

Distributed Data Aggregation Scheduling in Wireless Sensor Networks

Bo Yu
School of Computer Science and
Technology, Harbin Institute of
Technology, China
Email: bo_yu@hit.edu.cn

Jianzhong Li
School of Computer Science and
Technology, Harbin Institute of
Technology, China
Email: lijzh@hit.edu.cn

Yingshu Li
Department of Computer Science, Georgia
State University
Email: yli@cs.gsu.edu

Abstract—Data aggregation is an essential operation in wireless sensor network applications. This paper focuses on the data aggregation scheduling problem. Based on *maximal independent sets*, a distributed algorithm to generate a collision-free schedule for data aggregation in wireless sensor networks is proposed. The time latency of the aggregation schedule generated by the proposed algorithm is minimized using a greedy strategy. The latency bound of the schedule is $24D + 6\Delta + 16$, where D is the network diameter and Δ is the maximum node degree. The previous data aggregation algorithm with least latency has the latency bound $(\Delta - 1)R$, where R is the network radius. Thus in our algorithm Δ contributes to an additive factor instead of a multiplicative factor, which is a significant improvement. To the best of our knowledge, the proposed algorithm is the first distributed algorithm for data aggregation scheduling. This paper also proposes an adaptive strategy for updating the schedule when nodes fail or new nodes join in a network. The analysis and simulation results show that the proposed algorithm outperforms other aggregation scheduling algorithms.

I. INTRODUCTION

In many sensor network applications, such as environmental monitoring, spatial exploration and battlefield surveillance, sensed data is aggregated and transmitted to the sinks for analysis. Thus, in-network data aggregation [1] becomes an important technique in wireless sensor networks and has been well studied in recent years. Unfortunately, previous researches on in-network aggregation seldom consider the collision problem but leave it to the MAC layer. Solving this problem in MAC layer incurs a large amount of energy consumption and time latency during aggregation.

Recently, a few researchers begin to consider the collision problem for data aggregation and try to construct a feasible schedule to eliminate collisions during aggregation. Huang *et.al.* proposed a scheduling algorithm with the latency bound $23R + \Delta - 18$ [2], where Δ is the maximum node degree and R is the network radius. However, the schedules generated by this algorithm are not collision-free in many cases. Hence collisions occur during aggregation and data cannot be aggregated to the sink within the expected latency bound in many cases. The details about the problem of the algorithm will be discussed in Section V-C. Chen *et.al.* proposed an algorithm to generate a collision-free schedule with a latency bound of $(\Delta - 1)R$ [3]. Though this algorithm has a larger time

latency bound than the one in [2], it is still the state-of-the-art algorithm for generating collision-free schedules.

To the best of our knowledge, all the previous algorithms for generating collision-free schedules are centralized which require the sink to compute the schedule and disseminate it to the sensors. Once all the sensor nodes receive the schedule, they work according to the schedule. Since topology changes often occur in sensor networks such as node failures, the sink has to gather new topology information from the network, recompute a schedule and disseminate it frequently. These processes consume lots of energy, which makes centralized algorithms inefficient.

In summary, there are two problems in previous researches. One is that the time latencies of the existing scheduling algorithms are still high. The other one is that all the existing algorithms are centralized, which are inherently inefficient.

To solve the two problems, this paper proposes a distributed scheduling algorithm generating collision-free schedules with the latency bound of $24D + 6\Delta + 16$, where D is the network diameter and Δ is the maximum node degree. Since $D \leq 2R$ for any arbitrary network graph and Δ is an additive factor instead of a multiplicative one as in [3], this paper makes a significant improvement.

The rest of the paper is organized as follows. Section II outlines the related work. Section III defines the problem. Our distributed scheduling algorithm is presented in Section IV and is analyzed in Section V. An adaptive method for aggregation scheduling is proposed in Section VI. Section VII presents the simulation results. Section VIII concludes the paper.

II. RELATED WORK

Data aggregation in sensor networks has been well studied in recent years [4]–[7]. In-network aggregation means computing and transmitting partially aggregated data rather than transmitting raw data in networks to reduce the energy consumption [1]. There are a vast amount of extant work on in-network aggregation in the literature [8], [9]. Suppression scheme and model-driven approach were proposed in [10], [11] towards reducing communication cost. The tradeoff between energy consumption and time latency was considered in [12]. A heuristic algorithm for both broadcast and data aggregation was designed in [13]. Another heuristic algorithm

for data aggregation was proposed in [14] aiming to reduce time latency and energy consumption. [15] proposed a randomized and distributed algorithm for aggregation in a n -node sensor network with an expected latency of $O(\log n)$. In their model, there are two assumptions. One is that each sensor node has the capability of detecting whether a collision occurs after transmitting data. Another one is that sensor nodes can adjust their transmission range without any limitation. These assumptions pose some challenging issues for hardware design and the latter assumption is almost impossible when the network scale is very large. A collision-free scheduling method for data collection is proposed in [16] aiming at optimizing energy consumption and reliability. All these work did not discuss the minimal-time aggregation scheduling problem.

The most related work to aggregation scheduling is as follows. The minimum data aggregation time problem was proved NP-hard and a $(\Delta - 1)$ -approximation algorithm was proposed in [3], where Δ is the maximum degree of the network graph. Another aggregation scheduling algorithm was proposed in [2], which has a latency bound of $23R + \Delta - 18$, where R is the network radius and Δ is the maximum degree. Unfortunately, there are some mistakes in their algorithm and the generated schedules are not collision-free in many cases. We discuss their algorithm in Section V-C. All these algorithms mentioned above are centralized. In many cases centralized algorithms are not practical, especially when the network topology often changes in a large sensor network.

It is worth to note that the only distributed algorithms for convergecast scheduling were proposed in [17], [18]. However, this work focused on the scheduling problem for data collection in sensor networks, but not data aggregation. In data collection, the sink must receive N packets from all the nodes since data cannot be merged, where N is the number of sensor nodes in the network. Thus the lower bound of latency is N . The upper bound of the time latency of this algorithm is $\max(3n_k - 1, N)$, where n_k is the number of nodes in the largest one-hop-subtree. This result has much higher latency than our algorithm because it solves the collection scheduling but not aggregation scheduling.

