Wireless Netw
DOI 10.1007/s11276-016-1337-5

A time efficient aggregation convergecast scheduling algorithm for wireless sensor networks

Cheng Pan[1] · **Hesheng Zhang**[1]

Abstract We investigated the aggregation convergecast scheduling problem in wireless sensor networks. In order to reduce the time needed for data collection through aggregation convergecast, we propose a scheduling algorithm based on an aggregation tree which enables a small delay lower bound and a time slot allocation method which uses the time slots efficiently. To achieve a small delay lower bound, we take the sum of the receiver's depth and child number as the cost of the transmission links and then construct an aggregation tree gradually by adding to it a link with the minimum cost iteration by iteration. To use the time slots efficiently, we use a neighbor degree ranking algorithm together with a supplementary scheduling algorithm to allocate time slot for the sensor nodes. Experiments show that the proposed scheduling algorithm outperforms other work in most cases by reducing the number of time slots needed for data collection by more than 10 %, which indicates the feasibility of our approach for data collection in wireless sensor networks.

Keywords Wireless sensor networks · Data collection · Data aggregation · Convergecast · Scheduling

✉ Hesheng Zhang
hszhang@bjtu.edu.cn

Cheng Pan
pattision@bjtu.edu.cn

[1] School of Electrical Engineering, Beijing Jiaotong University, Beijing, China

1 Introduction

Wireless sensor networks are self-organized systems constructed by plenty of sensor nodes through wireless communication. One of the main tasks of wireless sensor networks is data collection. To fulfill the task, all the sensor nodes in the network need to transmit their data to a sink node. The many-to-one transmission pattern is called convergecast [1].

In many applications of wireless sensor networks, sensor nodes are densely deployed. The convergecast of all raw data will lead to a burst of traffic load. Since data are usually spatial correlated in wireless sensor networks, in-network aggregation can be applied to reduce the traffic load. Aggregation refers to the scheme to have the relay nodes merge their received data packets with their own ones by means of data fusion [2], data compression [3], aggregation function [4], etc. By applying aggregation scheme in the process of convergecast, each sensor node needs only to transmit once and their transmission links form a tree topology, which is called the aggregation tree.

In the course of convergecast, random contention based transmission would cause lots of packet collisions and thus long time delay. However, reducing the time needed for data collection is an important issue in many monitoring applications, especially for those time-critical and safety–critical systems. To avoid packet collision and reduce time delay, transmission scheduling is necessary. By scheduling, the procedure of convergecast is divided into several time slots and the transmissions assigned in the same time slot are collision-free.

In this paper, we investigate the problem of Aggregation Convergecast Scheduling (ACS), aiming at providing collision-free schedule that enables wireless sensor networks to collect data as fast as possible. A solution of ACS

Published online: 29 August 2016

problem should provide for every sensor node two pieces of information–the one-hop target node and the time slot to send its data. Correspondingly, an aggregation tree construction phase and a time slot allocation phase are included in the scheduling algorithms for aggregation convergecast.

The existing algorithms on ACS problem usually construct aggregation tree based on Shortest Path Tree (SPT) [1] or Connected Dominating Set (CDS) [5] and allocate time slots following a similar framework as described in section II. The problem is that the SPT and CDS based aggregation trees may result in a large delay lower bound and the existing time slot allocation framework does not make full use of each time slot.

Therefore, we propose a heuristic algorithm focusing on reducing the delay lower bound of the aggregation tree and using the time slots efficiently. Our contributions are as follows.

- Minimum Lower bound Spanning Tree (MLST) algorithm for aggregation tree construction. We define the cost of a transmission link to be the sum of the receiver's depth and child number and build an aggregation tree by adding a link with the minimum cost iteration by iteration. This tree produces small delay lower bound and thus enables more potential to get a shorter convergecast delay.
- Time slot allocation algorithm based on Neighbor Degree Ranking (NDR). We use the sum of the neighbor nodes' degree to characterize the density of each node's ambient region and allocate time slot preferentially to the nodes in the dense area. This algorithm can alleviate the bottleneck effect of the network.
- Supplementary Scheduling (SS) scheme. To make full use of each time slot, we add to the time slot allocation framework a supplementary scheduling scheme, which takes possible transmission links out of the aggregation tree into consideration after the tree links are checked. This scheme improves the performance of time slot allocation algorithm by introducing additional concurrent transmissions in each time slot.

The rest of this paper is organized as follows. Sect. 2 reviews the related work. In Sect. 3, we described the ACS problem in detail. In Sect. 4, we describe our scheduling algorithm for aggregation convergecast including an aggregation tree construction algorithm, a time slot allocation algorithm, and a supplementary scheduling algorithm. Sect. 5 evaluates the performance of our aggregation scheduling algorithm through simulation experiments. Sect. 6 concludes the paper.

2 Related work

As a key problem in data collection of wireless sensor networks, ACS problem has been well researched in recent years. Most algorithms in the literature solve the problem by two consecutive phases: first the aggregation tree construction, and then the time slot allocation.

In terms of computational complexity, Chen et al. [6] has proved that ACS problem is NP-hard. Even on a given aggregation tree, optimal time slot allocation problem is still NP-complete [7].

According to the way of aggregation tree construction, the existing ACS algorithms can be divided into SPT based ones, CDS based ones and other ones.

In SPT based algorithms, every sensor node transmits data to the sink node through a path with minimum hops, which is intuitively able to reduce transmission delay. Incel et al. [8] construct an SPT with the constraint of child node number in order to avoid the situation that some nodes become bottle neck of the network due to too many child nodes. The father-child matching in every two adjacent levels in the SPT is considered in [1]. They take child nodes as load of father nodes and get a load balance matching by adopting a bipartite graph semi-matching algorithm [9]. SPT is also used by [6] as its initial tree which is adjusted during the time slot allocation process. SPT based algorithms minimize the effect of node depth on convergecast delay, but they do not take the potential of link conflict into consideration. Since all the transmission links are directed from the periphery of SPT to the sink node, there will be a large amount of link conflict in the region near the sink node.

In CDS based algorithms, a maximal independent set of nodes is usually used as the dominating nodes, and then connector nodes are chosen, these nodes form a CDS [10], which is regarded as the backbone of the aggregation tree. Due to the topological feature of CDS, this sort of algorithms can be proved to have an upper bound of data collection delay such as in [5, 11, 12], and [13]. The upper bound is usually a function of network radius (R) and maximum node degree (Δ). For instance, the algorithm in [13] has a delay upper bound of $16R + \Delta - 14$, and the upper bound of the algorithm in [12] is $4R + 2\Delta - 2$. The aggregation tree structure design in CDS based algorithm is devoted to theoretical upper bound derivation. Although it is useful to have a delay upper bound guarantee, as pointed out in [1], the upper bounds of those algorithms are far too pessimistic compared to their practical performance. Besides, in the CDS based algorithms, a dominating node is likely to be a node of very large degree with almost all of its neighbor nodes being its children. This may cause a long time for the parent nodes to wait for its child nodes to transmit and thus have negative effect on the performance

of those algorithms. SPT has the similar problem, because the sink node of SPT takes all its neighbors as its children. As we will discuss later, SPT and CDS based aggregation trees also have a large delay lower bound due to the above problem.

De Souza et al. form an aggregation tree by combining an SPT and a minimum interference tree built by Edmonds' algorithm [14]. The performance of the combined tree depends largely on the combination ratio α that is chosen by experiment, which is not an efficient way. Also, their timeslot allocation algorithm is non-polynomial.

