

Efficient Data Aggregation in Multi-hop WSNs

XiaoHua Xu*, ShiGuang Wang*, XuFei Mao*, ShaoJie Tang*, Ping Xu*, XiangYang Li*

Abstract—Data aggregation is a key functionality for wireless sensor network (WSN) applications. This paper focuses on data aggregation scheduling problem to minimize the latency. We propose an efficient distributed method that produces a collision-free schedule for data aggregation in WSNs. We theoretically prove that the latency of the aggregation schedule generated by our algorithm is at most $16R + \Delta - 14$ time-slots. Here R is the network radius and Δ is the maximum node degree in the communication graph of the original network. Our extensive simulation results corroborate our theoretical results and show that our algorithms perform better in practice.

Index Terms—Wireless networks, aggregation, scheduling, latency, sensor.

I. INTRODUCTION

Wireless sensor networks (WSNs) have drawn considerable amount of research interests for their omnipresent applications such as environmental monitoring, spatial exploration and battlefield surveillance. To design and deploy successful wireless sensor networks, many issues need to be resolved such as deployment strategies, energy conservation, routing in dynamic environment, localization and so on. All the issues essentially correlate to collecting data from a set of targeted wireless sensors to some sink node(s) and then performing some further analysis at sink node(s) which can be termed as many-to-one communication. In-network data aggregation [15] is one of the most common many-to-one communication patterns used in these sensor networks, thus it becomes a key field in WSNs and has been well-studied in recent years.

We consider the problem of designing a schedule for data aggregation from within networks to sink node(s) with minimum time-slot latency. Some of previous research works on in-network aggregation did not consider the collision problem and left it to the MAC layer. Resolving collisions in MAC layer could incur a large amount of energy consumption and a large latency during aggregation. Thus, in this paper we mainly concentrate on the TDMA scheduling problem above the MAC layer. To define the problem formally, consider a wireless network G formed by n wireless nodes $V = \{v_1, \dots, v_n\}$ deployed in a 2-dimensional region. $v_s \in V$ is the sink node that will collect the final aggregation result. Every node v_i has a transmission range r and interference range $r_I = \Theta(r)$. A node v_i can send data correctly to another node v_j , if and only if (1) v_j is within v_i 's transmission range, and (2) v_j is not within interference range r_I of any other transmitting

node. Every node v_i has an ability to monitor the environment, and collect some data (such as temperature), *i.e.*, v_i has a set of raw data A_i . Let $A = \cup_{i=1}^n A_i$ and $N = |A|$ be the cardinality of the set A . Then $\langle A_1, A_2, \dots, A_i, \dots, A_n \rangle$ is called a distribution of A at sites of V . Data aggregation is to find the value $f(A)$ for a certain function f , such as *min*, *max*, *average*, *variance* and so on with minimum time latency.

The data aggregation scheduling problems have been extensively studied in recent years. The most related ones are as follows. Huang *et al.* [9] proposed a scheduling algorithm with the latency bound of $23R + \Delta + 18$ time-slots, where R is the network radius and Δ is maximum node degree. However the interference model used in [9] is a simple primary interference model: no node can send and receive simultaneously. Under Protocol Interference Model, Yu *et al.* [3] proposed a distributed scheduling algorithm generating collision-free schedules that has a latency bound of $24D + 6\Delta + 16$ time-slots, where D is the network diameter.

