to obtain sensing data. These nodes are capable of not only measuring real world phenomena, but also storing, processing and transferring these measurements. Due to the above-mentioned attractive characteristics, sensor networks are adopted in many military and civil applications such as battlefield surveillance, environmental control, and security management. In these applications, sensing data often need to be disseminated from a source to multiple sinks, where the source and the sinks may frequently move. For example, a sensor network may be deployed in a battlefield to detect and monitor the enemy tanks and soldiers. When a target of interest is detected, the sensing data about the target should be sent to commanders and soldiers moving in the battlefield.

In recent years, many data dissemination schemes [2–8] have been proposed for sensor networks, but most of them cannot efficiently support multicasting from a dynamic source to multiple mobile sinks. For example, the external storage-based scheme [2], the data-centric storage-based (DCS) scheme [5] and the index-based scheme [8] only consider the point-to-point communication between a pair of source and sink. The directed diffusion scheme [3] and the two-tier data dissemination (TTDD) scheme [4] naturally support data multicasting, but they are not efficient when the source and the sinks are mobile. In the directed diffusion scheme, sinks need to flood their interests over the whole network, and then the sources matching the interests send their data to the sinks via multiple paths. When the network is large and/or the sinks change frequently, the flooding of interests will cause large overhead. The TTDD scheme proactively maintains a grid-based propagation structure over the whole network regardless of the actual locations of the sinks, and

the structure should be updated whenever the source location changes, which may also cause large maintenance overhead.

To avoid unnecessarily flooding information [3] or expanding propagation structure over the whole network [4], we may use a tree-based multicasting scheme. In this scheme, the source and the sinks form a tree rooted at the source, and the source pushes data to the sinks along the tree branches. However, due to the short communication range of each sensor node and the frequent movement of sources and sinks, a sink may frequently fail to receive data due to broken paths, and the tree should be frequently reconfigured to reconnect sources and sinks. To address the problems, we propose a *dynamic proxy tree-based framework*. In this framework, each source (sink) is associated with a stationary sensor node called *source (sink) proxy*. The proxies related to the same source form a *proxy tree*. Facilitated by the tree, a source can push data to its proxy, which further pushes the data to multiple sink proxies, and a sink can query its proxy to get the data.

As a source changes or a sink moves, the associated proxy should be changed to reduce the cost of pushing (querying) data to (from) the proxy. Accordingly, the tree should also be reconfigured to reduce the cost of multicasting data from the source proxy to the sink proxies. Due to the strict energy constraints in sensor networks, tree reconfiguration should be conducted in an energy efficient way. Many multicasting tree reconfiguration schemes have been proposed for the existing wired and wireless networks such as the Internet, the cellular network and the wireless ad hoc network. However, these schemes can not be directly applied to the wireless sensor network due to their large overhead. For example, in the *rearrangeable inexpensive edge-based on-line Steiner (ARIES) algorithm* [9], a new node joins an existing tree via the shortest path, and the subtrees including newly added or deleted nodes are reconfigured every certain time. This algorithm requires each multicasting member to know its distance to other members. It is feasible for the Internet and the cellular network, in which each router (base station) can naturally obtain the information through the underlying topology advertisement protocol (e.g., OSPF). However, it is not applicable in sensor networks, where running the topology advertisement protocol may cause large overhead. On the other hand, the multicasting protocols [10–12] for mobile ad hoc networks emphasize more on route robustness and pay less attention to energy efficiency, because mobile ad hoc networks have frequent path breaks due to high node mobility.

In this paper, we first formalize the tree reconfiguration problem as an on-line Euclidean Steiner problem [13], and present several centralized schemes to solve the problem. Considering the strict energy constraints and the locality requirements in wireless sensor networks, we propose two distributed heuristic-based schemes, the *shortest path-based (SP)* scheme and the *spanning range-based (SR)* scheme. These schemes are motivated by the following observations: First, the new proxy of a source (sink) can utilize the information provided by the previous proxy to efficiently join the tree. Second, localized adjustments can be conducted at individual nodes to gradually optimize the tree structure. In the SP scheme, when a sink (source) changes its proxy, the new proxy uses flooding to discover a parent node and joins the tree; the proxy changes cause the tree nodes to gradually adjust their locations in a localized way. The SP scheme, however, still has large overhead due to flooding, especially when the tree nodes are far away from each other. So we propose the SR scheme, in which each subtree is associated with a certain *spanning range*, which is dynamically assigned and adjusted. Using fewer messages, a new proxy can find the root of the smallest subtree whose spanning range covers itself, and joins the subtree.

We use extensive simulations to compare the proposed schemes in terms of data dissemination cost and tree reconfiguration overhead. The results show that the centralized schemes slightly outperforms the distributed schemes, and the SR scheme outperforms the SP scheme.

The rest of the paper is organized as follows: Section 2 describes the system model and the dynamic tree-based framework. Section 3 proposes centralized schemes for tree reconfiguration. The distributed schemes are presented in Section 4. Section 5 reports the performance evaluation results. Section 6 compares our schemes with related work. Section 7 discusses the load balance and fault tolerance issues. Section 8 concludes the paper.

2. Preliminaries

2.1. Assumptions

We consider a wireless sensor network consisting of many stationary sensor nodes. These nodes are densely deployed over a vast field to detect and continuously monitor some mobile targets. The network is connected, and the field can be completely sensed. Each sensor node knows its own location through GPS [14] or other inexpensive techniques such as triangulation [15]. We also assume that, using these techniques, the localization error is not obvious and will not affect our schemes significantly. Based on the location information, some location-based routing protocols (such as GPSR [16]) can be used for multi-hop communication between sensor nodes.

When a mobile target of interest appears in the sensing field, the sensor nodes surrounding it can detect it. A leader node (*source*) among them periodically generates sensing data about the target, based on its own detection [17] or the readings of multiple detecting nodes [18]. As the target moves

away from its current source, the source is changed to be another node closer to the target.

Some mobile hosts (e.g., PDAs) are moving within the sensing field. Each mobile host can register to a source and receive sensing data from the source. A mobile host can directly communicate with a sensor node if it is within the transmission range of the node. For simplicity, we do not consider the communication between the mobile hosts.

2.2. Matching sinks and sources

When a sink wants to receive sensing data about some target, it must register to the source that detecting the target. To facilitate a sink to find a source of interest, the index-based scheme proposed in [8] is adopted. In this scheme, some index nodes maintain the locations of sources, and a sink can query the appropriate index nodes to get the location of a source. When a source is changed, its location is updated at the related index nodes, such that the sinks can still find it. Specifically, the index nodes associated with a certain target type (say, type a) are nodes surrounding the location computed by $Hash(a)$, where $Hash(.)$ is a public hash function. Therefore, a new source node can find out the index node of the target that it detects. Similarly, a sink can also find out the index node associated with the target of interest, and hence find out the corresponding source node. More details about the scheme are available in [8].

2.3. Dynamic proxy tree-based framework

Due to the dynamic characteristics of sources and sinks, it is difficult to maintain a tree directly connecting a source and multiple sinks that are interested in the source, or disseminate data directly from the source to the sinks. To deal with the problem, we propose a proxy tree-based framework. In the framework, as shown in Fig. 1, a source (sink) is associated with a stationary sensor node called *source (sink) proxy*. As the location of the source (sink) changes, its proxy does not change until its distance to the source exceeds a certain threshold. A source proxy and the proxies of the sinks that need to frequently query the source form a proxy tree. Facilitated by the tree, sensing data is periodically pushed from the source to its proxy, and then is multicast to sink proxies in the tree. Each sink can query data from its proxy. The change of a source (sink) proxy may cause the proxy tree to be reconfigured to reduce the cost for pushing data from the source proxy to the sink proxies and from sink proxies to sinks. In the remaining of the paper, we focus on efficiently reconfiguring the proxy tree to minimize the data dissemination cost and the tree reconfiguration overhead.