Concerning the time slot allocation, the algorithms in many of the existing literature follow a similar framework. First, sort the leaf nodes of the aggregation tree according to a particular ranking criterion. Then allocate the current time slot to the leaf nodes successively, as long as there is no link conflict. After that, remove the nodes which have already got a time slot from the tree. The above procedure is repeated until all the nodes are scheduled. The main difference of the existing time slot allocation algorithm is how they determine the priority of the nodes. In [5] and [12], the nodes are ordered simply according to their ID, which approximately result in a random way of time slot allocation. This kind of scheme is convenient for distributed implementation, but it is not good enough from the delay optimization point of view. In [1], the non-leaf neighbor count is taken as the node ranking criterion. By scheduling the most "constrained" nodes first, their method enables more other nodes to become "eligible" for subsequent time slots. The existing work does not make the most of each time slot since they only pay attention to the child–parent links in the aggregation tree, we will show in our algorithm that there are still chances to have more nodes transmit after each round of time slot allocation.

3 Network model and problem statement

In this section, we introduce the network model and define the ACS problem.

3.1 Network model

We investigate the ACS problem on top of a wireless sensor network which consists of plenty of sensor nodes and a single sink node. The sensor nodes are deployed on a two-dimensional Euclidean plane. Data is generated in each sensor node and need to transmit to the sink through multi-hop wireless communication. Every node in the network has a single half-duplex transceiver and only one wireless channel is used.

We adopt protocol model [15] for interference. In the protocol model, the transmission range of node v is a disk centered in v. A pair of nodes can communicate with each other if and only if the distance between them is shorter than the radius of their transmission range. For the convenience of discussion, we assume that the transmission radius is the same for all the nodes in the network and that the interference range of each node is equal to its transmission range. In the protocol model, a collision occurs when more than one node send data packet simultaneously in the interference range of a receiving node.

Let V denotes the set of all the nodes and E denotes the set of all the edges, the topology of the wireless sensor network can be described by a graph $G = (V, E)$. In the graph, there is an edge between a pair of nodes if and only if they are in each other's transmission range.

Some concept relevant to the graph are defined below:

- *Neighbor*: a pair of nodes within each other's transmission range are neighboring nodes, that is, they are each other's neighbor.
- *Degree*: the degree of a node is the number its neighbors in the graph.
- *Sender/Receiver*: a node which sends/receives data packet.
- *Transmission Link* (u, v): a directed edge pointed from a sender u to a corresponding receiver v.
- *Link conflict* (*transmission conflict*): two transmission links conflict with each other if the simultaneously transmission through them leads to a collision. In other words, link (u, v) conflict with link (u', v') if u and v' are neighbors or v and u' are neighbors.

3.2 Problem statement

The *data aggregation* mentioned in this paper refers to the scheme to merge multiple data records from different sensors into one data packet. Data aggregation is applied on every intermedia node so that they can merge all the received data with their own data. We assume that the merged data can always be encapsulated in a packet with a fixed size.

Due to the adoption of data aggregation, each sensor node needs only to send one data packet after receiving all the packets it has to relay. And all the data should be transmitted to the sink node at last. In this case, from the graph theory point of view, there is one outgoing link for every sensor node and these links should not form a cycle, which indicates that the transmission links and the nodes actually form a tree rooted at the sink node. We call it *aggregation tree*. Here we define some relevant concept.

- *Aggregation tree*: A tree constructed by the chosen transmission links for aggregation convergecast.

- *Parent*: the receiver of a transmission link in the aggregation tree is called the parent of the corresponding sender.
- *Child*: the sender of a transmission link in the aggregation tree is called the child of the corresponding receiver.
- *Root*: the node which has no parent in the aggregation tree. The sink node is the only root of the aggregation tree.
- *Leaf*: a node which has no child in the aggregation tree.
- *Depth*: the depth of a node is the hop distance in the tree from the node to the root; the depth of a tree is the maximum depth of the nodes in the tree.

The nodes are assumed to have synchronized clocks, so that the aggregation convergecast can be conducted in a TDMA way. We also assume that each time slot has equal length, which is sufficient for a data packet to be sent, received, and aggregated. The total number of time slots needed to collect the data from all the sensor nodes is regarded as the convergecast delay.

An aggregation convergecast schedule chooses a parent for every node to send its data packet to and assigns a time slot for every child–parent link to transmit. The goal is to achieve a conflict free aggregation convergecast and minimize the convergecast delay.

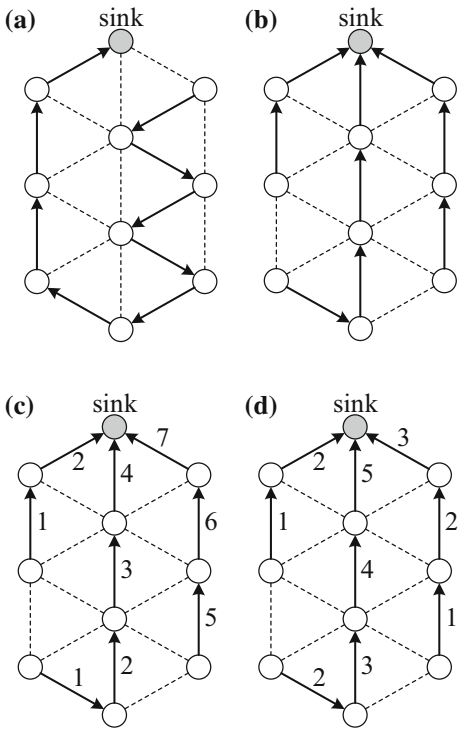

Fig. 1 An example of aggregation convergecast scheduling. **a** Line topology, **b** tree topology, **c** a time slot allocation scheme, **d** another time slot allocation scheme

Figure 1 shows an example of aggregation convergecast scheduling. In the figure, the solid lines with arrowheads represent the selected data transmission links, while the dotted lines represent other edges in the graph. Figure 1(a) and (b) are two different aggregation trees. (a) has one path go through all the nodes and needs at least 9 time slots to finish data collection, while (b) constructs a multi-branched tree topology, which enables some links to transmit concurrently and reduce the number of time slots needed for data collection. (c) and (d) provide two time slot allocation scheme based on the aggregation tree of (b). The numbers beside the links represent the time slots chosen for them to transmit. The total number of time slots needed for (c) and (d) to finish data collection are 7 and 5 respectively. That is to say, different tree topology and different time slot allocation scheme can both lead to different data collection delay.

Table 1 presents the notations used in this paper.

4 Scheduling algorithm

In this section we introduce our aggregation convergecast scheduling algorithm. To determine where and when to send the data packet for the sensor nodes, an aggregation tree construction phase and a time slot allocation phase of the scheduling algorithm are proposed respectively.

4.1 Aggregation tree construction

An aggregation tree is formed by the sender-receiver links chosen to be used in the convergecast process. This tree should be carefully constructed because the structure of the tree affects the convergecast delay greatly. As mentioned by Malhotra et al. [1], the convergecast delay can be lower bounded by the structure of the tree. A tree with smaller delay lower bound can be considered to have more potential to produce a schedule with shorter delay. We propose an aggregation tree construction algorithm to maximize this potential.

Malhotra et al. [1] have proved that the lower bound of the delay for the ACS problem is $max\{|C_v| + h_v : v \in V\}$, where C_v and h_v are the children set and the depth of node v in the aggregation tree. They also provided a balanced shortest path tree (BSPT) construction approach to produce a minimum-lower-bound SPT. Their approach minimizes the maximum child number in each depth of the tree. However, since the aggregation tree is pre-determined to be an SPT, the lower bound get from the tree is not an optimum of overall situation.