The main contributions of this paper are as follows. We propose efficient algorithms that will construct a data aggregation tree and a TDMA schedule for all links in the tree such that the latency of aggregating all data to the sink node is approximately minimized. For simplicity of analysis, we use the following interference model: when a node v is receiving data from a sender u , v is not within the interference range r_I of any other active sender x . As an illustration, we first present an efficient centralized algorithm that will build a TDMA schedule of links based on the aggregation tree which is build distributively. Our schedule uses a bottom-up approach: schedule nodes level by level starting from the lowest level. For simplicity of analysis we assume that the interference range $r_I = r$, and then we theoretically prove that the latency of the aggregation schedule generated by our algorithm is at most $16R + \Delta - 14$ time-slots. Notice that, for general r_I , our algorithm will produce a collision-free schedule for aggregation whose latency is at most $\Theta((\frac{r_I}{r})^2 R + \Delta)$ time-slots. We then present an efficient distributed algorithm that builds an aggregation tree and gives a schedule for each link. For simplicity, our distributed method assumes that the clocks of all nodes are synchronized. Unlike our centralized algorithm, our distributed algorithm will *not* explicitly produce a schedule for links in the aggregation tree. The link schedule is implicitly generated in the process of data aggregation. Our distributed scheduling algorithm thus works well in dynamic networks, as long as the constructed backbone of the network by our algorithm remains unchanged. Obviously, when $r_I = r$, for a network G with radius R and the maximum node degree Δ , the latency by *any* data aggregation algorithm is at least R . This implies that our algorithm is within a small constant factor of the optimum. We then conduct extensive simulations to study the practical performances of our proposed data

Dept. of Computer Science, Illinois Institute of Technology. Emails: {xxu23, swang44, xmao3, stang7, pxu3}@iit.edu, xli@cs.iit.edu. The research of authors are partially supported by NSF CNS-0832120, National Natural Science Foundation of China under Grant No. 60828003, the National High Technology Research and Development Program of China (863 Program) under grant No. 2007AA01Z180.

aggregation methods. Our simulation results corroborate our theoretical results and show that our algorithms perform better in practice. We find that data aggregation by our distributed methods have latency close to R .

The rest of the paper is organized as follows. Section II formulates the problem. We present our centralized and distributed scheduling algorithms in Section III and analyze their performances in Section IV. Section V presents the simulation results. Section VI outlines the related work. Section VII concludes the paper.

II. SYSTEM MODELS

A. Aggregation Functions

The database community classifies aggregation functions into three categories: distributive (e.g., *max*, *min*, *sum*, *count*), algebraic (e.g., *plus*, *minus*, *average*, *variance*) and holistic (e.g., *median*, *kth smallest or largest*). In this paper, we only focus on the distributive or algebraic ones.

A function f is said to be *distributive* if for every pair of disjoint data sets X_1, X_2 , we have $f(X_1 \cup X_2) = h(f(X_1), f(X_2))$ for some function h . For example, when f is *sum*, then h can be set as *sum*; when f is *count*, h is *sum*. Assume that we are given an aggregation function F that can be expressed as the combination of k distributive functions for some integer constant k , i.e.,

$$f(X) = h(g_1(X), g_2(X), \dots, g_k(X)).$$

For example, when f is *average*, then $k = 2$, g_1 can be set as *sum*, g_2 can be set as *count* (obviously both g_1 and g_2 are distributive) and h can be set as $h(y_1, y_2) = y_1/y_2$. Thus, instead of computing f , we will just compute $y_i = g_i(X)$ distributively for $i \in [1, k]$ and $h(y_1, y_2, \dots, y_k)$ at the sink node.

B. Network Model

We consider a wireless sensor network consisting of n nodes V where $v_s \in V$ is the sink node. Each node can send (receive) data to (from) all directions. For simplicity, we assume that all nodes have the same transmission range r such that two nodes u and v form a communication link whenever their Euclidean distance $\|u - v\| \leq r$. In the rest of the paper we will assume that $r = 1$, i.e., normalized to one unit. Then the underneath communication graph is essentially a unit disk graph (UDG).

Let $A, B \subset V$ and $A \cap B = \emptyset$. We say data are aggregated from A to B in one time-slot if all the nodes in A transmit data simultaneously in one time-slot and all data are received by some nodes in B without interference. We will define interference at the end of this section. Then a data aggregation schedule with latency l can be defined as 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) $\cup_{i=1}^l S_i = V \setminus \{v_s\}$;
- 3) Data are aggregated from S_k to $V \setminus \cup_{i=1}^k S_i$ at time-slot k , for all $k = 1, 2, \dots, l$ and all the data are aggregated to the sink node v_s in l time-slots.