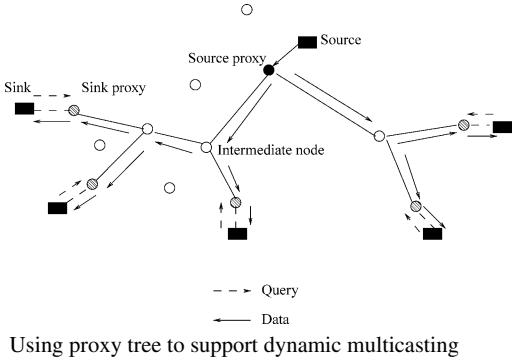


Fig. 1 Using proxy tree to support dynamic multicasting

3. Centralized tree construction and reconfiguration schemes

In this section, we study the tree construction and reconfiguration problem under an ideal assumption that there exists a centralized point that has the knowledge of the whole network and can direct sensor nodes to construct or reconfigure a tree.

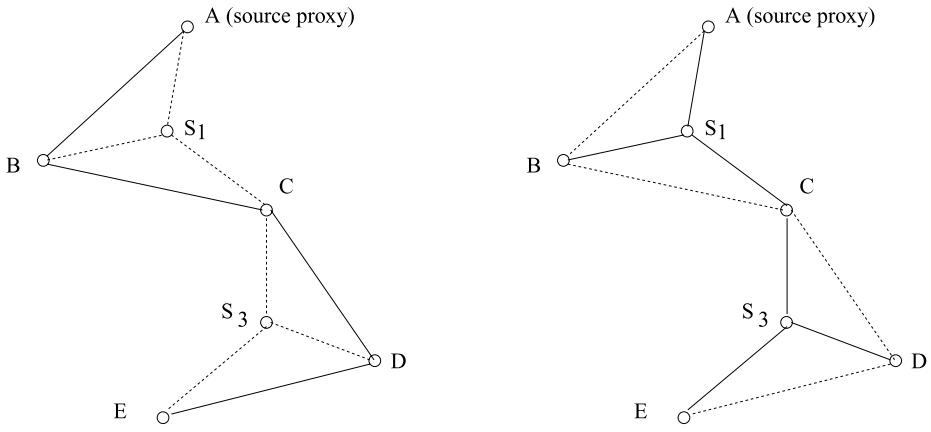
3.1. An off-line scheme

Given the topology of a sensor network and a set of sink (source) proxies, the problem of forming a minimum-cost proxy tree can be formalized as constructing a *minimum Steiner tree* [19] that connects the set of sink (source) proxies and may include some other nodes (called *Steiner nodes*) in the network. Due to dense deployment of sensor nodes, the problem can be further formulated as constructing a minimum Steiner tree in an Euclidean plane. In this section, we describe a centralized off-line scheme and several centralize on-line schemes to address the problem.

Constructing a Steiner minimum tree is known as NP-hard [20], and the exact solution has very high computational complexity. Therefore, some heuristic-based solutions have been proposed to solve the problem. For example, Smith *et al.* [21] proposed an algorithm, in which a minimum spanning tree connecting a given set of terminals is first constructed, and then the tree is optimized to approach a minimum Steiner tree. Following the idea, we present an off-line scheme for constructing an approximate minimum-cost proxy tree.

The scheme makes use of the observation that a minimum Steiner tree (minimum-cost proxy tree) is a union of *full Steiner trees (FSTs)*, and each FST is a tree with the following properties:

- It spans k ($k > 1$) terminals (proxies) and has $k - 2$ Steiner points.
- Each Steiner point has three edges making 120° with each other, and every proxy in the FST has degree one.



(a) Minimum spanning tree (b) Approximated minimum Steiner tree

Fig. 2 An illustration of the off-line scheme

Based on the above observation, a minimum-cost proxy tree can be structured in two steps: First, a minimum spanning tree (denoted as T) including all the proxies is constructed; second, T is reconfigured to be a set of FSTs. The *Kruskal*'s algorithm [22] can be used to construct T. The procedure for reconfiguring T is described as follows, and it is further illustrated by Fig. 2.

1. T is decomposed into multiple components, each with i ($i = 2, 3, \ldots, m$) proxies, where a 2-proxy component is an edge of T, and an i-proxy ($i > 2$) component is a corner that has $(i - 1)$ edges. To further explain the decomposition process, we show an example in Fig. 2, where A is a source proxy and $B - D$ are sink proxies. The minimum spanning tree is shown in Fig. 2(a). In this tree, the 2-proxy components include edges AB, BC, CD and DE; the 3-proxy components include triangles ABC, BCD and CDE.
2. For each i-proxy component T_i, a FST (denoted as $FST(T_i)$) is constructed if it exists. If there is no sensor node located at the calculated position of a Steiner point, the sensor node closest to the Steiner point is picked. In this case, the constructed FST is not the actual FST. Note that, this is different from Smith *et al.*'s algorithm [21], where the actual FST can always be constructed. The generated FSTs are placed in a priority queue Q accordingly to the value of

   $$| FST(T_i) | / | T_i | .$$

   All edges of T_i are also appended to Q. Considering the example shown in Fig. 2, for the 3-proxy component ABC, FST (S_1A, S_1B, S_1C) (where S_1 is the root, and A, B and C are leaves) is generated; for component BCD, there is no FST since $\angle BCD > 120^\circ$; for component CDE, FST (S_2C, S_2D, S_2E) is generated.
3. An approximated minimum-cost proxy tree is constructed by picking FSTs from Q in the same way as the Kruskal's algorithm. Considering the example in Fig. 2 again, the resulted Steiner tree is shown in Fig. 2(b).

3.2. On-line schemes

When a sink joins (leaves) a multicasting group, or moves far away from its current proxy, the proxy set has to change by adding (removing) a proxy, and the tree should also be reconfigured to reduce the data dissemination cost. Since it is too expensive to completely reconstruct the tree after each membership change, we borrow the idea of ARIES [9] and propose an *approximated on-line minimum Steiner tree (ONMST)* scheme.

In ARIES, as nodes are added and deleted, a rearrangement of the multicast tree is triggered within a subtree that is affected by the changes. Following this idea, as proxies join and leave, the ONMST scheme rearrange the subtrees affected by the changes. Specifically, a new proxy (denoted as P_n) is added to the current proxy tree in two steps: first, P_n is added to the current tree via the shortest path that connects the tree and P_n; second, a small subtree that contains P_n is optimized based on the locality property of the Voronoi diagram [19]. The procedure is described in the following and illustrated in Fig. 3.