Take the network in Fig. 2 for a simple example. In the network, the 6 sensor nodes (*a–f*) are all neighbors of the sink node. The only BSPT on the network will be the one

Table 1 Notations used in tins paper

Notation	Meaning
V	The set of all the nodes in the network
s	The sink node
E	The set of all the edges in the network
$G = (V, E)$	The graph consists of V and E
T	The aggregation tree
N_v	The set of node v's neighbor nodes
P_v	The parent of node v in the aggregation tree T
C_v	The set of child nodes of v in the aggregation tree T
t_v	The sequence number of the time slot for node v to transmit
k	The convergecast delay
d_v	The degree of node v in G
h_v	The depth of node v in G
$[T, \overline{T}]$	The cut-set of links between T and $\overline{T}$
R	The network radius
Δ	The maximum node degree
l	The side length of the square area for sensor deployment
r	The nodes' transmission radius
D	$n\pi r^2/l^2$
L	l/r

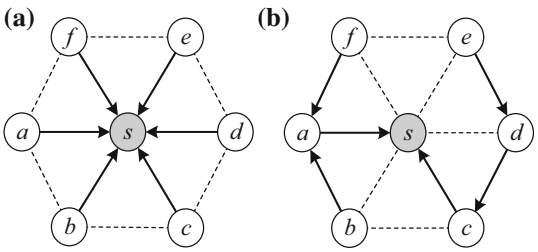

Fig. 2 An example of aggregation trees constructed by BSPT and MLST. **a** BSPT $\max\{|C_v| + h_v : v \in V\} = 6$, **b** MLST $\max\{|C_v| + h_v : v \in V\} = 3$

shown in Fig. 2(a), with all six sensor nodes being the children of the sink node. In this case, the BSPT provides a delay lower bound of 6, which is not an optimum result. As we can see in Fig. 2(b), there is a better tree with delay lower bound of 3. Generally speaking, BSPT will always be bounded by the degree of the sink node, which limits its performance, especially in the situation with sensor nodes densely deployed.

To build an aggregation tree with minimum lower bound, which is not restricted to be an SPT, we propose the Minimum Lower bound Spanning Tree (MLST) algorithm. Inspired by the Prim algorithm [16] for finding a minimum spanning tree, we build the aggregation tree starting from the sink node and add a minimum-cost link to the tree iteration by iteration.

First, we define the cut-set of links, which is denoted by $[S, \overline{S}]$. A cut-set is the set of links pointed from a node in $\overline{S}$ to another node in S, where S is a sub-graph of G and $\overline{S}$ is the sub-graph induced by the nodes out of S. That is to say, a link $(u,v) \in [S, \overline{S}]$, if and only if $u \in \overline{S}$ and $v \in S$.

Then, the cost of a link (u, v) in the cut-set is defined as below.

$$\text{cost}(u, v) = M^2 \cdot (|C_v| + h_v) + M \cdot d_v + d_u \quad (1)$$

where h_v denotes the depth of v in the aggregation tree, d_v denotes the degree of v in the graph G, and M is a large positive number that is greater than the total number of the sensor nodes.

As we can see, the main part of the cost is the sum of the receiver's children number and depth. The maximum of this value in the aggregation tree is just the lower bound of the convergecast delay. The basic idea of the MLST algorithm is to add a new child node to the potential parent node with minimum value of $|C_v| + h_v$ in each iteration, so that the lower bound, $max\{|C_v| + h_v : v \in V\}$, will retain small to a great extent. What is more, minimizing the value of $|C_v| + h_v$ also means that the value of $|C_v|$ or h_v alone is unlikely to be a very large number, so the potential of conflict in the tree and the depth of the tree will be kept low. According to the definition of the link cost, the value of $|C_v| + h_v$ is considered in the first place, in addition to this, the degree of the receiver (d_v) and the degree of the sender (d_u) are also used to screen for the best link. In other words, for those links with the same value of $|C_v| + h_v$, the one with the minimum d_v will be chosen. If there are still multiple links, the one with the minimum d_u will be chosen.

The pseudo-code of the proposed MLST algorithm is presented in Algorithm 1. In the algorithm, we maintain an aggregation tree, T, which initially consists of only the sink node (line 1). In each iteration, we update the cost value for the edges in the cut-set $[T, \overline{T}]$ (line 3–5) and add the link with minimum cost to the tree (line 6–7). The algorithm terminates when all the nodes are in the aggregation tree (line 2).

Algorithm 1. MLST (Minimum Lower bound Spanning Tree)

Input: G=(V,E) and s

Output: Aggregation tree T

1. $T \leftarrow \{s\}$
2. **while** $\overline{T} \neq \varnothing$ **do**
3. **for** each link $(u,v) \in [T, \overline{T}]$ **do**
4. $\text{cost}(u,v) = \text{M}^2 \cdot (|C_v| + h_v) + \text{M} \cdot d_v + d_u$
5. **end-for**
6. $(u_0, v_0) \leftarrow$ find a link with minimum cost in $[T, \overline{T}]$
7. $T \leftarrow T \cup \{(u_0, v_0)\}$
8. **end-while**
9. **return** T

Figure 2(b) shows an example of the aggregation tree constructed by MLST. At first, the sink node will be put in T. Then, we have a cut-set which consists of the six links between the sink node and the other six nodes. Since the cost of those links are equal, we randomly choose one, for example, (a, s). After adding this link to the tree, the next iteration begin. Now the cut-set $[T, \overline{T}]$ contains all the links pointed from node b, c, d, e, f to node a or the sink node. Since the links pointed to a have smaller cost values, the link (b, a) is chosen this time. Then, the links (c, s), (d, c), (f, a), (e, d) are added to the tree successively in the same way. Finally, MLST produces a tree shown in Fig. 2(b) with a delay lower bound of 3, which is only half the value of that for the SPT shown in Fig. 2(a).

4.2 Time slot allocation

The aggregation tree with a small delay lower bound only provide a potential for a shorter delay. For a conflict-free aggregation convergecast scheduling, what affects the delay are not only the structure of the tree but also the link conflict, especially in a densely deployed network. Since the tree structure is considered in the tree construction phase, now we focus on the factor of link conflict in the time slot allocation phase. In the following, we say a time slot is allocated to a node when the node is arranged to send data in the time slot. We also say a time slot is allocated to link, which means the transmission through this link is arranged to be conducted in the time slot.

The idea of our time slot allocation algorithm lies in two aspects. First, the regions in the network with higher node density normally need longer time for data collection due to more link conflict. Those regions can be regarded as the key regions or the bottleneck of the network. Giving priority to the transmissions in those regions is conducive to faster data collection. Second, each time slot needs to be fully utilized by enabling every possible concurrent transmission, so that the delay can be shortened. In a word, the basic idea is *bottleneck first, concurrency most*. To carry out the idea, the Neighbor Degree Ranking (NDR) algorithm which contains a Supplementary Scheduling (SS) scheme is proposed.

In the NDR algorithm, we maintain a sub-tree of T which contains only the unscheduled nodes, i.e. the nodes have not got a time slot. In each iteration, a time slot is allocated to a set of conflict-free transmission links. The leaf nodes of the aggregation tree are considered to be the candidate senders and their parent nodes are considered to be the potential receivers in the time slot to be allocated. Since not all the leaf nodes can be senders in the same time slot due to link conflict, a rank value is defined for every leaf node to determine its priority of transmission. Then, we list the leaf nodes in the order of their rank value. To produce a conflict-free schedule, we go through the ordered list and allocate the time slot to a node v only if the transmission link (v,P_v) does not conflict with any existing links in the same time slot.

The key of the time slot allocation algorithm is to define a proper rank value for each node. As mentioned above, the node in the crowded region is the bottleneck of the network and should be arranged to transmit earlier. As a transmission of a sender can influence and can be influenced by other nodes within its two hop range, we use the sum of the degree of a node's neighbors to characterize the extent of crowding in its ambient region. So the rank of a leaf node v is defined as below.

$$\text{rank}(v) = sum\{d_u : u \in N_v\} \tag{2}$$

where N_v denotes the set of node v's neighbor nodes.

The pseudo-code of our time slot allocation algorithm, NDR, is presented in Algorithm 2. One time slot is arranged in each iteration of the algorithm (line 2–23). The leaf nodes of the tree are checked in descending order of their rank value. If the transmission link from a leaf node to its parent does not conflict with any other child–parent links which have already settled in the same time slot, the concerned time slot is allocated to the leaf node (line 14–19). In the end of each iteration, nodes that have already scheduled are removed from the graph and from the tree (line 21–22). Accordingly, some parameters of the nodes, such as C_v, N_v, and d_u are updated and new leaves

may be generated. The iteration goes on till every sensor node gets a time slot to transmit.