In summary, there have been lots of work on in-network aggregation and some work on centralized aggregation scheduling, but no work on distributed aggregation scheduling and the existing aggregation scheduling algorithms still have high latencies.

III. PROBLEM DEFINITION

The network model in this paper is as follows. The network consists of n sensor nodes and one base station that is also called a sink. Each sensor node can send or receive data to or from all directions. The transmission/reception area of each sensor node is roughly a disk centered at the node. We also assume that all nodes have the same transmission range for simplicity. Let $G(V, E)$ be the topology graph of a network. V is the set of all the nodes in the network and E is the edge set of G . An edge (u, v) is in E if and only if u is in the transmission area of v . A node can either send or receive data

at a time and it can receive a data packet correctly when it hears only this packet at that moment. If a node hears two or more messages at the same time, it cannot receive any of them correctly due to the interference. At this moment a collision occurs. In aggregation scheduling, the output schedule should be collision-free.

Let $A, B \subset V$ and $A \cap B = \emptyset$. We say data is *aggregated* from A to B in one time slot if all the nodes in A transmit data in one time slot simultaneously and all the data are received collision-free by some node in B , and A is called a sender set. An aggregation schedule can be defined as a sequence of sender sets S_1, S_2, \dots, S_l , where S_1 aggregates data to $V - S_1$ in time slot 1, S_2 aggregates data to $V - (S_1 \cup S_2)$ in time slot 2 and so on. In the last time slot l , data is aggregated from S_l to $V - \bigcup_{i=1}^l S_i$. As long as $\bigcup_{i=1}^l S_i = V - \{r\}$ where r denotes the sink, data can be aggregated to the sink in l time slots, l is the aggregation latency. We define a data aggregation schedule as follows.

A data aggregation schedule is a sequence of sender sets S_1, S_2, \dots, S_l satisfying the following conditions:

- 1) $S_i \cap S_j = \emptyset, \forall i \neq j$;
- 2) $\bigcup_{i=1}^l S_i = V - \{r\}$;
- 3) Data are aggregated from S_k to $V - \bigcup_{i=1}^k S_i$ at time slot k , for all $k = 1, 2, \dots, l$ and all the data are aggregated to the sink r in l time slots.

The distributed aggregation scheduling problem is to find a data aggregation schedule S_1, S_2, \dots, S_l in a distributed way such that l is minimized and this problem is proved to be NP-hard in [3]. This paper proposes an approximate distributed algorithm with latency $24D + 6\Delta + 16$, where D is the network diameter and Δ is the maximum node degree.

IV. DISTRIBUTED AGGREGATION SCHEDULING ALGORITHM

Our distributed aggregation scheduling algorithm, named DAS, consists of two phases. One is to construct a distributed aggregation tree. Another one is to perform the distributed aggregation scheduling. We adopt an existing method for the first phase and the second phase is the key part of our algorithm. We present these two phases in the following two subsections. At the end of this section, an example is presented to show the scheduling process of DAS. Note that the schedule generated by DAS is collision-free thus data aggregation can be done even without MAC protocol. However, since our scheduling algorithm is a distributed one which requires communications of nodes, we adopt an underneath MAC protocol through our algorithm. The MAC protocol can be any existing MAC protocol, such as TDMA.

A. Distributed Aggregation Tree Construction

As DAS is aggregation-tree-based, an aggregation tree is constructed in a distributed way using an existing approach [19] in the first phase of DAS. As a connected dominating set (CDS) can behave as the virtual backbone of a sensor network, a CDS is employed in this phase. A distributed approach of constructing a CDS has been proposed by Wan *et al.* in [19].

In their algorithm, an MIS of the network is constructed first and then a dominating tree is constructed concatenating MIS nodes and the other nodes. The root of the dominating tree is one neighbor of the sink. This dominating tree can be used as the aggregation tree in DAS with a little modification as follows. We change the root of the dominating tree to the sink, and the original root of the dominating tree to be one of the sink's children.

We claim that the constructed aggregation tree must be of the form shown in Figure 1 according to the algorithm in [19]. All the black nodes form an MIS. The rest nodes are gray and white. The white nodes are leaves in the aggregation tree and each gray node plays the role of connecting two black nodes. The root of the aggregation tree is a black node and all its neighbors are its children that can only be white or gray nodes. For each gray node, its children must be black nodes. For each black node, its children must be either white or gray.

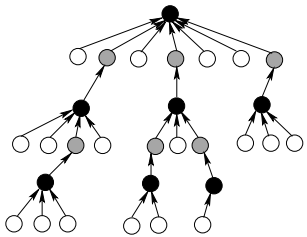


Fig. 1. The form of aggregation tree

B. Distributed Aggregation Scheduling

A schedule of a node u in a sensor network is a sending time slot (sending slot for short) for u to send data during the slot. The second phase of DAS is to determine the schedules for all the nodes in a sensor network in a distributed manner to solve the distributed aggregation scheduling problem. In the subsection, we first discuss the competitors of a node, then present the framework of the distributed aggregation scheduling algorithm (SCHDL for short), and finally give the algorithm and prove its correctness.

Definition 1: For any node u , a node is called a *competitor* of u if it cannot send data while u is sending data due to the collision. The set of all the competitors of u is called u 's *competitor set*.

Given a node u , its competitor set can be determined by the following proposition.

Proposition 1: For each node u , let $p(u)$ be u 's parent. Let $N(u)$ be the set of u 's 1-hop neighbors except $p(u)$ and let $Ch(u)$ be the set of u 's children, then u 's *competitor set* = $N(p(u)) \cup (\bigcup_{v \in N(u) \setminus Ch(u)} Ch(v)) \setminus \{u, p(u)\}$. ■

Proof: Suppose u is sending data to its parent p and a collision occurs. The collision must occur at node p or at another node w . Both cases are shown in Figure 2. In the first case, p must have received two or more messages, that is, there exists some node v sending data at the same time as u , and v is in p 's transmission range. Thus it is true that p 's 1-hop neighbors are u 's *competitors*. In the second case, node w must have received two or more messages. One is

from u and another one is from some child of w . Similarly it can be concluded that u 's 1-hop neighbors' children are u 's *competitors*. Thus we conclude that for each node u , u 's parent p 's 1-hop neighbors and u 's 1-hop neighbors' children constitute u 's *competitor set*. Among these nodes p 's parent t can be excluded because t will not send data before all its descendants sending data to it. Also u 's 1-hop neighbors can exclude u 's parent p and u 's children. This is because p 's children are already in p 's 1-hop neighbors and u 's grandchildren cannot send data at the same time as u . Therefore, Proposition 1 is true. ■

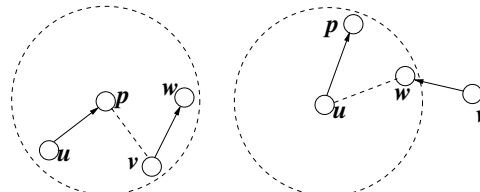


Fig. 2. Two cases of a collision

The *competitor set* of each node can be simply obtained by broadcasting after the aggregation tree is constructed.