1. The current proxy tree (denoted as T_c) is divided into multiple Voronoi cells, as shown in Fig. 3(a).
2. Suppose P_n is covered by the Voronoi cell of node P_i. We construct a node set $\mathcal{Y}$ which includes P_n and each node that is either a vertex of the Voronoi cell or a neighbor of a vertex of the cell. In the example shown by Fig. 3, $\mathcal{Y} = \{P_n, P_i, P_j, P_k\}$.
3. In the subgraph (denoted as G_s) of T_c which contains nodes in $\mathcal{Y}$, the off-line scheme presented in Section 3.1 is used to construct one or more approximated Steiner trees. Note that, in the tree(s), a pair of nodes is connected if and only if they are connected in T_c (except that P_n is

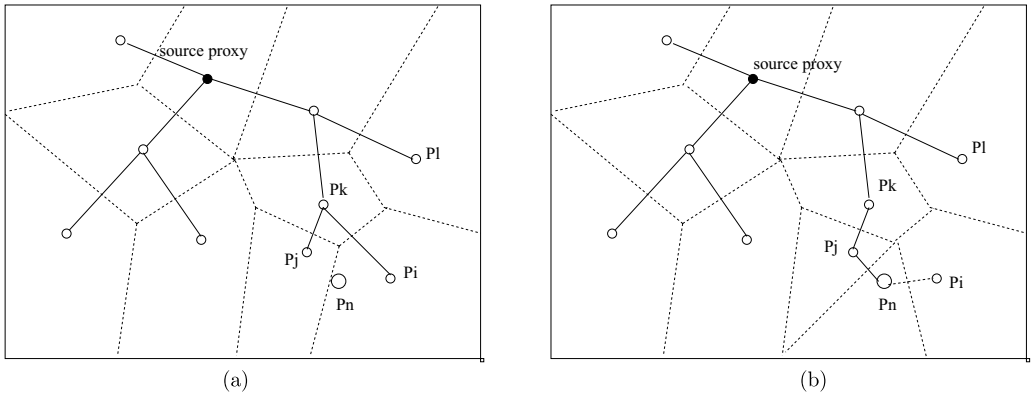


Fig. 3 Illustration of the on-line scheme

connected with P_i). The reconfigured subgraph is denoted as G'_s.

4. T_c is replaced by a new tree T'_c, which is obtained by replacing G_s with G'_s. Considering the example shown by Fig. 3, the new tree is shown in Fig. 3(b).

When a proxy should leave the current tree, it is removed only if it is a leaf.

The ONMST scheme can be further optimized by letting each Steiner node (denoted as P_i) on the tree to adjust its location every certain time interval. Let $\mathcal{Y}$ denote a node set containing the neighbors of P_i. A FST that consists of the nodes in $\mathcal{Y}$ is computed, and P_i is replaced by the newly introduced Steiner nodes in the FST. In the following, we call the enhanced ONMST scheme *E-ONMST*.

4. Distributed tree reconfiguration schemes

Even though the ONMST scheme and the E-ONMST scheme have lower complexity than the off-line scheme, they may not be suitable for sensor networks due to the following reasons: Each sensor node has only partial knowledge of the multicasting group; i.e., it only knows its neighbors in the tree. When a proxy changes from one node to another, requiring the new proxy or its neighbor to collect necessary information to construct Voronoi diagram and reconfigure the subgraph surrounding itself may incur large overhead. To address the problem, we propose two distributed heuristic-based schemes in this section.

In each distributed scheme, a proxy tree includes only a source proxy node at the beginning. When a sink wants to receive data from a source, it registers to the source. To do this, it picks a sink proxy and then the sink proxy joins the proxy tree rooted at the proxy of the source. During this process, the index nodes and the source proxy node must be contacted. Note that other events are all handled in distributed manners.

4.1. Shortest path-based (SP) scheme

The SP scheme is based on the heuristic that a new proxy (P_n) should join the current proxy tree by attaching to the tree node (P_i) that has the shortest distance to it. P_i then conducts localized reconfigurations within the subtree containing itself and its neighbors. Also, each node periodically conduct localized reconfiguration to gradually optimize the tree.

4.1.1. Proxy join and leave

In the beginning, a proxy tree includes only a source proxy node. When a sink wants to join the proxy tree, it selects a nearby sensor node (P_n) as its proxy. P_n joins the tree by going through the following three steps.

Step 1: Pre-searching. P_n obtains the location of the current source proxy (root) from the appropriate index nodes (refer to Section 2), and then sends a *join_req* to the root. On receiving the request, as shown in Fig. 4 (a), the root forwards the request to the neighbor closest to P_n. The forwarding procedure continues, until it reaches the node (P_j) which is closer to P_n than any neighbor, and a message *join_rep* is sent from P_j to P_n.

Step 2: Finding the closest node. On receiving the reply, as shown in Fig. 4(b), P_n floods a message *discover* within a circle that is centered at itself and has a radius of d_{P_n,P_j} (i.e., the distance between P_n and P_j). Every node receiving the *discover* replies its location to P_n. Based on the replies, P_n finds the node (P_i) which is closest to itself, and sends a confirm message to P_i.

Step 3: Node join. On receiving the confirm message, as shown in Fig. 4(c), P_i adds in P_n, and reconfigures the subtree containing itself and its neighbors into a FST.

When P_n wants to leave, and it is a leaf in the tree, as shown in Fig. 5(a), it leaves the tree and sends a *leave_req* to its parent. Otherwise, P_n has to stay in the tree and mark

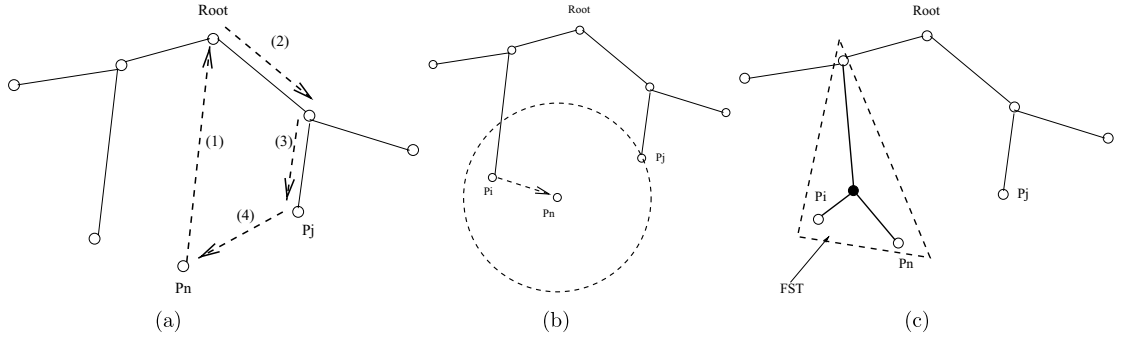
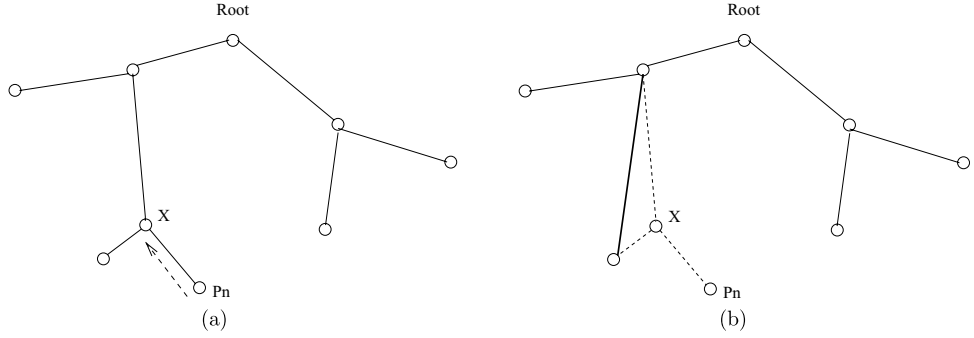


Fig. 4 A new node joins the tree: (a) Pre-searching; (b) Finding the closest node; (c) Joining the node