Notice that here $\cup_{i=1}^l S_i = V \setminus \{v_s\}$ is to ensure that every data will be aggregated; $S_i \cap S_j = \emptyset, \forall i \neq j$ is to ensure

that every data is used at most once. To simplify our analysis, we will relax the requirement that $S_i \cap S_j = \emptyset, \forall i \neq j$. When the sets $S_i, 1 \leq i \leq l$ are not disjoint, in the actual data aggregation, a node v , that appears multiple times in $S_i, 1 \leq i \leq l$, will participate in the data aggregation only once (say the smallest i when it appears in S_i), and then it will only serve as a relay node in the following appearances.

The distributed aggregation scheduling problem is to find a 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 [4]. This paper proposes an approximate distributed algorithm with latency $16R + \Delta - 14$ time-slots, where R is the network radius and Δ is the maximum node degree.

We assume that a node cannot send and receive data simultaneously. In protocol interference model, we assume that each node has a transmission range r and an interference range $r_I \geq r$. A receiver v of a link uv is interfered by another sender p of a link pq if $\|p - v\| \leq r_I$. As [4], [9], we first assume that $r_I = r$, which is normalized to 1 unit in this paper. We will later study a more general case $r_I \geq r$.

C. Related Terminologies

For simplicity, we present our distributed algorithms in a synchronous message passing model in which time is divided into slots. In each time-slot, a node is able to send a message to one of its neighbors for unicast communication. Note that, at the cost of higher communication, our algorithms can also be implemented in asynchronous communication settings using the notions of synchronizer.

In a graph $G = (V, E)$, a subset S of V is a *dominating set* (DS) if for each node u in V , it is either in S or is adjacent to some node v in S . Nodes from S are called dominators, whereas nodes not in S are called dominatees. A subset S of nodes is *independent set* (IS), if for any pair of nodes in S , there is no edge between them. An IS S is a maximal IS if no another IS that is a superset of S . Clearly, a maximal IS is always a DS. A subset C of V is a *connected dominating set* (CDS) if C is a dominating set and C induces a connected subgraph. Consequently, the nodes in C can communicate with each other without using nodes in $V \setminus C$. A CDS is also called a backbone here.

III. DISTRIBUTED AGGREGATION SCHEDULING

Our data aggregation scheduling (DAS) algorithm consists of two phases: 1) aggregation tree construction and 2) aggregation scheduling. As an illustration of our methods, we first present a centralized version of our data aggregation scheduling. We adopt an existing method for the first phase and the second phase is the core of our algorithm. We will present these two phases in the following two sections. At the end of the section, we present a distributed implementation based on our centralized aggregation scheduling algorithm.

A. Aggregation Tree Construction

In the first phase we construct an aggregation tree in a distributed way using an existing approach [14]. We employ

a connected dominating set (CDS) in this phase since it can behave as the virtual backbone of a sensor network. A distributed approach of constructing a CDS has been proposed by Wan *et al.* [14]. In their algorithm, a special dominating set using a MIS of the network is constructed first and then a CDS is constructed to connect dominators and the other nodes. All nodes in the MIS is colored **black** and all other nodes are colored **grey**. This CDS tree can be used as the aggregation tree in our scheduling algorithm with a small modification as follows.

1) We choose the *topology center* of the UDG as the root of our BFS tree. Notice that, previous methods will use the sink node as the root. Our choice of the topology center enables us to reduce the latency to a function of the network radius R , instead of the network diameter D proved by previous methods. Here a node v_0 is called the *topology center* in a graph G if $v_0 = \arg \min_v \{\max_u d_G(u, v)\}$, where $d_G(u, v)$ is the hop distance between nodes u and v in graph G . $R = \max_u d_G(u, v_0)$ is called the *radius* of the network G . Notice that for most networks, the topology center is different from the sink node.

2) After the topology center gathered the aggregated data from all nodes, it will then send the aggregation result to the sink node via the shortest path from the topology center v_0 to the sink node v_s . This will incur an additional latency $d_G(v_0, v_s)$ of at most R .

The *rank* of a node u is $(level, ID(u))$, where *level* is the hop-distance of u to the root in the BFS. The ranks of nodes are compared using lexicographic order.