In the rest of the paper, we assume that each node u maintains the following information.

- u 's unique ID.
- u 's *competitor set*, denoted by $M(u)$.
- u 's *earliest possible sending slot*, K , initialized to 1.
- $R(u) = R_1(u) \cup R_2(u)$, where $R_1(u) = \{v | v \in M(u) \text{ and is ready to make schedule}\}$, $R_2(u) = \{v | v \in M(u) \text{ and finished scheduling}\}$, and $R_1(u)$ and $R_2(u)$ are both initialized to a null set. For $\forall v \in R_1(u)$, v 's *earliest possible sending slot* is maintained and for $\forall w \in R_2(u)$, w 's schedule is maintained.
- The number x of u 's children that have not scheduled, which is initialized to the number of u 's children.

Now we discuss the idea of the SCHDL algorithm. To determine the schedules for all the nodes in a sensor network based on the competitor sets obtained by Proposition 1, each node is set to one state of NOT-READY (NRY), READY (RY), WAIT0, WAIT1, SCHEDULE-COMPLETED (SC) and SLEEP at every time instant. During the scheduling, each node works according to the automaton shown in Figure 3. Initially each non-leaf node is in NRY state and each leaf node is in RY state. Once all the children of a node finish their scheduling, the node turns into RY state. When a node u comes into RY state, u sends a MARK message containing u 's ID and K to request for the feedback messages from all its *competitors* and turns into state WAIT0. WAIT0 is a state where a node waits for feedback messages. When a node receives a MARK message, it sends a message back including its state and *earliest possible sending slot*. When a node u in WAIT0 gets all feedback messages, it checks if its ID is larger than IDs of all the nodes in $R_1(u)$. If so, u calls a procedure to fix its schedule and send a SCH-COMplete message consisting of u 's ID and K to all nodes in $R_1(u)$, else it goes to WAIT1 state, waiting for scheduling later. A node in RY state is called a ready node. A node u in WAIT1 state must

wait for being scheduled until all the ready nodes with larger IDs are scheduled. Upon receiving a new SCH-COMplete message, a node in WAIT1 checks if its ID is larger than IDs of all the nodes in $R_1(u)$. If so, it calls a procedure to fix its schedule, else it keeps waiting in WAIT1 state. When the root comes into RY state, the algorithm ends and all the nodes complete their schedules. Then during aggregation, each node gets awake in its sending slot to send data and in its children's sending slots to receive data from its children.

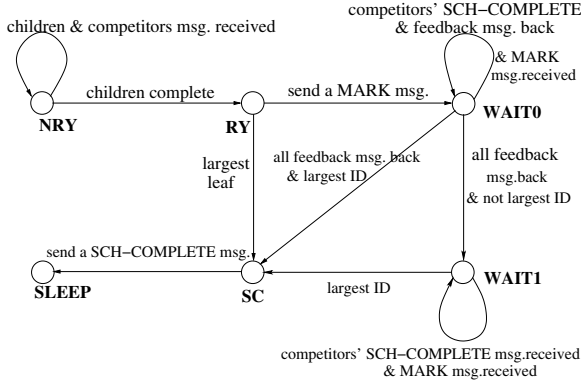


Fig. 3. Automaton of node behavior

The description of the SCHDL algorithm is as follows.

- 1) Initially all the leaf nodes are set to state RY and the rest nodes are set to state NRY.
- 2) For each leaf u , if its ID is larger than IDs of all the nodes in $M(u) \cap \{leaves\}$, then multicast a SCH-COMplete msg. to $M(u) \cup \{p\}$, and u turns to sleep, else send a MARK msg. to $M(u)$ and turn to state WAIT0.
- 3) For each node u , upon receiving a SCH-COMplete msg. from node v ,
 - a) if u is in state WAIT0, record the msg. as a feedback message and $R_2(u) \leftarrow R_2(u) \cup \{v\}$.
 - b) if u is in state NRY, do
 - i) if $v \in M(u)$, do $M(u) \leftarrow M(u) \setminus \{v\}$, $R_2(u) \leftarrow R_2(u) \cup \{v\}$.
 - ii) if u is parent of v , do $x \leftarrow x-1$, $K \leftarrow \max\{K, \text{msg}.K + 1\}$.
 - iii) if $x = 0$, do
 - A) if $M(u)$ is empty, send a SCH-COMplete msg. to its parent p and turns to sleep;
 - B) else send a MARK msg. to $M(u)$ and turn to state WAIT0.
 - c) if u is in state WAIT1, do $R_1(u) \leftarrow R_1(u) \cup \{v\}$, $R_2(u) \leftarrow R_2(u) \cup \{v\}$ and check if u 's ID is larger than IDs of all the nodes in $R_1(u)$. If so, call FIX-SCH(u); else u keeps in state WAIT1.
- 4) For a node u , upon receiving a MARK msg. from node v ,
 - a) $R_1(u) \leftarrow R_1(u) \cup \{v\}$;
 - b) if $x = 0$, send a READY msg. containing K back to v ;
 - c) else send a NOT-READY msg. back to v .
- 5) For a node u , upon receiving a READY or NOT-READY msg. from node v ,

- a) record the message as feedback.
- b) if it is a READY msg., do $R_1(u) \leftarrow R_1(u) \cup \{v\}$.
- c) if all feedback messages are received, check if u 's ID is larger than IDs of all the nodes in $R_1(u)$. If so, call FIX-SCH(u); else turn to state WAIT1.