Fig. 5 A node leaves the tree



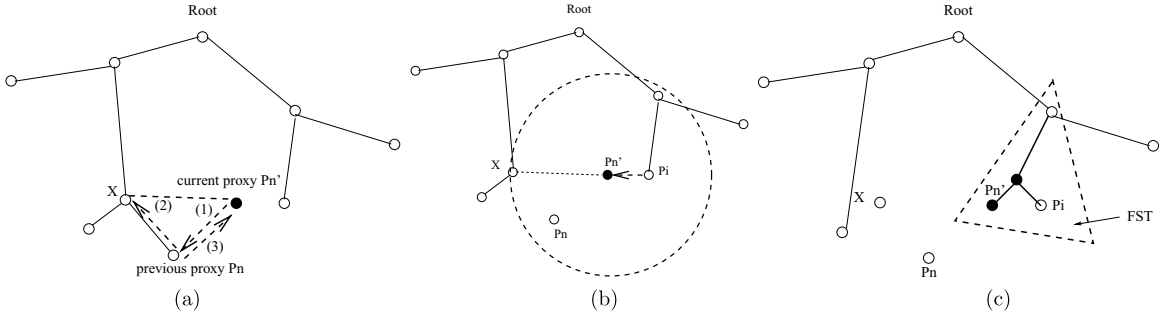


Fig. 6 Sink (source) movement initiated tree reconfiguration

itself as a Steiner node. On receiving *leave_req*, the parent node removes P_n. If the parent is a Steiner point and has only two neighbors in the tree, as shown in Fig. 5 (b), it removes itself and lets its neighbors directly connect with each other.

4.1.2. Sink (source) movement initiated tree reconfiguration

As a sink (source) moves and becomes far away from its proxy, the current proxy (P_n) should be changed to another node (P_n') which is closer to the sink (source). The tree reconfiguration initiated by a proxy change goes through the following three steps.

Step 1: Establishing a temporary edge. As shown in Fig. 6(a), P_n' sends a *migrate_req* to P_n. On receiving the message, P_n establishes a temporary edge between P_n' and its parent (denoted as X), and leaves the tree.

Step 2: Finding the closest node. As shown in Fig. 6(b), this step is similar to the Step 2 of the new proxy joining procedure. If the found closest node (P_i) is not X, P_n' tears down the temporary connection with X, and attaches to P_i.

Step 3: Joining the tree. As shown in Fig. 6(c), this step is the same as the Step 3 of the new proxy joining process.

4.1.3. Periodic localized tree reconfiguration

When a proxy moves, as shown in Fig. 6, the subtrees that it leaves or joins are reconfigured, but the remaining part of the tree is untouched even after it has been affected by

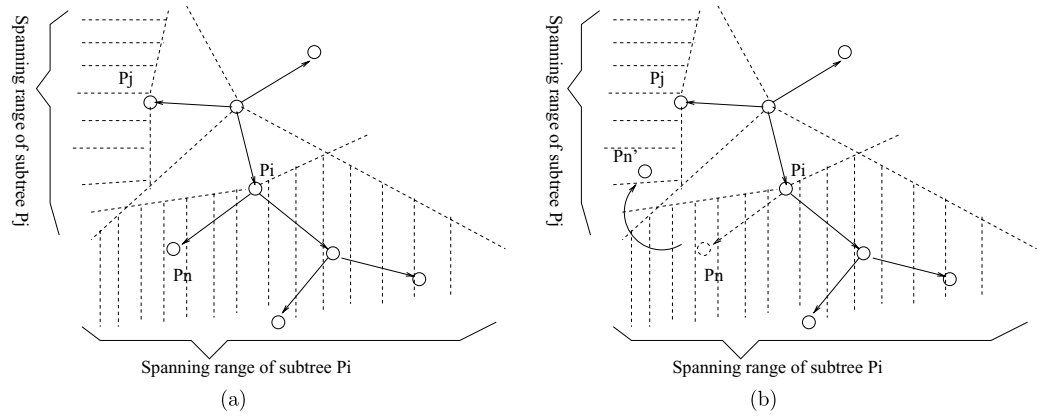
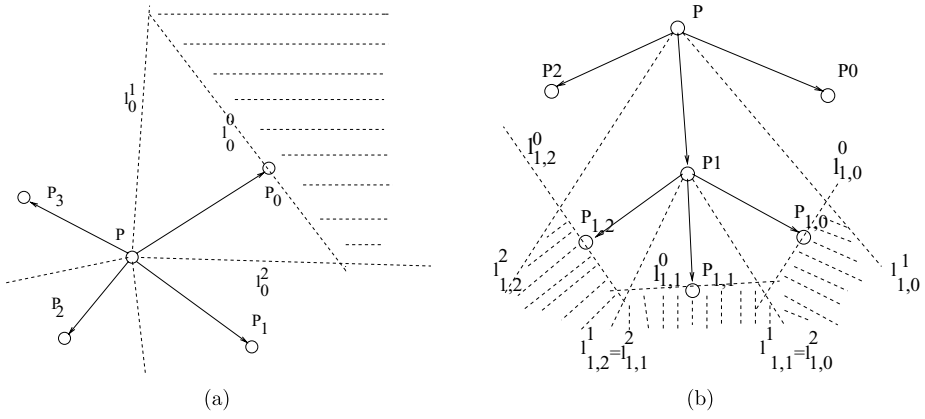


Fig. 7 The basic idea of the SR scheme

Fig. 8 Spanning ranges of nodes



the reconfigurations. To address the problem, we propose a periodic localized tree reconfiguration mechanism. With this mechanism, each Steiner point node monitors the changes of its neighbors. Every certain time, it computes the FST of the subgraph including its neighbors and finds the optimal location for itself. If the cost difference between transmitting data via the new FST and via the current subtree exceeds a certain percentage (α), the node replaces itself with the node closest to the calculated optimal Steiner point. With the periodic localized reconfiguration scheme, the tree can be gradually reconfigured with low cost.

4.2. Spanning range-based (SR) scheme

In the SP scheme, a new proxy needs to flood *discover* messages to find its position in the proxy tree. The flooding overhead can be large, especially when the multicasting members are far away from each other. To deal with the drawback, we propose a spanning range-based (SR) scheme. The basic idea of SR is illustrated in Fig. 7. As shown in Fig. 7(a), each subtree is assigned a certain spanning range, and the nodes in the subtree tree should be within the range. If a proxy (P_n) in a subtree (P_i) is changed to another one (P'_n), as shown in Fig. 7(b), P'_n should leave subtree P_i and join subtree P_j. During this process, both subtrees should be reconfigured. In the following, we first present the strategy to assign spanning ranges, and then present the algorithms for adding (removing) a proxy and dealing with source (sink) changes.

4.2.1. *Spanning range assignment*

Let P be the root of the tree, and P_i ($i = 0, \ldots, m-1$) be the children of P. As shown in Fig. 8(a), the spanning range for each P_i is the halfplane that does not cover P and is confined by the following three lines:

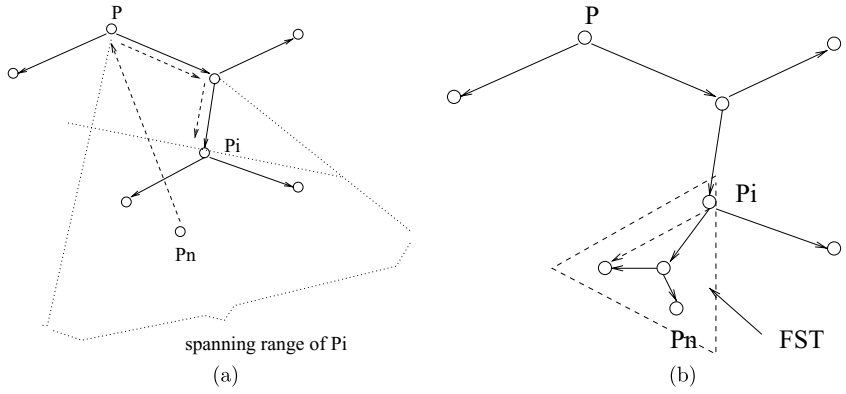
- l_i^0, which passes P_i and is perpendicular with line PP_i;
- l_i^1, which equally divides $\angle P_{i-1} P P_i$;
- l_i^2, which equally divides $\angle P_i P P_{i+1}$.