B. Centralized Approach

The second phase is aggregation scheduling which is the core of the whole algorithm. It is based on the aggregation tree constructed in the first phase. As an illustration, we first present an efficient centralized algorithm. We will then present our distributed scheduling implementation in Section III-C.

Algorithm 1 shows how the data from the dominatees are aggregated to the dominators. Our method is a greedy approach, in which at every time-slot, the set of dominators will gather data from as many dominatees (whose data has not been gathered to a dominator yet) as possible. Notice that since the maximum degree of nodes in the communication graph is Δ , our method guarantees that after at most Δ time-slots, all the dominatees' data will be gathered to their corresponding dominators when considering the interference, which will be proved in Lemma 2. The basic idea is as follows: each dominator will randomly pick a dominatee whose data is not reported to any dominator yet. Clearly, these selected dominatees may not be able to send their data to corresponding dominators in one time-slot due to potential interferences. We then reconnect these dominatees to the dominators (and may not schedule some of the selected dominatees in the current time-slot), using Algorithm 2, such that these new links can communicate concurrently.

Suppose that two directed links $u_i z_i$ and $u_j z_j$ interfere with each other (see Fig.1), where the dominatees u_i and u_j are

Algorithm 1 Aggregate Data to Dominators

- 1: **for** $i = 1, 2, \dots, \Delta$ **do**
 - 2: Each dominator randomly chooses 1 neighboring dominatee, whose data is not gathered yet, as transmitter. The set of such chosen links form a link set L .
 - 3: If there are conflicts among chosen links, resolve interference using Algorithm 2;
 - 4: All the remaining links in L now transmit simultaneously;
 - 5: $i = i + 1$;
-

Algorithm 2 Reconnect Dominatees to Dominators

- 1: **while** (exist a pair of conflicting links) **do**
 - 2: Let $u_i z_i$ and $u_j z_j$ be one of the pairs of conflicting links.
 - 3: Find the sets D_i and D_j based on rules described previously.
 - 4: **if** ($|u_i z_j| \leq 1$ and $|u_j z_i| > 1$) **then**
 - 5: If $D_j \neq \phi$, replace $u_j z_j$ by a link $u_j z_{j_0}$, for a $z_{j_0} \in D_j$.
 - 6: If $D_j = \phi$, then remove the link $u_j z_j$.
 - 7: **else if** ($|u_j z_i| \leq 1$ and $|u_i z_j| > 1$) **then**
 - 8: If $D_i \neq \phi$, then replace $u_i z_i$ with $u_i z_{i_0}$, for $z_{i_0} \in D_i$.
 - 9: If $D_i = \phi$, then remove link $u_i z_i$.
 - 10: **else if** ($|u_j z_i| \leq 1$ and $|u_i z_j| \leq 1$) **then**
 - 11: If $D_i = \phi$, remove the link $u_i z_i$; otherwise, if $D_j = \phi$, remove the link $u_j z_j$.
 - 12: If both D_i and D_j are not empty, replace $u_i z_i$ and $u_j z_j$ by two new links $u_i z_{i_0}$, $u_j z_{j_0}$, for $z_{i_0} \in D_i$, $z_{j_0} \in D_j$.
-

transmitters in these two links respectively and z_i and z_j are dominators. For each dominatee v , let $D(v)$ be the set of neighboring dominators. Obviously, $|D(v)| \leq 5$ for any node v . Let $D(u_i) = D(u_i) \setminus \{z_i\}$, $D(u_j) = D(u_j) \setminus \{z_j\}$. Notice that here $D(u_i)$ and $D(u_j)$ may be empty, and $D(u_i) \cap D(u_j)$ may also not be empty.

For each active transmitter v , $v \neq u_i$ and $v \neq u_j$, we delete all dominators from $D(u_i)$ (and also from $D(u_j)$) that are within the transmission range of v . Notice that we can discard these dominators since their degrees are already decreased by at least 1 because of the existence of some active transmitter v . We also delete the dominators that are within transmission range of both u_i and u_j from $D(u_i)$ and $D(u_j)$. Notice that we can do this because these dominators' degree will be decreased by 1 since our re-scheduling can guarantee at least one transmitter of u_i and u_j will remain as an active transmitter, as we will show later.