- 6) If the root w comes into RY state, SCHDL ends.

FIX-SCH(u) is the procedure to fix u 's schedule. It assigns the first available sending slot after u 's earliest possible sending slot K to u using a greedy strategy. The pseudocode of FIX-SCH(u) is as follows.

Algorithm 1 FIX-SCH(u)

```

1:  $sch \leftarrow u.K$ 
2: while (true) do
3:   if  $\exists$  node  $v \in R_2(u)$ ,  $v.schedule = sch$  then
4:      $sch \leftarrow sch + 1$ 
5:   else
6:      $u.schedule \leftarrow sch$ 
7:     Send a SCH-COMplete msg. and  $u.schedule$  to  $M(u) \cup \{p\}$ , then let  $u$  go to sleep and return;
8:   end if
9: end while
    
```

The following theorem shows the correctness of SCHDL. Its complete proof is available in [20].

Theorem 2: SCHDL generates a collision-free aggregation schedule in finite time.

Figure 4 illustrates the process of SCHDL for a small sensor network. Figure 4(a) shows the topology of the network. The IDs of the nodes are labeled beside the nodes. Node 7 is the sink. The solid lines represent edges in the aggregation tree and the dotted lines represent the other edges in the graph. Figure 4(a)-(h) show the states of the nodes in the network while time increases. A node is black if it has already completed scheduling at that time. The gray nodes are ready to be scheduled and the white nodes are not. The schedule of each node is labeled beside it with brackets.

At first, all the leaves are ready and are in RY state. They send MARK messages to their competitor sets and turn into WAIT0 state. Only node 11, 12 and 14 set their schedules to time slot 1 and turn into SLEEP state while the other leaves turn to WAIT1 state. Consider node 11, its competitor set is $\{16, 5, 8, 6\}$. According to SCHDL, node 11 finds its ID is larger than all the leaf nodes in its competitor set, which are node 5 and 8. It sets its schedule to time slot 1, sends a SCH-COMplete message to node 16, 5, 8, 6 and its parent node 1, and goes to sleep.

Next, nodes 3, 4, 5, 8 and 9 are ready and in WAIT1 state. Nodes 3, 5, 8 and 9 receive SCH-COMplete messages, respectively from node set $\{12\}$, $\{11, 14\}$, $\{11, 12\}$ and $\{14\}$. Upon receiving these messages, they check if their IDs are larger than all the ready nodes in their competitor sets. Nodes 8, 9 find they are and call FIX-SCH. They both set their schedules to time slot 2 and send out SCH-COMplete messages. Nodes 3, 5 still stay in WAIT1 state. As node 6 has received both SCH-COMplete messages from its children, it turns from NRY state into RY state.

In this way, all the nodes complete their scheduling when SCHDL ends. The schedule is shown in Figure 4(h). Thus node 7, the sink, can gather all the data in 7 time slots.

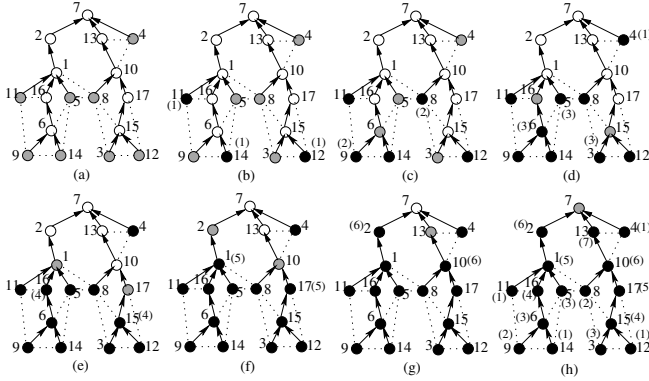


Fig. 4. An example of DAS scheduling

V. PERFORMANCE ANALYSIS

We analyze the performance of DAS from three aspects, i.e. time latency, time cost and communication cost. At the end of the section, an example is presented to show the collision problem in Huang *et.al.*'s algorithm [2].

A. Time Latency Analysis

Lemma 1: For any node u , if u 's schedule is set to t by SCHDL and its *earliest possible sending slot* is K when it is ready in SCHDL, then $t \leq K + |M(u)|$.

Proof: Suppose a node u 's *earliest possible sending slot* is K when it is ready in SCHDL and its schedule is set to t by SCHDL. According to the SCHDL algorithm, when $\text{FIX-SCH}(u)$ is called, the first available sending slot after K is assigned to u 's schedule. Referring to FIX-SCH , a time slot is available for u if and only if it is not equal to the schedule of any node in $R_2(u)$. Thus u 's schedule $t \leq K + |R_2(u)|$ since the case for u 's latest schedule is that the schedules of the nodes in $R_2(u)$ are $K, K + 1, \dots, K + |R_2(u)| - 1$. As $R_2(u)$ is a subset of $M(u)$, $t \leq K + |M(u)|$. ■

Lemma 1 guarantees that the schedule of any node is no later than the *earliest possible sending slot* of the node when it is ready plus the number of its *competitors*.

Now we analyze the latency bound of DAS. Since we have made the assumption that all the nodes have the same transmission range in Section 3, we can normalize their radius to 1 and model the network as a unit disk graph (UDG) [21]. First, we partition the set of the nodes in the aggregation tree created by the first phase of DAS into layers and compute the latest schedule of the nodes in each layer as shown in Figure 5. Suppose the schedules of all nodes have finished by DAS. Let t_i be the latest schedule of the nodes in layer i , $i = 1, 2, \dots, m$, where m is the number of the deepest layer. It is obvious that the whole latency $T = t_1$.

To compute t_1 , two cases need to be considered since all the nodes in layer 1 are white or gray nodes as shown in Figure 5. Case one is that a gray node, say node u , has the schedule t_1 .

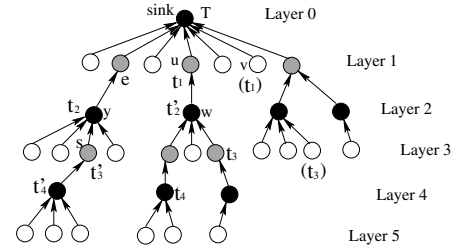


Fig. 5. Analysis on an aggregation tree

Case two is that a white node, say node v , has the schedule t_1 .