Here, P_{i-1} (P_{i+1}) is the anti-clockwise (clockwise) neighboring sibling of P_i.

For a node $P_{i,j}$ whose parent node is P_i, as shown in Fig. 8(b), its spanning range is decided as follows:

Case 1: $P_{i,j}$ is the most anti-clockwise child of P_i (e.g., $P_{1,0}$ in Fig. 8(b)). It is confined by

Fig. 9 Adding a new proxy (P_n)



- $l^0_{i,j}$, which passes $P_{i,j}$ and is perpendicular with line $P_i P_{i,j}$;
- $l^1_{i,j}$, which equally divides $\angle P_{i-1} P P_i$;
- $l^2_{i,j}$, which equally divides $\angle P_{i,j} P_i P_{i,j+1}$.

Case 2: $P_{i,j}$ is the most clockwise child of P_i (e.g., $P_{1,2}$ in Fig. 8(b)). It is confined by

- $l^0_{i,j}$, which is defined before;
- $l^1_{i,j}$, which equally divides $\angle P_{i,j-1} P_i P_{i,j}$;
- $l^2_{i,j}$, which equally divides $\angle P_i P P_{i+1}$.

Case 3: otherwise (e.g., $P_{1,1}$ in Fig. 8(b)). It is confined by

- $l^0_{i,j}$, which is defined before;
- $l^1_{i,j}$, which equally divides $\angle P_{i,j-1} P_i P_{i,j}$;
- $l^2_{i,j}$, which equally divides $\angle P_{i,j} P_i P_{i,j+1}$.

According to the spanning range assignment rule, each node on the tree can decide the spanning range of its children, and send the range to them. To reduce the overhead, the range information can be piggybacked in data packets sent from the node to its children.

4.2.2. Node join

When a mobile sink wants to join the multicasting tree, similar to the SP scheme, it selects a nearby sensor node P_n as its proxy and asks P_n to join the tree. P_n obtains the current location of the source proxy from some appropriate index nodes, and then sends a *join_req* to the source proxy (P). On receiving the request, P decides the location of P_n as follows:

(1) P calculates the spanning ranges of its children. If P_n is covered by the spanning range of a child P_i, P forwards *join_req*(P_n) to P_i.
(2) Otherwise, P adds P_n as its child. In order to add P_n at an appropriate position, P first finds a child P_j, such that $\angle P_n P P_j$ is no larger than $\angle P_n P P_i$ ($i = 0, \ldots, m-1$).
(2.1) If $\angle P_n P P_j < 120^\circ$, then a FST for triangle $P - P_n - P_j$ is calculated and replaces the subgraph containing P, P_n and P_j.
(2.2) Otherwise, P_n is directly added as a child of P.

On receiving a *join_req*(P_n) forwarded by its parent, P_i follows the same procedure as its parent to decide whether to add P_n as its child or to forward the *join_req* to one of its children. The process continues until P_n joins the tree. Figure 9 shows an example of adding a new proxy.

4.2.3. Sink movement-triggered tree reconfiguration

As a sink moves and becomes far away from its current proxy (P_n), P_n should be changed to another node (P'_n) which is closer to the sink. To conduct the migration, P'_n sends a message *migrate_req*(P'_n) to P_n. On receiving the message, P_n removes itself from the tree if it is a leaf, and sends an *add_req*(P'_n) to its parent (denoted as P_i).

When P_i receives the message, it checks if P'_n is in its spanning range. If it is still in the range, P_i follows the procedure of adding a new proxy (as described in Section 4.2.2) to add P'_n to the tree rooted at P_i. Otherwise, it sends a message *add_req*(P'_n) to its parent. The process continues until P'_n finally joins the tree.

4.2.4. Source movement-triggered tree reconfiguration

When a source becomes far away from its current proxy (P), P should also be changed to another node (P') which is closer to the source. P' becomes the new root of the proxy tree, and P becomes its child. The change of root causes the other nodes in the tree to change their spanning ranges, and the information about the new spanning ranges is passed from the root to leaves as the sensing data flow. On receiving its new spanning range, each node P_n checks its children one by one in a certain order (e.g., clockwise order), and decides whether the position of a child should be changed. Specifically, if a child $P_{n,k}$ becomes outside of the spanning range of P_n, a message *rearrange_req*($P_{n,k}$) is sent to its

parent, which decides the new position of $P_{n,k}$ in the same way as described in Section 4.2.2. Otherwise, the position of $P_{n,k}$ is unchanged.

5. Performance evaluations

We first use MATLAB to simulate the proposed centralized schemes and the distributed schemes, and compare their performance in terms of the average weight of proxy trees, without considering the tree reconfiguration overhead. After that, simulations based on NS2 are conducted in more practical scenarios to evaluate the performance of the proposed distributed schemes.

5.1. Comparing the centralized and the distributed schemes

The MATLAB-based simulations are conducted in the following settings: 516 (or 2064) nodes are uniformly distributed in a $500 \times 500m^2$ (or $1000 \times 1000m^2$) square. One target and 10 sinks move randomly within the sensing region. Data are sent from the source (whose location is the same as the target) to the sinks. The proposed centralized schemes, ONMST and E-ONMST, and the distributed schemes, SP and SR, are simulated. We use the sum of the average tree weight as the metric to compare the performance of these schemes. In the simulations, each experiment lasts for $300s$, and 60 experiments are conducted for each scheme. The average results of these experiments are shown in the figures.

As shown in Fig. 10, when the minimum spanning tree (MST) is used for data dissemination, the formed tree has the largest average weight. When the ONMST scheme is employed, the tree weight can be reduced by about 25%, since ONMST can select some Steiner points to reduce the tree weight. The tree weight can be further reduced by about 20% when using the E-ONMST scheme, which can periodically optimize the tree reconfigured by ONMST. The shortest path-based (SP) scheme and the spanning range-based (SR) scheme have about 9% and 12% higher tree weight than the E-ONMST scheme, respectively, because they use less information to reconfigure the tree.

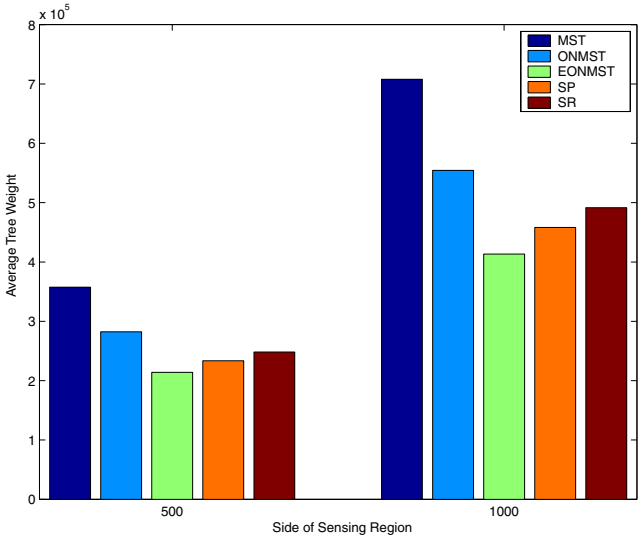


Fig. 10 Comparing the tree weights of different schemes (average velocity = 2.5m/s, localized reconfiguration interval = 1s)

5.2. Evaluating the distributed algorithms

5.2.1. Simulation model

In the NS2-based simulations, the IEEE 802.11 MAC layer protocol and the location-based GPSR routing algorithm are employed. We uniformly deploy 516 sensor nodes over a $500 \times 500m^2$ field. Each sensor node has a communication range of $40m$. One target and 10 sinks move randomly in the field, and the way-point model is used to simulate their movement. As a sink or a source (target) moves $80m$ away from its current proxy, the sensor node closest to it is selected as the new proxy.