Let D_i (resp. D_j) be the set of remaining dominators in $D(u_i)$ (resp. $D(u_j)$).

Fig. 1(a) illustrates one possible state after the preceding two deletions of dominators from $D(u_i)$ and $D(u_j)$. Notice that

- 1) The distance between u_i and any member of D_j is greater than 1. The distance between u_j and any member of D_i is greater than 1.

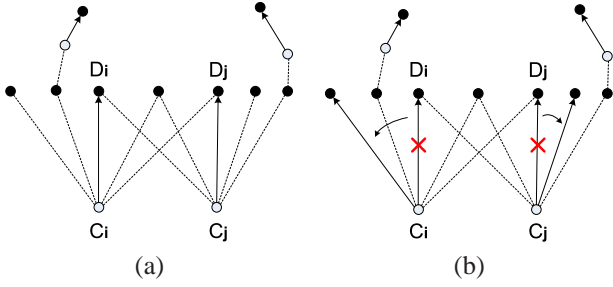


Fig. 1. (a) An interference between 2 links. (b) A state after rescheduling the two links.

2) Both D_i and D_j may be empty and may not be empty. Algorithm 2 shows how to re-connect dominatees to dominators to avoid the interference.

After all the data in the dominatees have been aggregated to dominators, our next step is to aggregate all the intermediate results in the dominators to the root.

We can see that in each layer of BFS tree, there are some dominator(s) and some dominatee(s). For every dominatee, it has at least one dominator neighbor in the same or upper level. Thus, every dominator (except the root) has at least one dominator in the upper level within two-hops. Using this property, we can ensure that all the data in the dominators can reach the root finally if every dominator transmits its data to some dominator in upper level within two-hops. From another point of view, considering dominators in the decreasing order of their levels, a dominator u in level L aggregates data from all dominators in level $L+1$ or $L+2$ that are within two-hops of u . This will ensure that all the data will be aggregated to the root. Algorithm 3 presents our method in detail.

In Algorithm 3 we only concentrate on communications between dominators. Since dominators cannot communicate directly, we have to rely on some dominatees, each of which acts as a bridge between two dominators. Hereafter we rename these dominatees as *connectors*. The algorithm runs from lower level to upper level in aggregation tree, every dominator will remain silent until the level where it locates begins running. When it is its turn, the dominator will try to gather all the data from other dominators in lower levels that have not been aggregated. If a dominator's data has been collected before, then it is unnecessary to be collected again. Actually we have to guarantee this for every data should be and only be used once. Our algorithm implements this by discarding the dominators after their data have been gathered to upper levels.

Notice that in our algorithm after we process dominators B_i (all dominators in level i), there may still have some dominators in B_{i+1} whose data are not aggregated. This could happen because a dominator in B_{i+1} could be within 2-hops of some dominator in B_{i-1} , but not within 2-hops of *any* dominator from B_i . We conclude that after the execution of all the dominators in B_i , all dominators in B_{i+2} have already been aggregated their data.

C. Distributed Implementation

Now we present a distributed implementation for our data aggregation scheduling. The distributed implementation consists of three stages:

Algorithm 3 Centralized-DAS

Input: The BFS tree with root v_0 and depth R , and a distributive aggregation function f , data A_i stored at each node v_i .

- 1: Construct the aggregation tree T' based on CDS. Remove the redundant connectors to ensure that each dominator uses at most 12 connectors to connect itself to all dominators in lower level and is within 2-hops. Here a connector node x (a dominatee of a dominator u) is said to be *redundant* for the dominator u , if removing x will *not* disconnect any of the 2-hop dominators of u from u . Let T be the final data aggregation tree.
 - 2: **for** $i = R - 1, R - 2, \dots, 0$ **do**
 - 3: Choose all dominators, denoted as B_i , in level i of BFS tree.
 - 4: **for** every dominator $u \in B_i$ **do**
 - 5: Node u finds the set $D_2(u)$ of unmarked dominators that are within two-hops of u in BFS, and in lower level $i + 1$ or $i + 2$.
 - 6: Mark all nodes in $D_2(u)$.
 - 7: Every node w in $D_2(u)$ sends $f(A_w, X_1, X_2, \dots, X_d)$ to the parent node (a connector node) in T . Here A_w is the original data set node w has, and X_1, X_2, \dots, X_d are data that node w received from its d children nodes in T .
 - 8: Every node z that is a parent of some nodes in $D_2(u)$ sends $f(X_1, X_2, \dots, X_p)$ to node u (which is the parent of z in T). Here X_1, X_2, \dots, X_p are data that node z received from its p children nodes in T .
 - 9: $i = i - 1$
 - 10: The root v_0 sends the result to the sink using the shortest path.
-