In case one, suppose a child w of u has schedule t'_2 while y has the latest schedule t_2 in layer 2. Thus, $t'_2 \leq t_2$, and each gray node in layer 1 has received the data from all of its children at time slot $t_2 + 1$ during aggregation. There are two situations again at $t_2 + 1$. One is that all the white nodes have sent their data to the sink, the other is the opposite.

- In the former situation, the gray nodes compete with only gray nodes. Each gray node must have at least one black child because the tree construction phase chooses gray nodes to interconnect black nodes. For this reason, the number of gray nodes in layer 1 is not greater than the number of black nodes in D_2 , a disk of radius 2 centered at the sink. According to a conclusion in [2], each black node can have at most 20 2-hop neighbors. Starting from time slot $t_2 + 1$, all the gray nodes in layer 1 can send data to the sink in at most 20 time slots according to Lemma 1 and the fact of there being at most 20 gray nodes in competing. Therefore, $t_1 \leq t_2 + 20$ in this situation.
- In the latter situation, some white nodes have not sent their data to the sink at the beginning of the time slot $t_2 + 1$. Let t_{w1} be the latest schedule of all the white nodes in layer 1. t_{w1} must be less than the largest size, Z , of the *competitor sets* of all the white nodes. This can be guaranteed by Lemma 1. After time slot t_{w1} , only gray nodes in layer 1 are left to send their data, thus less than 20 time slots are needed. Thus, the total latency $T \leq Z + 20$ in this situation. The bound of Z is shown in Lemma 2 later.

In case two, the white node v has the latest schedule t_1 . Similar as the latter situation in case one, the total latency is less than Z . Thus, we have $T \leq Z$ in this case.

In summary, we have the following inequation.

$$t_1 \leq \begin{cases} 20 + t_2, & \text{all white nodes finish sending at } t_2 + 1; \\ 20 + Z, & \text{otherwise.} \end{cases} \quad (1)$$

Similar to t_1 , we have the following inequation about t_2 . The detailed proof of the inequation can be found in our full paper [20].

$$t_2 \leq \begin{cases} 5 + t_3, & \text{all white nodes finish sending at } t_3 + 1; \\ 5 + Z, & \text{otherwise.} \end{cases} \quad (2)$$

In general, the following inequations for t_{2k-1} and t_{2k} hold. The proofs can also be referred to our full paper [20].

$$t_{2k-1} \leq \begin{cases} 19 + t_{2k}, & \text{all white nodes finish sending at } t_2 + 1; \\ 19 + Z, & \text{otherwise.} \end{cases} \quad (3)$$

$$t_{2k} \leq \begin{cases} 5 + t_{2k+1}, & \text{all white nodes finish sending at } t_2 + 1; \\ 5 + Z, & \text{otherwise.} \end{cases} \quad (4)$$

Lemma 2: In a UDG model, for an arbitrary white node u in the aggregation tree, the size of its *competitor set* is at most $20 + 6\Delta$, where Δ is the maximum node degree.

Proof: For a white node u , its *competitor set* can be divided into two sets according to Proposition 1. One is the set of 1-hop neighbors u 's parent p , named C_1 . The other is the set of children of u 's 1-hop neighbors, named C_2 . It is obvious that the size of C_1 is at most Δ . C_2 consists of the children of u 's black and gray neighbors as white neighbors have no children. First, we determine the number of the children of u 's black neighbors. A white node has at most 5 black neighbors in a UDG [21]. Thus, the number of the children of u 's black neighbors can be bounded by 5Δ . Now, we determine the number of the children of u 's gray neighbors. The children of u 's gray neighbors are all black nodes within the disk of radius 2 centered at u . Since there are at most 21 black nodes in a disk of radius 2 [2] but u 's parent p should be excluded, there are at most 20 black nodes in the disk. Thus the number of the children of u 's gray neighbors can be bounded by 20. Therefore, the size of C_2 is at most $5\Delta + 20$ and the size of u 's *competitor set* is at most $6\Delta + 20$. ■

Theorem 3: The latency bound of the DAS algorithm is at most $24D + 6\Delta + 16$, where D is the diameter of the network and Δ is the maximum node degree.

Proof Sketch: The latency T can be computed iteratively using formulas (1) to (4). The result is $T \leq 12m + Z - 4$, where m is the number of the deepest layer in the aggregation tree and Z is the largest size of the white nodes' *competitor sets*. According to the tree construction, we can prove $m \leq 2D$. Since $Z \leq 6\Delta + 20$ according to Lemma 2, we can easily obtain the conclusion in Theorem 3.

The details of the proof of Theorem 3 can be referred to our full paper [20]. In fact, this latency bound is rather loose, which is shown in the experiments. The reason is that we allow any network topology and there exist many complicated cases that made the tight bound very difficult to be analyzed.

B. Time and Message Complexities

Now we analyze the time cost and the communication cost in terms of the number of messages being transmitted.

The time cost of DAS is $O(n)$, where n is the number of the nodes in the network. This is because that the tree construction algorithm in the first phase of DAS takes $O(n)$

time and SCHDL takes $O(n)$ time since all the nodes make schedules one by one in the worst case.

The number of messages being transmitted is $O(n \cdot \max\{\Delta, \log n\})$ for the following reasons. First, the tree construction algorithm in the first phase of DAS needs $O(n \log n)$ message transmissions [19]. Next, SCHDL in the second phase requires $O(n\Delta)$ message transmissions. The reason is that each node sends and receives messages from its *competitors* and the number of messages transmitted by each node can be bounded by the largest size of the nodes' *competitor sets*, which is the largest size of the white nodes' *competitor sets* Z , $Z \leq 6\Delta + 20$, since the size of any *competitor set* of black or gray node is no more than $\Delta + 20$. Therefore, the message transmissions needed by SCHDL is $O(n\Delta)$.