Algorithm 2. NDR (Neighbor Degree Ranking)

Input: $G=(V, E)$ and T

Output: Parent P, time slot S and delay k

1. $i \leftarrow 0$
2. **while** there are still nodes that have not got a time slot **do**
3. $i \leftarrow i+1$
4. $Leaf \leftarrow \varnothing$
5. **for** each node $v \in T$ **do**
6. **if** $|C_v|=0$ **then**
7. $Leaf \leftarrow Leaf \cup \{v\}$
8. **end-if**
9. **end-for**
10. **for** each node $v \in Leaf$ **do**
11. $\text{rank}(v)=\text{sum}\{d_u : u \in N_v\}$
12. **end-for**
13. $Scheduled \leftarrow \varnothing$
14. **for** each node $u \in Leaf$ in decreasing order of rank **do**
15. **if** (u, P_u) does not conflict with any existing link in the i-th slot **then**
16. $t_v \leftarrow i$
17. $Scheduled \leftarrow Scheduled \cup \{u\}$
18. **end-if**
19. **end-for**
20. Apply Algorithm3, *Supplementary Scheduling*
21. $G \leftarrow G \setminus Scheduled$
22. $T \leftarrow T \setminus Scheduled$
23. **end-while**
24. $k \leftarrow i$
25. **return** P, S, and k

Till now, only the child–parent links in the aggregation tree constructed by MLST are considered. However, we find that if we take the links out of the aggregation tree into consideration, it is still possible to arrange more transmissions for some of the unscheduled leaf nodes. The aggregation tree we built previously surely gives us the guidance for choosing proper links, and we considered those links in the first place. But, after that, if other links out of the tree can be used to enable more parallel transmissions in each iteration, the previous tree topology needs not to be a constraint. That is why we propose the Supplementary Scheduling (SS) algorithm. This algorithm is operated after the tree links are checked in every iteration of Algorithm 2 (line 20).

In the tree link checking stage of the NDR algorithm introduced above (i.e., Algorithm 1, line 14–19), we go through the ordered list of the candidate senders, i.e. the leaf nodes, and allocate the concerned time slot to every possible child–parent link if there is no conflict. After that, there are two situations in which additional transmissions out of the tree can be arranged. In the first situation, some unscheduled leaf nodes may have chance to transmit to one of its non-leaf neighbor other than its parent. In the other situation, transmission between one leaf node and another may be possible. Figure 3 shows an example for each of the two situations with a fragment of the network. In the figure, the white nodes a, b, c, and d are leaf nodes, and the grey nodes a', b', c', and d' are their corresponding parent nodes.

Suppose that (c, c') and (d, d') have been scheduled to transmit in the concerned time slot beforehand in the tree link checking stage. Then, the tree links (a, a') and (b, b') cannot conduct transmission in the same time slot because they would conflict with the existing links (c, c') and (d, d'). Nevertheless, as we can see in Fig. 3(a), the transmission from node a to its non-leaf neighbor b', are able to carry out in the same time slot without conflict with either (c, c') or (d, d'). Similarly, in Fig. 3(b), the link from node a to its neighbor b, which is also a leaf node, is able to share the same time slot with (c, c') and (d, d').

The pseudo-code of the Supplementary Scheduling algorithm is shown in Algorithm 3. In the algorithm, the nodes left unscheduled in the tree link checking stage is checked again. If a conflict-free transmission link between an unscheduled leaf node and its non-leaf neighbor (line 1–9) or its leaf neighbor (8–18) is found, additional transmission is arranged in the concerned time slot. As a result, more nodes get scheduled in each iteration of the time slot allocation algorithm, and more concurrent transmissions in each time slot leads to a shorter delay.

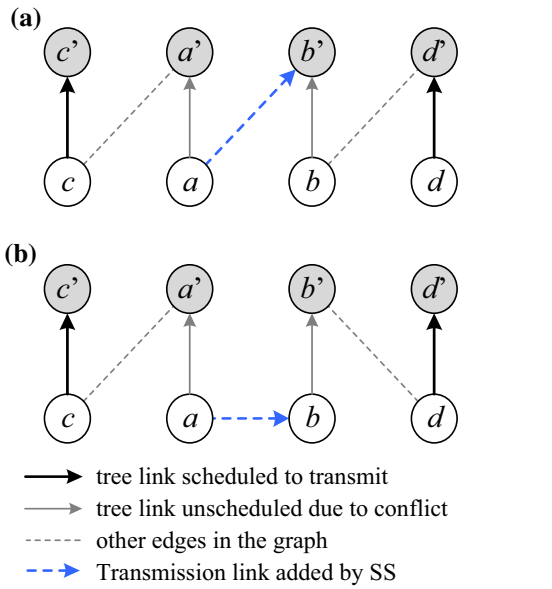

Fig. 3 Example for supplementary scheduling. **a** Transmission between a leaf and its non-leaf neighbor, **b** transmission between a leaf node and its leaf neighbor

Algorithm 3. SS (Supplementary Scheduling)

Input: $G=(V,E)$, T and i

Output: P and S

1. **for** each unscheduled leaf node u **do**
2. **for** each non-leaf neighbor v of u **do**
3. **if** (u,v) does not conflict with any existing transmission link **then**
4. $t_v \leftarrow i,\ P_u \leftarrow v$
5. $Scheduled \leftarrow Scheduled \cup \{u\}$
6. **break**
7. **end-if**
8. **end-for**
9. **end-for**
10. **for** each unscheduled leaf node u **do**
11. **for** each unscheduled leaf neighbor v of u **do**
12. **if** (u,v) does not conflict with any existing transmission link **then**
13. $t_v \leftarrow r,\ P_u \leftarrow v$
14. $Scheduled \leftarrow Scheduled \cup \{u\}$
15. **break**
16. **end-if**
17. **end-for**
18. **end-for**
19. **return** P and S

5 Experiments

In this section, we discuss the performance of our algorithm through simulation experiments.

5.1 Description of the scenario

In the following experiments, the sensor nodes are randomly deployed according to uniform distribution in a square area and the sink node is placed in the center of the area. For the convenience of discussion, all the experiments are done in a connected network. To describe the topological feature of a randomly deployed network, we use the following two parameters.

$$D = n\pi r^2/l^2 \tag{3}$$

$$L = l/r \tag{4}$$

where n is the number of the nodes in the network, r is the transmission radius of the nodes, and l is the side length of the square area. D denotes the node density and is defined by the average number of nodes within a disk area of radius r. L is the ratio of the side length of the square area to the transmission radius.

For all the following experiments, D and L are used to describe the scenario. The transmission radius r is fixed to be $r = 1$, so that L equals the side length l and we also use L to represent the side length below. The range of D is set to be from 5 to 95 and the range of L is set to be from 1 to 8. For each simulation setup, 20 runs of experiment are done and the average result is presented. Notice that the minimum value of D is set to be 5 in order to guarantee the network connectivity and $D = 95$ indicates a very large average degree of the generated network. We set the minimum value of L to be 1, which means the side length of the deployment area is equal to the transmission radius, so the area can almost be covered by the transmission range of each node in the network. When L is set to be its maximum value 8, a large radius of the network (the maximum hop distance from a sensor node to the sink node) can be expected. It can be calculated by (3) that the node number $n = 2$ when (D, L) are set to be their minimum value (5, 1) and $n = 1937$ when (D, L) are set to be their maximum value (95, 8). As we can see, the value of D and L in our experiments covers the scenario with networks that are sparse and dense, small and large. Thus the results get from our experiments can be considered to be applicable to most of the deployment scenarios of the ACS problem.