- 1) Every dominatee transmits its data to the neighboring dominator with the lowest level,
- 2) Data are aggregated from dominators from lower levels to dominators in upper levels and finally to the root of the aggregation tree which is the topology center of the network,
- 3) Topology center then transmits the aggregated data to the original sink via the shortest path.

The distributed implementation differs from the centralized one in that the distributed one seeks to transmit greedily: we will try to allocate a node v a time-slot to transmit whenever v has collected the aggregated data from all its children nodes in the data aggregation tree T . Thus the first two phases may interleave in our distributed implementation. The interleaving will reduce the latency greatly since it increases the number of simultaneous transmissions. Later, we will provide the simulation result of our distributed method, which shows that our distributed implementation is quite close to $(1 + \varepsilon)R + \Delta + \Theta(1)$, where ε is a small positive constant. Therefore we conjecture that the data aggregation latency by our distributed implementation indeed has a theoretical performance guarantee of $(1 + \varepsilon)R + \Delta + \Theta(1)$. It will be interesting if we can prove or disprove this conjecture, which

is left as a future work.

To run our algorithm, every node v_i should maintain some local variables, which are

- 1) Leaf indicator: $\text{Leaf}[i] \in \{0, 1\}$, to indicate whether the node v_i is a leaf node in the data aggregation tree.
- 2) Competitor Set: $\text{CS}[i]$, the set of nodes such that for each $j \in \text{CS}[i]$, nodes v_i and v_j cannot transmit simultaneously to their parents due to interference. In other words, if $j \in \text{CS}[i]$, we have either the parent $p_T(i)$ of node v_i in the data aggregation tree T is within the interference range of node v_j ; or the parent $p_T(j)$ of node v_j in the data aggregation tree T is within the interference range of node v_i ; or both. Notice that under the interference model studied in this paper, each node in $\text{CS}[i]$ is within a small constant number of hops of i .
- 3) Ready Competitor Set: $\text{RdyCS}[i]$, which is the set of nodes that collides with i and it is ready to send data to its parent, *i.e.*, it has received the data from all its children nodes.
- 4) Time Slot to Transmit: $\text{TST}[i]$, which is the assigned time-slot that node v_i indeed sends its data to its parent.
- 5) Number of Children: $\text{NoC}[i]$, which is the number of children nodes of v_i in the data aggregation tree T .

Observe that here, at sometime, if we let Rdy be the set of nodes which are ready to transmit (*i.e.*, $v \in Rdy$ iff v has collected the aggregated data from all its children nodes in the data aggregation tree T), and let F denote all the nodes which have finished their transmission, then $\text{RdyCS}[i] = \text{CS}[i] \cap Rdy - F$. The TST of all nodes are initialized to 0. The details of our distributed algorithm are shown in Algorithm 4.

When a node v_i finishes its scheduling, it sends a message FINISH to all nodes in its competitor set $\text{CS}[i]$. When a node i received a message FINISH, it sets its $\text{TST}[i]$ to the larger one of its original $\text{TST}[i]$ and $\text{TST}[j] + 1$. When all the children of node v_i finished their transmission, the node v_i is ready to compete for the transmission time slot and it will send a message $\text{READY}(i, r_i)$ to all nodes in its competitor set. When a node v_i received a message READY from another node v_j , it will add the sender j to its ready competitor set $\text{RdyCS}[i]$ if j is in $\text{CS}[i]$. When the scheduling ends, all nodes will transmit their data based on $\text{TST}[i]$. In the end, the topology center aggregates all the data and sends the result to the sink node via the shortest path.