C. Discussion of Huang's Algorithm [2]

We found that the algorithm in [2] cannot derive a collision-free schedule in some cases. Let's refer to Figure 4. The algorithm first constructs an aggregation tree and then schedules the nodes layer by layer. Since the tree construction in the first phase is based on maximal independent set [2], the tree in Figure 4 is a typical form of the aggregation tree constructed by the algorithm. We now run the algorithm on the tree. As the nodes are scheduled layer by layer, all the white leaf nodes are scheduled in the first step of the algorithm. The leaves in the tree are 8, 5, 4, 3, 9, 11, 12, and 14. The main idea of the algorithm to schedule a node set S is that each iteration of the algorithm forms a node set X that send data in one time slot and checks each unscheduled node u whether it is a neighbor of the parent of a node v in X . If it is not, add u to X . At first, X is a null set so that node 8 is added to X . Then node 5 finds it is not a neighbor of 8's parent 10, node 5 is also added to X . In this way, the output schedule of the algorithm is that {8, 5, 3, 9} send data in slot 1 and {4, 11, 12, 14} send data in slot 2. We can see that collisions occur at node 1 and node 15 in slot 1 due to nodes 8 and 5 sending data simultaneously and nodes 8 and 3 sending data simultaneously. The reason is that though node 5 and 3 are not node 8's parent's 1-hop neighbors, they are the children of node 8's 1-hop neighbors. The algorithm ignores the children of a node's 1-hop neighbors, which form an important part of a node's *competitors*. This fault makes their algorithm generate schedules with collisions in many cases because the number of the children of a node's 1-hop neighbors is often large comparing to its parent's 1-hop neighbors. Since collisions occur during aggregation, their algorithm does not give a correct solution to the scheduling problem. Thus the time latency bound of their algorithm is not fair. For this reason, we will not compare with the result in [2].

Thus the state-of-the-art algorithm generating collision-free schedules is the one in [3] rather than the algorithm in [2]. Its latency is $(\Delta - 1)R$ where both Δ and R can be of the same order of the network size. The latency of our algorithm is $24D + 6\Delta + 16$, where Δ is only an additive factor. Since $D \leq 2R$ for any arbitrary network graph, DAS is a nearly

constant approximation since the latency cannot be less than the network radius R .

VI. AN ADAPTIVE SCHEDULING METHOD

In this section, we add adaptive strategies into the DAS algorithm to deal with the case where the network topology changes. First we introduce the strategy for maintaining the aggregation tree and then we propose the strategy for aggregation scheduling.

A. Maintenance of Aggregation Tree

This paper only considers the stationary sensor networks where nodes cannot move. Thus, the changes of the networks are mainly caused by new nodes joining in and node failures.

When a node u joins in a network, it sends a JOIN message. All the nodes in u 's transmission area receive the message. For any node v receiving the JOIN message, it sends back an ACK message including its node color. After u collects all the ACK messages from its neighbors, it checks if there are messages from black nodes. If so, it picks any one, say w , as its parent and sends a CHILD message to w . Node u becomes a leaf and its color is set to white. Otherwise u has no black neighbors, and u makes itself a black node. Since the network is connected, u must have at least a white or gray neighbor. u can randomly pick one as its parent and sends a CHILD message. Upon receiving a CHILD message, a node records the sender as its child and checks if it is a white node. If so, it turns itself to a gray one.

When a node failure occurs, the case is a bit complex. Typically node failures are detected by the mechanism of sending confirming messages periodically. If node v sends a confirming message to node u and has not received u 's response for a predefined time duration, v believes that u fails. Assume u fails. If u is a leaf node, its parent p will find out that u fails after a while. In this case, p removes u from its child list. If u is a gray node, its parent p and its black children will find out after a while. In this case, p deletes u from its child list and for any black child v of u we adopt the following strategy to find a parent for v . First, v finds a white or gray neighbor w , which is not its child, as its parent and sends a CHILD message to node w . If v cannot find one, v turns its color to white and finds its parent. Its white children also need to find parents for themselves. The white nodes can be seen as the new joining nodes during the finding of their parents. v 's gray children find their parents using the strategy proposed as follows. If u is a black node, u 's parent p deletes u from its child list and checks if it has any other black children. If not, p turns itself to a white node. For any white child of u , again we take it as a new node joining in. For any gray child v of u , v finds its parent in the following way. First, v checks if there are black neighbors who are not its children. If so, v randomly chooses one from them as its parent and sends to it a CHILD message. Otherwise, v randomly chooses one of its children w , which must be black, as its parent and sends to w a CHILD message. v 's black children lose their parents and use the strategy above of finding the parent for a black node to

find their parents. As the process is recursive, any node who loses its parent can find a new one as long as the network is connected.

In this way, the new aggregation tree is maintained without changing its original structure. In most cases the above process does not need much time and communication since it is done locally.

B. Adaptive Scheduling using DAS

After the aggregation tree has been updated, the nodes who have changed their parents compute their new *competitors* locally and send their updated information to their competitors. It is relatively easy to make a new schedule using DAS. All the nodes whose parents have changed form a set Upd . For each node u in Upd , it marks itself *renewed* and sends a RENEW message to its parent. For each node v receiving a RENEW message, if it is not a child of the sink, it marks itself *renewed* and sends a RENEW message to its parent. This is because if a node has some new child, it may need to change its schedule and so do its parent and grandparent and so on. Then the *unrenewed* nodes send their schedules to their *renewed competitors*. After these processes, the *renewed* nodes start to run SCHDL of DAS.

Since the *renewed* nodes have collected all the updated local information, their schedules made by SCHDL cannot collide with the *unrenewed* nodes. This guarantees that the new schedule is also collision-free. Generally, only a small number of nodes join or fail in the network in most cases so that there are not many *renewed* nodes in the network. For very rare occasions where there are large amounts of nodes joining or failing, we make all nodes run DAS again, which is reasonable. Therefore, the adaptive scheduling algorithm can be efficient in most cases.