We evaluate the following metrics:

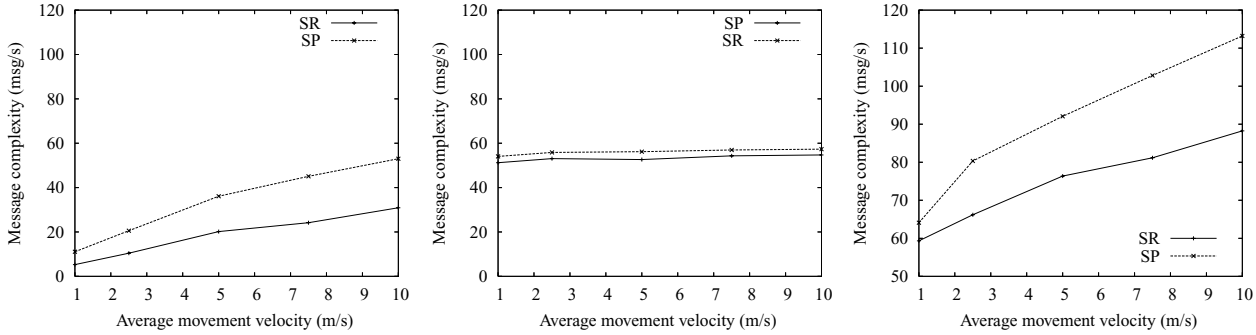
- **Control message complexity:** the number of control messages transmitted in the network.
- **Data message complexity:** the number of data messages transmitted in the network.
- **Overall message complexity:** the sum of the control message complexity and the data message complexity.

In the simulations, each experiment lasts for $300s$, and 60 experiments are conducted for each scheme. The average results of these experiments are shown in the figures.

5.2.2. Comparing SR and SP

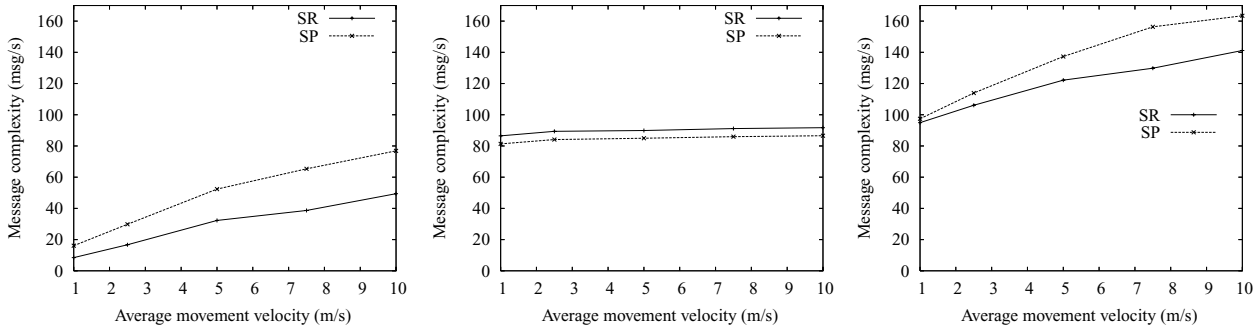
Figure 11(a) shows that SR has smaller control message complexity than SP, which is due to the following reasons: As a sink (source) changes its proxy, SP floods *discover* messages within a certain area to let the new proxy join the proxy tree. After that, the new proxy has to exchange several messages with the tree nodes within the flooding area to select the appropriate parent node. However, when SR is used, only a few messages need to be sent, because the new proxy is usually still within the spanning range of the parent of the previous proxy. Thus, it can immediately join the subtree rooted at the parent node. Even if the new proxy is out of the spanning range, a reconfiguration is conducted in the smallest subtree that covers the new proxy, and the process will not incur many control messages.

Figure 11(b) shows that SR has slightly larger data message complexity than SP. This phenomenon is consistent to



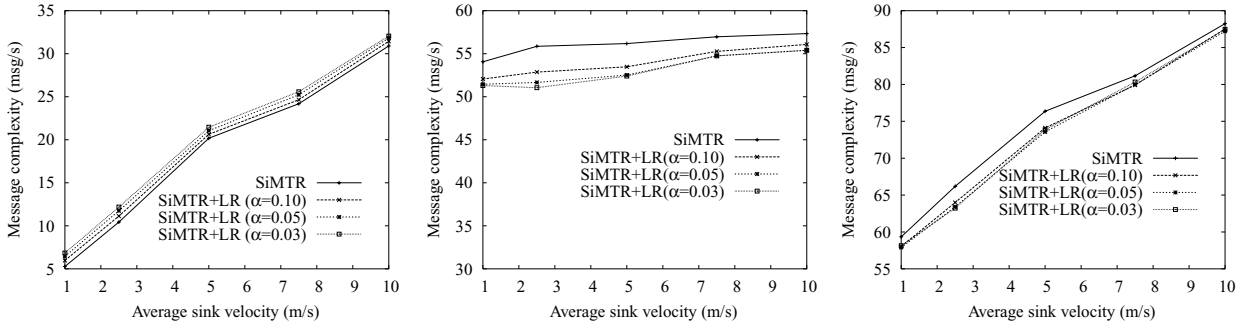
(a) Control message complexity (b) Data message complexity (c) Total message complexity

Fig. 11 Comparing SR and SP



(a) Control message complexity (b) Data message complexity (c) Total message complexity

Fig. 12 SR and SP (Number of sinks: 20)



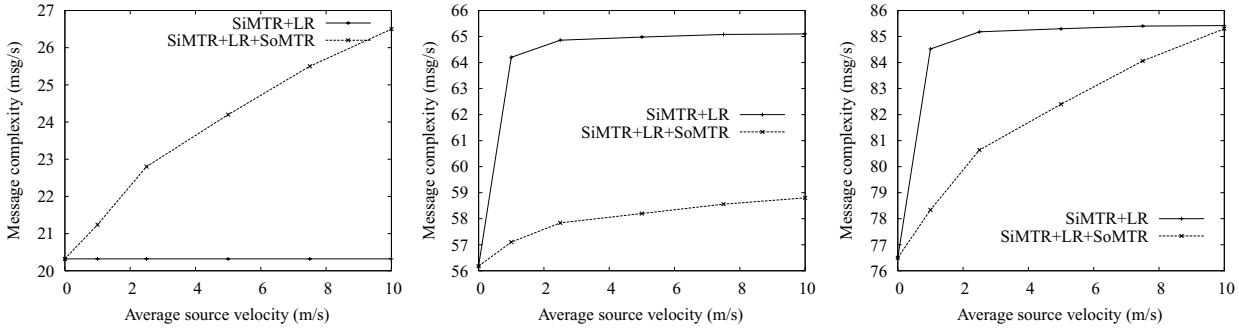
(a) Control message complexity (b) Data message complexity (c) Overall message complexity

Fig. 13 Impact of the localized reconfiguration mechanism (SiMTR = Sink movement-initiated tree reconfiguration)

that shown in Fig. 10, which verifies that the shortest path heuristic is slightly better than the spanning range heuristic. However, as shown in Fig. 11(c), SR outperforms SP in terms of the overall message complexity. Figure 12 demonstrates the simulation results when the number of sinks is increased to 20. As we can see, the trend is unchanged.