5.2 Overall performance

The delay produced by our scheduling algorithm under different density and side length is presented in Fig. 4. As we can see in the figure, there is a clear correlation between delay and the value of density and side length. The smaller the density and the side length are, the smaller or sparser the network is, hence the shorter the delay is, and vice versa.

The performance comparison of our algorithm with other algorithms is shown in Fig. 5. WIRES [1], SDA [6] and DAS [5] are chosen for comparison because they are typical and have relatively good performance among the existing work. The figures are plotted with a confidence interval of 95 %. As we can see, our algorithm outperforms the others.

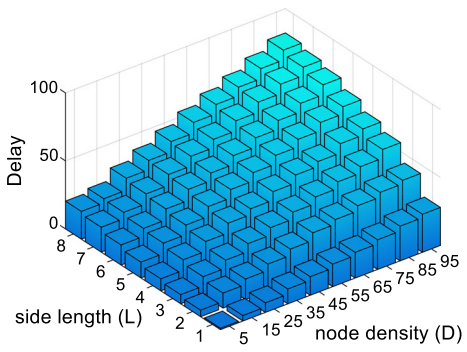

Fig. 4 Average delay under different network deployment scenario

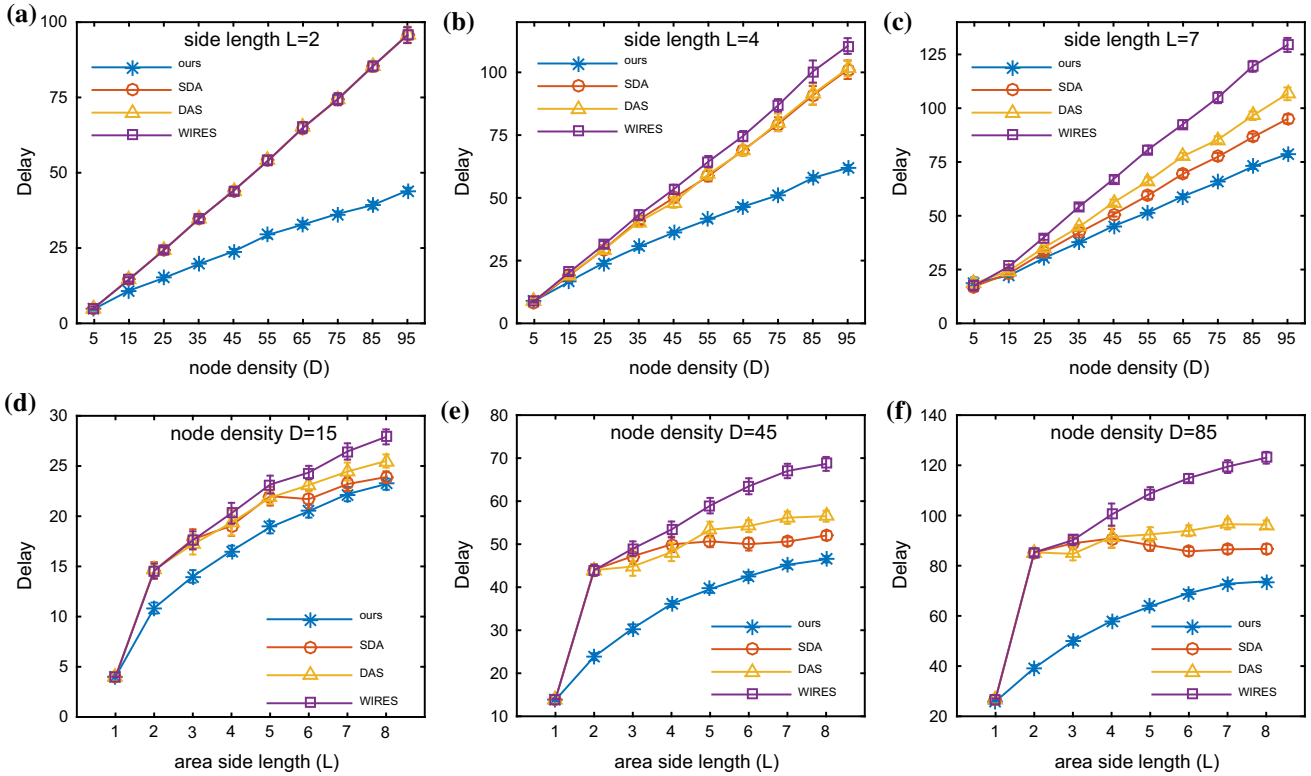

Fig. 5 Performance comparison of scheduling algorithms. Delay versus node density with **a** $L = 2$, **b** $L = 4$, and **c** $L = 7$. Delay versus side length with **d** $D = 15$, **e** $D = 45$, and **f** $D = 85$

Figure 5(a), (b), and (c) show the convergecast delay as a function of the node density of the network, with L fixed at 2, 4 and 7. In these figures, delay almost increases linearly with the increase of D. Our algorithm gains larger comparative advantage with smaller side length. Take $L = 2$ in Fig. 5(a) for a special example, in this scenario, almost all the sensor nodes are neighbors of the sink node, which is located in the center of the area. SDA, DAS, and WIRES just make all the sensor nodes to be the children of the sink node and form an aggregation tree with only one hop, which produces a delay equals to the number of the sensor nodes. Our algorithm produces different tree with smaller lower bound and get a delay 25–53 % shorter than that of the others. $D = 5$ is a special case, too, in which the network is very sparse and there are not many choices of links to build a connected tree. That is why the value of delay are almost the same for all the algorithms when $D = 5$.

Figure 5(d), (e), and (f) show the delay as a function of the side length of the square area, with D fixed at 15, 45 and 85. Comparing these figures, we can tell that our algorithm have larger comparative advantage with larger node density. For example, in Fig. 5(f) with a large density of 85, the delay of our algorithm is 15–54 % smaller than that of the others. As another special case, when $L = l/r = 1$, the topology of the network is almost a complete graph where the nodes are pairwise neighboring, thus the delay is approximate to the number of the sensor nodes for all the algorithms.

In summary, our algorithm outperforms the others by 10 % in most scenarios and the performance is comparatively better in the scenario with smaller side length and lager node density.

5.3 Evaluation of MLST

Since the proposed aggregation convergecast scheduling algorithm includes two independent phases, their performance can be evaluated separately.

By operating the time slot allocation part of SDA, WIRES, and DAS on the tree constructed by our aggregation tree construction algorithm, MLST, we form combined algorithms, which are called MLST-SDA, MLST-WIRES, and MLST-DAS. The performance of MLST can be evaluated by comparing the delay get from the combined algorithms with that get from the original ones. The result is shown in Fig. 6. Figure 6(a)-(d) depict the performance of MLST-SDA and MLST-WIRES and the improvement they made to SDA and WIRES. The