IV. PERFORMANCE ANALYSIS

In this section we first theoretically prove that the latency of the data aggregation based on our scheduling is at most $16R + \Delta - 14$, where R is the radius of the network and Δ is the maximum node degree in the original communication graph. We conjecture that the theoretical performance of our centralized and distributed algorithms could actually be much better than $16R + \Delta - 14$, which is supported by our extensive simulations. On the other hand, we also present a network example to show that our centralized algorithm cannot achieve a latency lower than $4R + \Delta - 3$. It remains a future work to find bad network examples to show that our distributed

Algorithm 4 Distributed Data Aggregation Scheduling

Input: A network G , and the data aggregation tree T ;

Output: $\text{TST}[i]$ for every node v_i

- 1: The node v_i initializes the value $\text{NoC}[i]$, and $\text{Leaf}[i]$ based on the constructed aggregation tree T .
 - 2: Initializes the set $\text{CS}[i]$ based on the tree T and the original interference relation,
 - 3: $\text{RdyCS}[i] \leftarrow \text{CS}[i] \cap \{j \mid j \text{ is a leaf in } T\}$.
 - 4: $\text{TST}[i] \leftarrow 0$; $\text{DONE} \leftarrow \text{FALSE}$;
 - 5: Node i randomly selects an integer r_i . Then we say $(r_i, i) < (r_j, j)$ if (1) $r_i < r_j$ or (2) $r_i = r_j$ and $i < j$.
 - 6: **while** (not DONE) **do**
 - 7: **if** $\text{NoC}[i] = 0$ **then**
 - 8: Send message $\text{READY}(i, r_i)$ to all nodes in $\text{CS}[i]$.
 - 9: **if** $(r_i, i) < (r_j, j)$ for each $j \in \text{RdyCS}[i]$ **then**
 - 10: Send message $\text{FINISH}(i)$ to all nodes in $\text{CS}[i]$;
 - 11: $\text{DONE} \leftarrow \text{TRUE}$;
 - 12: **if** i received a message $\text{FINISH}(j)$ **then**
 - 13: Delete j from $\text{RdyCS}[i]$;
 - 14: $\text{TST}[i] \leftarrow \max \{\text{TST}[i], \text{TST}[j] + 1\}$;
 - 15: **if** j is a child of i **then**
 - 16: $\text{NoC}[i] \leftarrow \text{NoC}[i] - 1$;
 - 17: **if** i received a message $\text{READY}[j, r_j]$ **then**
 - 18: **if** j is in $\text{CS}[i]$ **then**
 - 19: Add j to $\text{RdyCS}[i]$.
 - 20: Node i transmits data based on the time slot in $\text{TST}[i]$.
 - 21: The topological center transmits aggregated data to the sink.
-

methods could perform worse than $(1 + \varepsilon)R$ for a sufficient small constant $\varepsilon > 0$. At last, we present a general lower-bound on the latency of data aggregation for any algorithm.

A. Performances of Our Algorithm

First we show that, by using Algorithm 2, for each dominator, the number of neighboring dominatees whose data is not collected is reduced by at least 1.

Claim 1: For Algorithm 2, our schedule will ensure that after every time-slot, for each dominator, the number of neighboring dominatees whose data is not collected is reduced by at least 1.

Proof: First, according to Algorithm 1 each dominator u chooses a dominatee randomly from its neighbors and let the chosen dominatee transmits to u . The selected dominatees are called *active transmitters* for the set of selected links. Therefore, there are n links transmitting at the same time-slot. If these n links do not conflict with each other, our claim holds. Here two directed links wv and xy conflict with each other if either receiver y is in the interference range of sender u or receiver v is in the interference range of sender x .

If there exists interference among chosen links, there are only 3 possible cases. For each case, it is easy to show that Algorithm 2 re-schedules these two links to avoid the interference while the number of neighboring dominatees whose data is not collected is reduced by at least 1. ■

Lemma 2: Given a communication graph G of network, under the assumption that the interference range r_I is the same

as the transmission range r , Algorithm 1 (aggregating data from dominatees to dominators) costs at most Δ time-slots where Δ is the maximum node degree in G .