5.2.3. Impact of the localized reconfiguration (LR) mechanism

Figure 13(a) shows that using the LR mechanism increases the control message complexity. Also, the control message complexity increases as the system parameter α decreases. This is due to the reason that the localized reconfiguration is conducted more frequently as α becomes smaller. However, as shown in Fig. 13(b), using the LR mechanism can decrease the data message complexity, and the data message complexity decreases as the system parameter α decreases. When α is very small (e.g., 0.05), decreasing the parameter does not significantly decrease the data complexity. This is due to the reason that the node density is not large enough, and hence there may not exist a node at the optimal location to further minimize the cost when α is too small. Figure 13(c) shows that, with an appropriate parameter



(a) Control message complexity (b) Data message complexity (c) Overall message complexity

Fig. 14 Impact of the SoMTP mechanism (average sink velocity = 5.0 m/s)

α, using the LR mechanism can reduce the overall message complexity.

5.2.4. Impact of the source movement-initiated tree reconfiguration (SoMTR) mechanism

Figure 14(a) shows that using the SoMTP mechanism increases the control message complexity, and the complexity increases as the source velocity increases. This phenomenon can be explained as follows: As the source moves, the source proxy changes accordingly. When the SoMTP mechanism is employed, changing the source proxy causes some nodes to migrate from one branch to another. To do the reconfigurations, some control messages are exchanged. As the source velocity increases, the source proxy changes more frequently, and introduces more control messages.

Figure 14(b) shows that using the SoMTP mechanism can reduce the data message complexity. This can be explained as follows. If the SoMTP mechanism is not used, the tree is not reconfigured as the source proxy changes. Thus, data should be transmitted from the source to the previous proxy (root) before being transmitted to the sink proxies. However, when the SoMTP mechanism is used, the tree structure is optimized as the source proxy changes, and hence reduces the data dissemination cost.

Figure 14(c) compares the overall message complexity based on whether the SoMTP mechanism is used or not. As shown in the figure, as the source velocity increases, the SoMTP mechanism can significantly reduce the overall message complexity when the source velocity is small. However, the difference becomes smaller when the source velocity is very large. The reasons can be found from Fig. 14(a) and (b). As the source velocity is small, the increasing rate of the control message complexity is much slower than the reducing rate of the data message complexity. However, as the source velocity becomes very large, the increase of the control message complexity is similar to the decrease of the data message complexity.

6. Related work

The Euclidean Steiner Tree Problem (ESTP) is known as NP-hard, and the history of heuristics for ESTP dates back to the early 1970s. Most heuristics [23–25, 21] from the literature use a Minimum Spanning Tree (MST) of the given set of terminals as the initial solution, and then optimize it to approach a Steiner Minimum Tree. In particular, Smith, Lee and Liebman [21] proposed a heuristic, in which a list of full Steiner trees (FSTs) are constructed, and a subset of the FST sets are found such that the FSTs spans all terminals and the resulting tree is as short as possible. The scheme has a computational complexity of $O(n \log n)$, and it is both theoretically and practically the fastest heuristic known for ESTP. Although our centralized schemes are based on this heuristic, our schemes are different in the following aspects: First, if there is no sensor node located at the computed Steiner point, the sensor node closest to the point is picked; second, we consider the situations that nodes may join or leave the multicasting group over the time, and propose on-line schemes which have lower overhead than applying the heuristic directly.

To support dynamic multicasting in the Internet, two types of schemes have been proposed: those that allow rearrangement of the tree, called rearrangeable on-line schemes [26], and those do not, called non-rearrangement on-line schemes [27, 28, 9]. Among them, the heuristic ARIES [9] performs the best. ARIES adds a new node to an existing tree via the shortest path, and only the subtrees including newly added or deleted nodes are reconfigured every certain time. Our centralized off-line schemes and distributed schemes are based on the same idea that only the affected subtrees (not the whole tree) are configured after nodes join or leave. Different from ARIES, we do not require each multicasting member to be aware of the whole network topology or its distance to other members. The requirement may be feasible for the Internet, where routers can naturally obtain the information through the underlying topology advertisement protocol. However, it is not applicable to sensor networks, where running the topology advertisement protocol may cause large overhead.

Many multicasting protocols [10–12] have been proposed for mobile ad hoc networks (MANETs). These protocols emphasize on route robustness since MANETs have frequent path breaks due to high node mobility. Our schemes also consider the mobility of sources and sinks. However, we stress more on energy efficiency due to the strict energy constraints in sensor networks. In particular, we propose a proxy-based framework to address the mobility problem and focus on the problem of efficiently reconfiguring the proxy tree to accommodate the mobility of sinks and sources.

In the area of wireless sensor networks, a number of data dissemination schemes can support multicasting. For example, the directed diffusion scheme [3] asks sinks to flood their interests to the sensor network, and asks sources matching the interests to send their data to the sinks along multiple paths. In the TTDD scheme [4], each source maintains a system-wide grid-based structure, through which the availability advertisement and data are disseminated. These schemes, however, do not efficiently support sink or source mobility since the changes of sink or source positions cause large overhead, especially when the network size is large. In addition, the data dissemination structure is not optimized to reduce the cost. Particularly, in [3], data are usually disseminated along the shortest paths connecting the source and sinks, with only opportunistic optimizations; in [4], data is always disseminated along the grid structure. Different from the those schemes, our proposed schemes can address the above issues. Recently, Huang *et al.* [29] studied spatiotemporal multicasting in sensor networks. The study optimizes multicast from a source to a certain geographic area, while our schemes optimize multicast from a source to multiple sinks scattered over the network. Kim and Bhattacharya *et al.* [30, 31] proposed asynchronous multicasting schemes for sensor networks, in which a source pushes data to multiple replicas in the network, and sinks query their nearby replicas to get the disseminated data. These schemes mainly studied how to optimize the replica placement based on factors such as the sink locations, the data updating rate and the data querying rate. However, our schemes focus on optimizing the data dissemination tree structure to minimize the dissemination cost.

7. Discussions

7.1. Load balance and system lifetime

If sinks and sources are stationary, using the proposed schemes, sensing data will be disseminated along the same tree repeatedly. As a result, sensor nodes in the tree will consume more energy than nodes off the tree, which will eventually lead to network partition and thus short network lifetime. Therefore, the proxy tree should alter gradually. In this paper, however, we assume that the sources could dynamically change and the sinks are mobile. If the mobility pattern is balanced in the network, the energy consumption will be balanced among the sensor nodes. Otherwise, some measures could be taken to address the issue.

One approach for balancing the energy consumption is to let nodes rotate to serve on the tree. Specifically, when a node has been on the tree for a certain time period $\max_{on}$, it leaves from the tree and some nearby node is picked up to replace it. The quiting node will not be recruited again until it has been off the tree for a certain time period min_{off}.

Proxy nodes themselves may also be overloaded, and the above approach can be applied on them as well. In particular, when a sink (source) proxy has been on duty for more than $\max_{on}$, it sends a notice to the associated sink (source), which will pick another node that has been off the tree for more than $\min_{\text{off}}$, as its new proxy node.