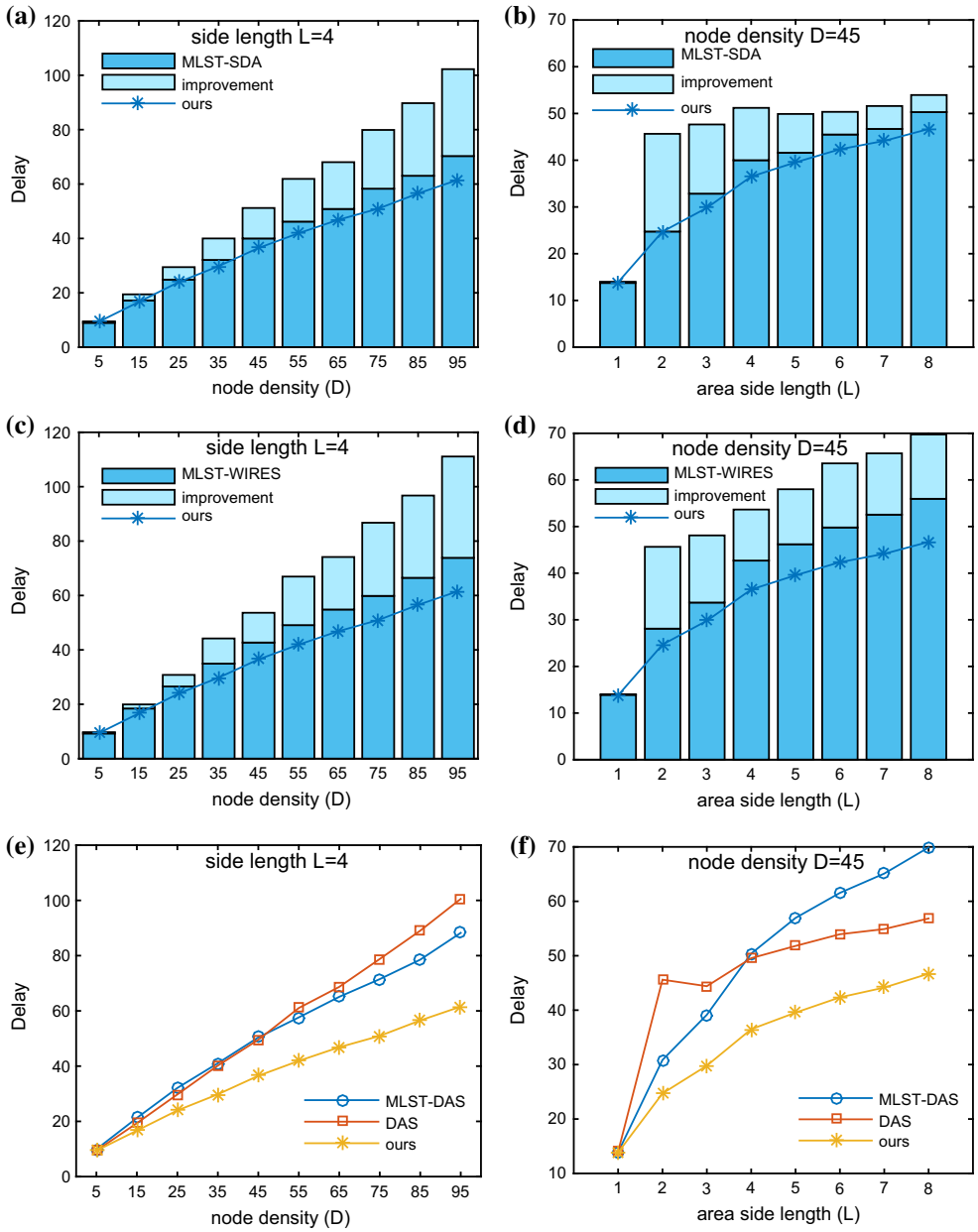

Fig. 6 Performance of SDA, WIRES, and DAS on the tree constructed by MLST. **a**, **b** Performance of MLST-SDA, **c**, **d** performance of MLST-WIRES, **e**, **f** performance of MLST-DAS

performance improvement is achieve by using MLST instead of their own aggregation tree, which implies that our aggregation tree outperforms their trees. At the same time, we can see that the performance of our algorithm is still better than the combined algorithms.

For DAS, the combined algorithm MLST-DAS does not perform well when the side length is relatively long (i.e. when $L > 4$). That is because (i) DAS allocates time slots according to node ID, which is quite randomized without considering parallel transmissions. (ii) MLST introduces a larger depth of the aggregation tree than the original tree of DAS. It can thus be seen, to get a good performance, the time slot allocation phase should cooperate with the tree construction phase.

5.4 Evaluation of NDR

To evaluate our time slot allocation algorithm, NDR, we combine it with the aggregation trees construction phase of other scheduling algorithms and compare the performance of

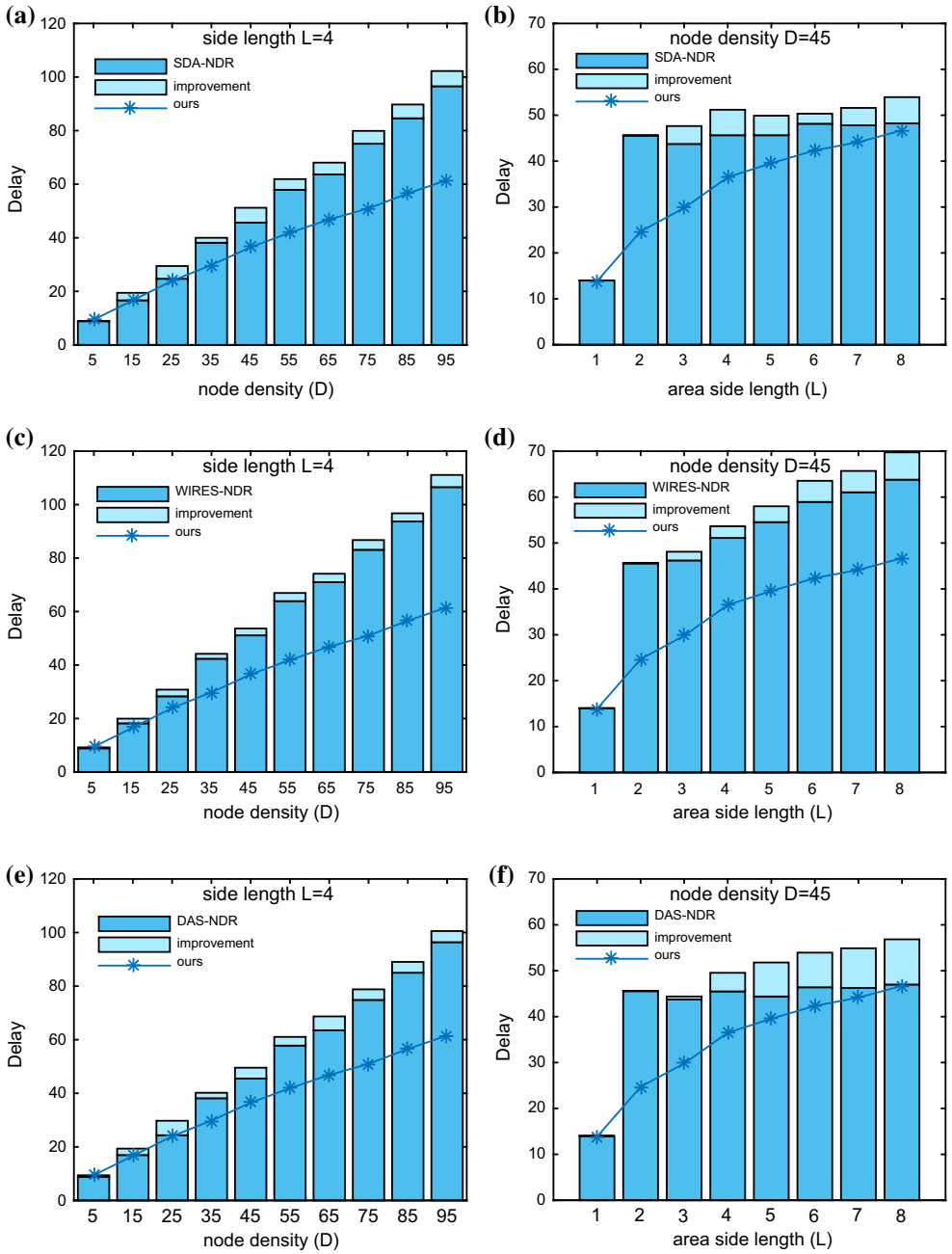

Fig. 7 Performance of NDR on top of aggregation trees constructed by SDA, WIRES, and DAS. **a**, **b** Performance of SDA-NDR, **c**, **d** performance of WIRES-NDR, **e**, **f** performance of DAS-NDR

the combined algorithms with the original ones. As shown in Fig. 7, the combined algorithms produce shorter delay than the original algorithms in most cases, which indicates that our time slot allocation algorithm outperforms the others. What is more, as can be seen, our scheduling algorithm, the combination of MLST and NDR, still outperforms all the other combinations.