VII. SIMULATION RESULTS

In our simulation, we randomly deploy sensors into a region of $200m \times 200m$. All sensors have the same transmission radius. Since the algorithm in [2] cannot give a correct solution to the scheduling problem, we compare the performance of our algorithm DAS with the only correct algorithm for aggregation scheduling proposed in [3] by Chen *et al.*

A. Evaluating DAS

We first implement DAS and Chen's algorithm using C++. In Figure 6, the transmission radius of each sensor is fixed to $25m$. The figure shows the number of the time slots needed to aggregate data from the leaves to the sink by running the two algorithms while the number of nodes increases. Figure 7 compares the number of slots to aggregate data using the two algorithms when the transmission radius varies. Here the number of nodes is fixed to 1600. It can be seen from the two figures that DAS outperforms Chen's algorithm with much lower latencies. The bigger the number of nodes or the transmission radius is, the better the improvement of DAS is in comparison with Chen's algorithm. From the figures, DAS's latencies are 1.65 to 2.88 times smaller than Chen's

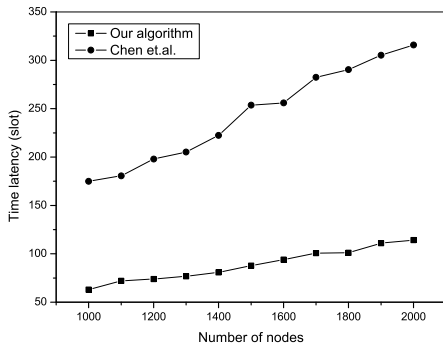


Fig. 6. Latency with different number of nodes

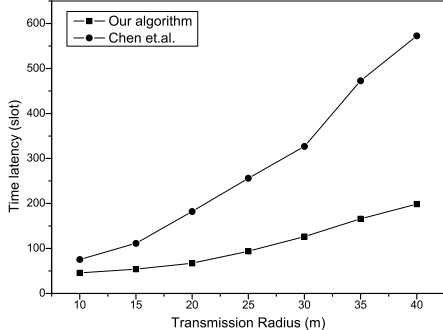


Fig. 7. Latency with different transmission radius

algorithm, and with the increment of the number of nodes and the transmission radius the improvement will be larger. It is worth to note that the time latencies in the experiments are 7 to 8 times better than the theoretical latency bound. For instance, the time latency for 1400 nodes is 80.9 while the upper bound is 712 ($D = 12$ and $\Delta = 68$). The reason is that we magnify the latency too much when computing the upper bound, which causes the upper bound for latency very loose.

Next we examine the running time of DAS in Figure 8 when the number of sensor nodes and the transmission radius vary. The running time refers to the time required to generate the schedule. We set $1/20$ of a second for each time slot when running DAS, which is greater than the time required to transmit one packet. Though the time complexity is $O(n)$, we can see that the running time is rather small. We can see that the running time is 5 to 20s on average. For instance, the running time for 2000 nodes with transmission radius of 30m is only 14.6s. The reason is that many nodes can schedule simultaneously and the occasion where only one node can schedule at each time rarely happens. As the number of the nodes or the transmission radius increases, the average size of the nodes' *competitor sets* also increases, and thus each node has to compete with more *competitors* that costs more time. Since DAS is a distributed algorithm, we also study its communication cost in terms of the number of message transmissions. Figure 9 shows the transmissions per node with different number of nodes and transmission radius. The transmission for a node means the number of messages sent and received by the node. For the similar reason above, DAS incurs more transmissions when the number of the nodes and the transmission radius increases.

B. Evaluating Adaptive DAS

In this subsection, we study the performance of DAS while confronting node joining and node failures. First we carry out experiments to observe the changes of latency and the changes of transmission per node when there are nodes joining and then we look into the cases of node failures.

In the first case, we study the increasing rate of latency and transmission when the node joining rate increases. The increasing rate of latency refers to the ratio of the difference of the new and old latency to the old latency. Figure 10 shows the increased latency after nodes join in the network with different joining rate. The three curves correspond to different original size of the network as is shown in the Figure 10. We find that in different network scales, the trends of latency increments are similar while the node joining rate increases but not much. For example, the increasing rate of latency is from 0.371 to 0.378 with different number of nodes when the node joining rate is 0.3. We find that the percentage of the latency increment is only a little larger than that of joining nodes. This result shows that the adaptive DAS scales well with the size of the network. Figure 11 shows the increasing rate of average transmissions with different node joining rate, i.e. the ratio of the transmissions per node when running adaptive DAS to the original transmission when running DAS. This is also a metric for measuring the scalability of the adaptive algorithm. The increased transmission rate is going up slowly with more nodes joining in and the trends are similar in different network scales, which again shows that adaptive DAS has a good scalability.

In the second case, the time latency increments and increased transmissions are studied with different failure rate. We find in Figure 12 that the latency difference is between -1 and 1 in most cases, which means the time latency keeps almost invariant with different node failure rate. Figure 13 shows the average transmissions to modify the schedule after node failures, which is rather small. For example, the transmission increment per node is from 6.6 to 7.5 with different number of nodes when the node failure rate is 0.2. Our philosophy is that only schedules of a small set of nodes need to be updated using very few transmissions since the node failure cannot be of a large scale. Since the latency of the whole schedule is determined by the latest schedule of the nodes, our updating strategy has very little chance of modifying the latest schedule. Therefore, adaptive DAS keeps the latency almost invariant using very few transmissions. Another strategy is to run DAS again on all the nodes and form a new schedule with a smaller latency, but the transmissions will be much more. We need a tradeoff between latency and energy consumption. We choose saving energy first in handling dynamic network and we can run DAS one more time when the failure rate is higher than a predefined threshold.

VIII. CONCLUSION

Data aggregation is critical to the network performance in wireless sensor networks and aggregation scheduling is a feasible way of improving the aggregation quality. In this