7.2. Node failures

Some nodes on the tree may fail for some reasons. If a sink (source) proxy fails, the associated sink (source) will detect the failure since it cannot query (push) sensor data successfully. If a node on the path connecting a proxy node and a Steiner node (or two Steiner nodes) fails, the failure can be tolerated by the underlying routing protocols. This is due to the reason that the data sending from an upstream proxy (Steiner) node to a downstream Steiner (proxy) node using a geographic routing such as Greedy Perimeter Stateless Routing (GPSR) [16], which can avoid defective points or areas.

If a Steiner node on the tree fails, the information about its downstream Steiner (proxy) nodes is lost. Therefore, we need a method that not only detects the failure but also rescues the lost information. The method can be designed as follows: When a node is selected as a Steiner node, its location is recorded at its parent Steiner (proxy) node; meanwhile, the location information of its downstream Steiner (proxy) nodes is replicated at the nodes surrounding it. Later, if the Steiner node fails, the message disseminated from its parent Steiner (proxy) node will eventually reach the node closest to the failed node, and the node will be picked to replace the failed node. Recall that the location information of its downstream Steiner (proxy) nodes is replicated at this node. Therefore, the message can still be disseminated along the tree.

8. Conclusions

In this paper, we addressed the problem of efficient dynamic multicasting in wireless sensor networks. We proposed a dynamic proxy tree-based framework, and focused on the issue of efficiently reconfiguring the proxy tree as proxies frequently change from one node to another. The problem was modeled as on-line reconstructing a Steiner minimum tree in

an Euclidean plane. Some centralized on-line schemes were proposed to solve the problem. Considering the strict energy constraints and the locality requirements in wireless sensor networks, we further proposed two distributed schemes, the shortest path-based (SP) scheme and the spanning range-based (SR) scheme. Extensive simulations were conducted to evaluate the proposed schemes. The results showed that the distributed schemes can achieve similar performance as the centralized schemes, and the SR scheme outperforms the SP scheme.

Acknowledgments We would like to thank the anonymous referees whose insightful comments helped us to improve the presentation of the paper. A preliminary version of the paper appeared in IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS), 2004. This work was supported in part by the US National Science Foundation under grant CNS-0519460.

References

1. I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E.Cayirci, Wireless Sensor Networks: A Survey, Computer Networks 38(4) (March 2002).
2. W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, Energy-Efficient Communication Protocal for Wireless Microsensor network, in: *Proc. of the Hawaii International Conference on System Sciences* (January 2000).
3. C. Intanagonwiwat, R. Govindan, and D. Estrin, Directed Diffusion: A Scalable and Robust Communication, MobiCOM '00 (August 2000).
4. F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, A Two-Tier Data Dissemination Model for Large-scale Wireless Sensor Networks, *ACM International Conference on Mobile Computing and Networking (MOBICOM'02)* (September 2002) pp. 148–159.
5. S. RatNasamy, B. karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, GHT: A Geographic Hash Table for Data-Centric Storage, *ACM International Workshop on Wireless Sensor Networks and Applications* (September 2002).
6. A. Ghose, J. Grobklags and J. Chuang, Resilient data-centric storage in wireless ad-hoc sensor networks, in: *Proceedings the 4th International Conference on Mobile Data Management (MDM'03)* (2003) pp. 45–62.
7. B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy and S. Shenker, DIFS: A Distributed Index for Features in Sensor Networks, *First IEEE Ineternational Workshop on Sensor Network Protocols and Applications* (May 2003).
8. W. Zhang, G. Cao, and T. La Porta, Data Dissemination with Ring-Based Index for Sensor Networks, *IEEE International Conference on Network Protocol* (ICNP) (November 2003).
9. F. Bauer and A. Varma, ARIES: A Rearrangeable Inexpensive Edge-Based On-Line Steiner Algorithm, IEEE Journal of Selected Areas in Communications 15(3) (1997) 382–397.
10. S. Lee, W. Su, and M. Gerla, On-Demand Multicast Routing Protocol (ODMRP) for Ad Hoc Networks, IEEE ICNP'98 (1998).
11. M. Liu, R. Talpade, A. Mcauley, and E. Bommaiah, Ad Hoc Multicast Routing Protocol (AMroute), UMD TechReport 99-8 (1999).
12. H. Laboid and H. Moustafa, Source Routing-based Multicast Protocol (SRMP), Internet Draft (2001).
13. N. Alon and Y. Azar, On-line Steiner Trees in the Euclidean Plane, (2000).
14. US Naval Observatory (USNO) GPS Operations (April 2001). http://tycho.usno.navy.mil/gps.html.
15. N. Bulusu, J. Heidemann, and D. Estrin, GPS-less Low Cost Outdoor Location For Very Small Devices, IEEE Personal Communication, Special Issue on "Smart Space and Environments" (October 2000).
16. B. Karp and H. Kung, GPSR: Greedy Perimeter Stateless Routing for Wireless Networks, *The Sixth Aunual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)* (August 2000).
17. M. Chu, H. Haussecker and F. Zhao, Scalable Information-driven Sensor Querying and Routing for Ad Hoc Heterogeneous Sensor Networks, International Journal of High Performance Computing Applications (2002).
18. W. Zhang and G. Cao, Optimizing Tree Reconfiguration for Mobile Target Tracking in Sensor Networks, IEEE Infocom'04 (March 2004).
19. A. Ivanov and A. Tuzhilin, *Minimal Networks* (CRC Press, Inc., 1994).
20. F. Hwang, D. Richards, and R. Winter, The Steiner Tree Problem, Annals of Discrete Mathematics 53. Elsevier Science Publishers (1992).
21. J. Smith, D. Lee, and J. Liebman, An O(nlogn) Heuristic for Steiner Minimal Tree Problems on the Euclidean Metric, Networks 11 (1981) 12–29.
22. T. Cormen, C. Leiserson and R. Rivest, *Introduction to Algorithms* (The MIT Press, 1990) pp. 514–543.
23. E. Thompson, The Method of Minimum Evolution, Annals of Human Genetics 36 (1973) 333–340.
24. S. Chang, The Generation of Minimal Trees with a Steiner Topology, J. Assoc. Comput. Mach. 19.
25. F. Hwang, A Linear Time Algorithm for Full Steinter Trees, Operations Research Letters 4(5) (1986) 235–237.
26. B. Waxman, Routing of Multipoint Connections, IEEE Journal on Selected Areas in Communications 40(9) (Dec. 1988) 45–72.
27. M. Imase and B. Waxman, Dynamic Steinter Tree Problem, SIAM J. Disc. Math. 4(3) (Aug. 1991) 369–384.
28. J. Kadirire, Comparison of Dynamic Multicast Routing Algorithms for Wide-Area Packet Switched (Asynchronous Transfer Mode) Networks, IEEE INFOCOM (April 1995) 212–219.
29. Q. Huang, C. Lu, and G. Roman, Spatiotemporal Multicast in Sensor Networks, *ACM Conference on Embedded Networked Sensor Systems (Sensys'03)* (Nov. 2003).
30. S. Bhattacharya, H. Kim, S. Prabh and T. Abdelzaher, Energy-Conserving Data Placement and Asynchronous Multicast in Wireless Sensor Networks, ACM Mobisys'03 (2003).
31. H. Kim, T. Abdelzaher, and W. Kwon, Minimum-Energy Asynchronous Dissemination to Mobile Sinks in Wireless Sensor Networks, *ACM Conference on Embedded Networked Sensor Systems (Sensys'03)* (Nov. 2003).