Notice that the performance of WIRES, SDA, and DAS in the situation of $L = 2$ is not improved distinctly by the combined algorithm. The reason for this is that the aggregation tree of those algorithms is a one hop tree with all the sensor nodes being the children of the sink node when $L = 2$, as mention in Sect. 5.2 and depicted in Fig. 2(a). Due to the limitation of their tree structure, there is not much chance for improvement.

5.5 Evaluation of the supplementary scheduling scheme

We proposed the Supplementary Scheduling (SS) scheme as a part of our time slot allocation algorithm to use the time slots efficiently. In this section, we discuss the effect of SS on the performance of our scheduling algorithm, by comparing the performance of the scheduling algorithm with and without SS.

The idea of SS is to find supplementary parallel transmission links after the ranking based time slot allocation. For the scheduling algorithms both with and without the SS, the candidate senders in the first time slot are the initial leaf nodes of the aggregation tree. So we compare the number of transmissions in the first time slot, with the same candidate senders, to see how many supplementary transmissions SS can bring about. As shown in Fig. 8, our scheduling algorithm with SS enables about 15–27 % more transmissions than the one without it. That means the SS scheme helps the scheduling algorithm to use the time slot more efficiently.

More parallel transmissions in each time slot will apparently lead to a shorter convergecast delay, actually, the delay can be reduced by about 10–15 % by the supplementary scheduling scheme, as shown in Fig. 9.

5.6 Discussion about the aggregation tree

As mentioned in section IV A, the Lower Bound (L. B. for short) of convergecast delay in a fixed tree is $max\{|C_v| + h_v : v \in V\}$ and the idea of our aggregation tree construction algorithm is to minimize this lower bound. In this section, we discuss the effect of the structural feature, especially the lower bound, of the aggregation tree on the performance of the scheduling algorithm.

To figure out the relevance of the delay lower bound of the aggregation tree and the average performance of the scheduling algorithm, we calculate the delay lower bound of each aggregation tree and compare them with the average delay produced by the corresponding scheduling algorithm. The result is shown in Fig. 10. Notice that since the initial tree of SDA and our algorithm may be changed in the time slot allocation phase, the delay lower bound is calculated on the final aggregation tree. We can see in Fig. 10 that the performance of the scheduling algorithms and the delay lower bound of their aggregation trees have similar trend with the increasing of node density or side length. $L = 2$ is still a special point for SDA, DAS, and WIRES because the sink node takes almost all the other nodes as its children as mentioned before. The value of delay get from DAS and WIRES is quite close to the delay lower bound of their aggregation tree. We can say that, to a large extend, the performance of DAS and WIRES is just limited by their tree structure. Our algorithm, as expected, produces a very small delay lower bound and a delay which is shorter than the delay lower bound of DAS and WIRES. SDA improves its initial SPT tree in the time slot allocation phase and changes it to a tree with smaller lower bound. That makes its performance relatively better than DAS and WIRES in many situations.

To analyze how the tree topology affects the convergecast delay, we plot an instance of the network and three kinds of aggregation tree on top of it under the setting of $L = 4$ and $D = 45$, as shown in Fig. 11. The values of the delay get from WIRES, DAS, and our algorithm on this network are 51, 47, and 35, respectively. These values are typical because they are approximate to the average delay under the setting of $L = 4$ and $D = 45$, which are 53.4, 48.15, and 36.1, respectively.

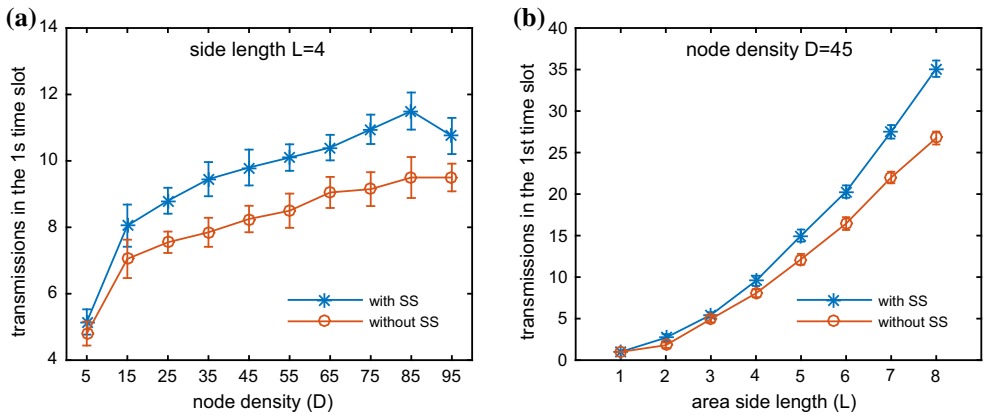

Fig. 8 Number of transmissions in the first time slot of our scheduling algorithm with and without SS. **a** Number of transmissions versus node density with $L = 4$, **b** number of transmissions versus side length with $D = 45$

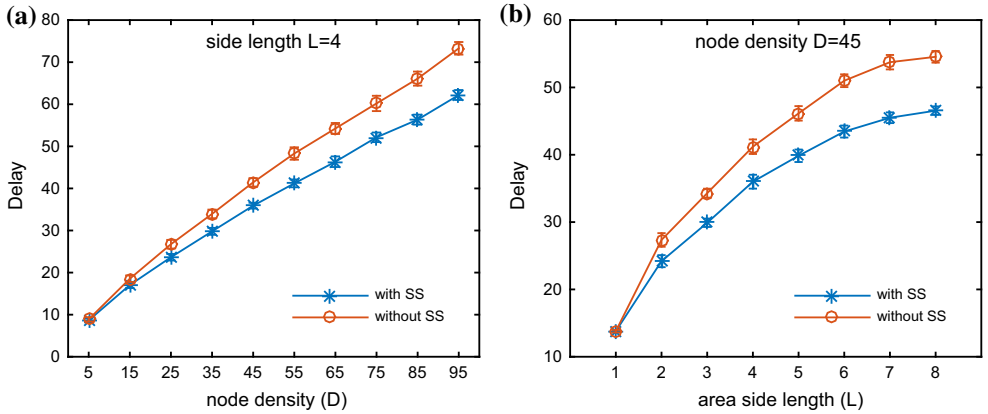

Fig. 9 Delay get from our scheduling algorithm with and without SS. **a** Delay versus node density with $L = 4$, **b** delay versus side length with $D = 45$

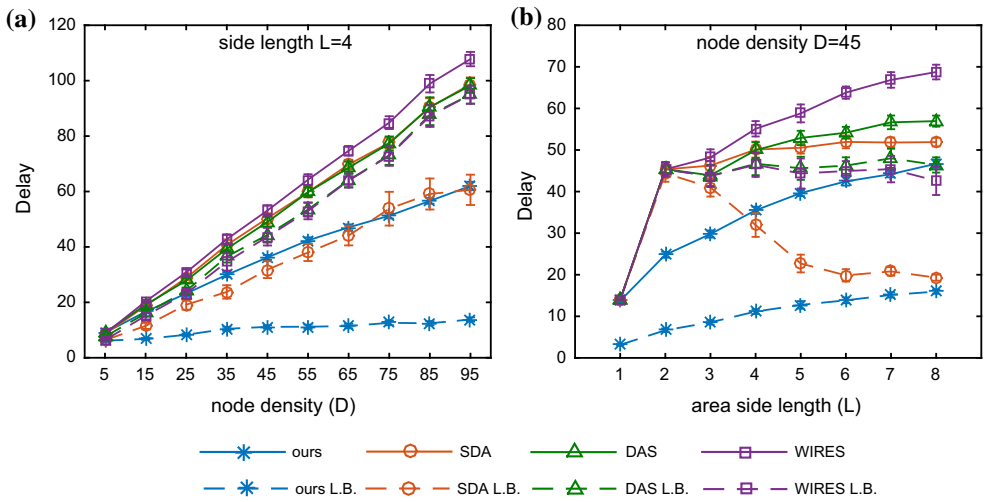

Fig. 10 Performance and delay lower bound of the scheduling algorithms with **a** $L = 4$ and **b** $D = 45$