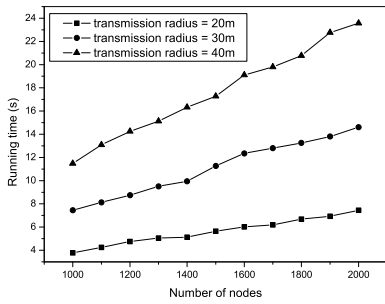


Fig. 8. Running time with different number of nodes

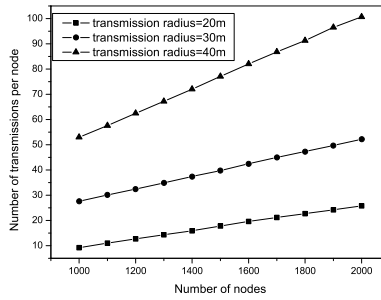


Fig. 9. Average transmission with different number of nodes

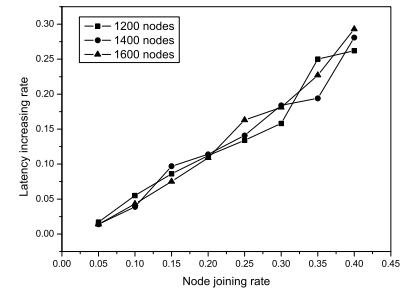


Fig. 10. Latency increments with different joining rate

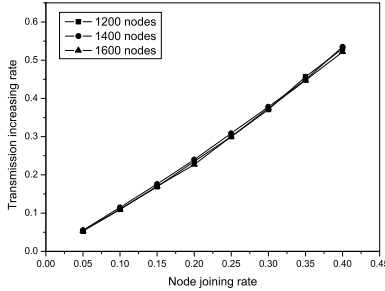


Fig. 11. Transmission increments with different joining rate

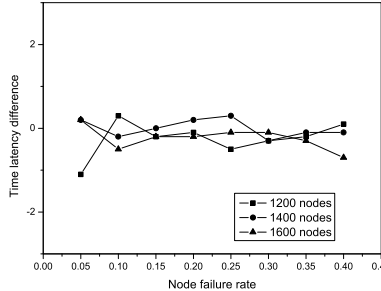


Fig. 12. Latency differences with different failure rate

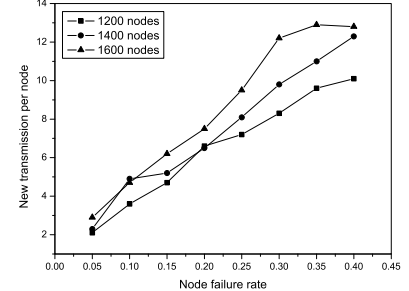


Fig. 13. Transmission increments with different failure rate

paper we study the problem of distributed aggregation scheduling in sensor networks and propose a distributed scheduling algorithm with latency bound $24D + 6\Delta + 16$. This is a nearly constant approximate algorithm that significantly reduces the aggregation latency. The theoretical analysis and the simulation results show that our algorithm outperforms the algorithms in [3]. An adaptive method is also proposed, which makes the algorithm suitable for networks with topology changes.

ACKNOWLEDGMENT

This work is supported in part by the National Grand Fundamental Research 973 Program of China under grant 2006CB303000, the Key Program of National Science Foundation of China under grant 60533110, the NSFC-RGC of China under grant 60831160525, the Program of New Century Excellent Talents in University under grant NCET-05-0333, the US NSF under grants CCF-0545667 and CCF-0844829.

REFERENCES

- [1] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis, "Tina: A scheme for temporal coherency-aware in-network aggregation," in *Proc. of MobiDE Workshop*, 2003.
- [2] S. C.-H. Huang, P.-J. Wan, C. T. Vu, Y. Li, and F. Yao, "Nearly constant approximation for data aggregation scheduling in wireless sensor networks," in *INFOCOM*, 2007, pp. 366–372.
- [3] X. Chen, X. Hu, and J. Zhu, "Minimum data aggregation time problem in wireless sensor networks," in *1st Int'l Conference on Mobile Ad-hoc and Sensor Network-MSN*, 2005, pp. 133–142.
- [4] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: a tiny aggregation service for ad-hoc sensor networks," in *OSDI*, 2002.
- [5] J. Beaver, M. A. Sharaf, A. Labrinidis, and P. K. Chrysanthis, "Location aware routing for data aggregation for sensor networks," in *Post Proc. of Geo Sensor Networks Workshop*, 2003.
- [6] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann, "Impact of network density on data aggregation in wireless sensor networks," in *22nd Int'l Conference on Distributed Computing Systems-ICDCS*, 2002.

- [7] A. Silberstein, R. Braynard, and J. Yang, "Constraint-chaining: On energy-efficient continuous monitoring in sensor networks," in *SIGMOD*, 2006.
- [8] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, "Medians and beyond: New aggregation techniques for sensor networks," in *Sensys*, 2004.
- [9] J. Considine, F. Li, G. Kollios, and J. Byers, "Approximate aggregation techniques for sensor databases," in *ICDE*, 2004, pp. 449–460.
- [10] D. Chu, A. Deshpande, J. Hellerstein, and W. Hong, "Approximate data collection in sensor networks using probabilistic models," in *ICDE*, 2006.
- [11] A. Deshpande, C. Guestrin, W. Hong, and S. Madden, "Exploiting correlated attributes in acquisitional query processing," in *ICDE*, 2005.
- [12] B. K. Y. Yu and V. K. Prasanna, "Energy-latency tradeoffs for data gathering in wireless sensor networks," in *INFOCOM*, 2004.
- [13] V. Annamalai, S. K. S. Gupta, and L. Schwiebert, "On tree-based convergecasting in wireless sensor networks," in *IEEE Wireless Communications and Networking-WCNC*, vol. 3, 2003, pp. 1942–1947.
- [14] S. Upadhyayula, V. Annamalai, and S. K. S. Gupta, "A low-latency and energy-efficient algorithm for convergecast in wireless sensor networks," in *IEEE Global Telecommunications Conference-GLOBECOM*, vol. 6, 2003, pp. 3525–3530.
- [15] A. Kesselman and D. Kowalski, "Fast distributed algorithm for convergecast in ad hoc geometric radio networks," in *2nd Annual Conference on Wireless On-demand Network Systems and Services-WONS*, 2005, pp. 119–124.
- [16] H. Lee and A. Keshavarzian, "Towards energy-optimal and reliable data collection via collision-free scheduling in wireless sensor networks," in *INFOCOM*, 2008.
- [17] S. Gandham, Y. Zhang, and Q. Huang, "Distributed minimal time convergecast scheduling in wireless sensor networks," in *ICDCS*, 2006.
- [18] Y. Zhang, S. Gandham, and Q. Huang, "Distributed minimal time convergecast scheduling for small or sparse data sources," in *RTSS*, 2007.
- [19] P.-J. Wan, K. M. Alzoubi, and O. Frieder, "Distributed construction of connected dominating set in wireless ad hoc networks," in *INFOCOM*, 2002.
- [20] "http://www.cs.gsu.edu/yli/datasets/das-full.pdf."
- [21] B. N. Clark, C. J. Colbourn, and D. S. Johnson, "Unit disk graphs," in *Discrete Mathematics*, vol. 86, 1990, pp. 165–177.