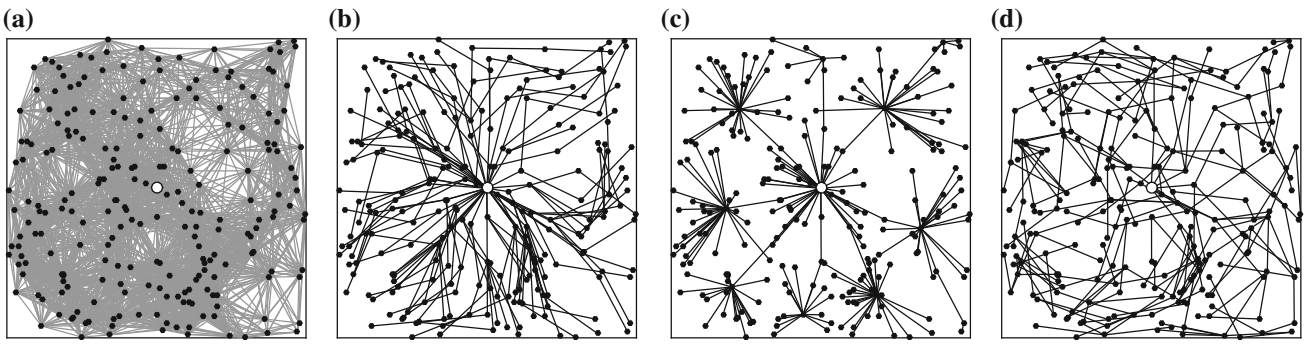

Fig. 11 Instance of network and aggregation trees. **a** Network, **b** SPT of WIRES delay = 51, **c** CDS of DAS delay = 47, **d** our tree delay = 35

As we can see in Fig. 11, the aggregation trees formed by different algorithms have very different characteristics. As shown in Fig. 11(b), all the nodes in the SPT transmit to the sink node through a shortest path, so all the data can reach the sink node in minimum hops. However, with the transmission links all pointing to the sink node, many

conflict may be introduced, especially when the node density is high. CDS based algorithms generate clusters with dominating nodes being the cluster heads, as we can see in Fig. 11(c). Data is firstly transmitted to the dominating nodes and then to the sink node. The problem is that the dominating nodes might be those of very large degree. In this case, the dominating nodes need to wait for its children to transmit for a long time.

In both SPT and CDS based trees, the links are intensively connected to a few nodes, especially the sink node. Instead, the links are more dispersedly connected in the tree formed by our scheduling algorithm as shown in Fig. 11(d), which produces a shorter delay as well as a smaller delay lower bound. This explains partly the advantage of our scheduling algorithm upon the others.

6 Conclusion

In this paper we investigate the aggregation convergecast scheduling problem in wireless sensor networks and propose an algorithm including a minimum lower bound spanning tree construction phase and a time slot allocation phase based on neighbor degree ranking and supplementary scheduling scheme. Comparative experiments show that our scheduling algorithm outperforms the others by more than 10 % in most scenarios. For the networks with relatively high node density and small network radius ($D >= 85$ and $2 <= H <= 4$), the performance improvement can be more than 50 %, which implies that our algorithm has more advantage compared with the others in this kind of networks. Further experiments are done for the evaluation of the scheduling algorithm's two independent phases. The results indicate that not only the overall performance of our scheduling algorithm, but also the separate performance of its two phases are better than the existing work. Discussion of the proposed algorithm show that the advantage of it is gained by a dispersedly connected tree topology and a well-designed time slot allocation algorithm with supplementary scheduling scheme.

Acknowledgments This research was supported by the National Natural Science Foundation of China under Grant 61174179 and 60874079 and the Science and Technology Development Project of China Railway Corporation under Grant 2015Z005-D.

References

1. Malhotra, B., Nikolaidis, I., & Nascimento, M. A. (2011). Aggregation convergecast scheduling in wireless sensor networks. *Wireless Networks, 17*(2), 319–335.
2. Hong, L., Huixiang, T., Huadong, M., & Das, S. K. (2011). Data fusion with desired reliability in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems, 22*(3), 501–513.
3. Srisooksai, T., Keamarungsi, K., Lamsrichan, P., & Araki, K. (2012). Practical data compression in wireless sensor networks: A survey. *Journal of Network and Computer Applications, 35*(1), 37–59.
4. Madden, S., Franklin, M. J., Hellerstein, J. M., Wei, H., & Usenix, U. (2002). TAG: A tiny aggregation service for ad-hoc sensor networks. In *Usenix Association Proceedings of the Fifth Symposium on Operating Systems Design and Implementation* (pp. 131–146).
5. Yu, B., Li, J., Li, Y., & Ieee, (2009). Distributed data aggregation scheduling in wireless sensor networks. *Proceedings of IEEE Conference on Computer Communications, 1–5*, 2159–2167.
6. Chen, X. J., Hu, X. D., & Zhu, J. M. (2005). Minimum data aggregation time problem in wireless sensor networks. *Proceedings of Mobile Ad-Hoc and Sensor Networks, 3794*, 133–142.
7. de Souza, E., & Nikolaidis, I. (2013). An exploration of aggregation convergecast scheduling. *Ad Hoc Networks, 11*(8), 2391–2407.
8. Incel, O. D., Ghosh, A., Krishnamachari, B., & Chintalapudi, K. (2012). Fast data collection in tree-based wireless sensor networks. *IEEE Transactions on Mobile Computing, 11*(1), 86–99.
9. Harvey, N. J. A., Ladner, R. E., Lovász, L., & Tamir, T. (2006). Semi-matchings for bipartite graphs and load balancing. *Journal of Algorithms, 59*(1), 53–78.
10. Wan, P. J., Alzoubi, K. M., & Frieder, O. (2004). Distributed construction of connected dominating set in wireless ad hoc networks. *Mobile Networks and Applications, 9*(2), 141–149.
11. Wan, P.-J., Huang, S. C. H., Wang, L., Wan, Z., & Jia, X. (2009). Minimum-latency aggregation scheduling in multihop wireless networks. In *Proceedings of the Tenth Acm International Symposium on Mobile Ad Hoc Networking and Computing* (pp. 185–193).
12. Li, Y., Guo, L., & Prasad, S. K. (2010). An energy-efficient distributed algorithm for minimum-latency aggregation scheduling in wireless sensor networks. In *Proceedings of 2010 international conference on distributed computing systems ICDCS 2010* (pp. 827–836).
13. Xu, X., Li, X.-Y., Mao, X., Tang, S., & Wang, S. (2011). A delay-efficient algorithm for data aggregation in multihop wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems, 22*(1), 163–175.
14. Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of Standards B, 71*(4), 233–240.
15. Gupta, P., & Kumar, P. R. (2000). The capacity of wireless networks. *Information Theory, IEEE Transactions on, 46*(2), 388–404.
16. Prim, R. C. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal, 36*(6), 1389–1401.

Cheng Pan is currently pursuing the Ph.D. degree with the School of Electrical Engineering, Beijing Jiaotong University, Beijing, China. His research interests include sensor deployment and data collection in wireless sensor networks.

Hesheng Zhang received the B.S. and M.S. degrees from Northern Jiaotong University, Beijing, China, in 1992 and 1995, respectively, and the Ph.D. degree from Tsinghua University, Beijing, China, in 2006. He is currently a Professor with Beijing Jiaotong University. His research interests include sensor network and sensor fusion, field bus, and control network.