Proof: Assume that there are n dominators $z_i, (i = 1, 2, \dots, n)$ in the network using a method in [14]. Each dominator has at most Δ neighboring dominatees. We define a dominator's degree as the number of the neighboring dominatees whose data has not been aggregated to dominators yet. At first, each dominator's degree is bounded by Δ . Our adjustment (Algorithm 2) repetitively adjusts the set of links whenever there is a pair of conflicting links. Observe that since the recursive nature of our adjustment algorithm, we must prove that our algorithm will terminate in a limited number of rounds. Clearly, when it terminates, there is no pair of conflicting links, *i.e.*, the remaining links from Algorithm 2 are a valid schedule. In addition, we also have to show that when the adjustment terminates, our schedule can ensure that each dominator's degree indeed will be reduced by at least 1.

We define **Loop Invariant** as: every dominator's degree will be decrease at least 1. We can see that loop invariant is always kept before and after one round of execution (which is exactly Algorithm 2) in the loop, To show that Algorithm 2 terminates, we define a **Potential Function** for a schedule as the cardinality of the set $\mathcal{C} = \{(x_1, x_2) \mid x_1, x_2 \text{ are active transmitters and their corresponding links } x_1y_1, x_2y_2 \text{ are conflicting links}\}$. We call the pair $(x_1, x_2) \in \mathcal{C}$ a pair of conflicting transmitters. Clearly, the initial cardinality of the set \mathcal{C} is at most $n(n-1)/2$. After one round of re-scheduling, the interference between at least one pair of conflicting transmitters is resolved. Observe that, our adjustment will not introduce new pairs of conflicting transmitters. Thus the potential function will be decreased by at least 1 after 1 round, which means that Algorithm 2 will terminate after at most $\frac{n(n-1)}{2}$ rounds of execution of the while loop in Algorithm 2. ■

We now bound the number of connectors that a dominator u will use to connect to all dominators within 2-hops. Our proof is based on a technique lemma implied from lemmas proved in [18].

Lemma 3: Suppose that dominator v and w are within 2-hops of dominator u , v' and w' are the corresponding connectors for v and w respectively. Then either $|vw'| \leq 1$ or $|vv'| \leq 1$ if $\angle vuw \leq 2 \arcsin \frac{1}{4}$.

Lemma 4: In Algorithm 3, a dominator requires at most 12 connectors to connect to all dominators within 2-hops.

Proof: Consider any dominator u , let $I_2(u)$ be the set of dominators within 2-hops of u in the original communication network G . Assume that we have already deleted all the redundant connectors for node u . Let C be the set of connectors left for a dominator u . Then for each remaining connector $x \in C$, there is at least one dominator (called a *non-sharing dominator*) that can only use this connector to connect to u (otherwise, connector x is redundant and thus will be removed). Assume there are 13 connectors in C . Then there are at least 13 non-sharing dominators in $I_2(u)$. From pigeonhole principle, we know that there must have 2 dominators v_1 and v_2 such that $\angle v_1uv_2 \leq 2\pi/13 < 2 \arcsin(\frac{1}{4})$. Thus, using Lemma 3, v_1 and v_2 will share a common connector in C , which contradicts to the selection of v_1 and v_2 . ■

In the rest of the proof, for a dominator u , we use $C(u)$ to denote the set of connectors used to connect all dominators in $D_2(u)$.

Lemma 5: In Algorithm 3, a dominator u in level i can receive the data from all neighboring dominators $D_2(u)$ in at most 16 time-slots.

Proof: Each dominator u will collect the aggregated data from all dominators within 2-hops in lower level. Any connector in $C(u)$ has at most 4 other neighboring dominators, besides u . Similar to the proof of Lemma 2, we can show that it takes at most 4 time-slots for each connector to collect data from those neighboring dominators other than u . Recall that at most 12 connectors are needed for u to reach all dominators in $D_2(u)$. Thus, it will take at most 12 time-slots for the dominator u to collect data from all these connectors. Consequently, within at most $12 + 4 = 16$ time-slots, every dominator u can collect the aggregated data from all the dominators in $D_2(u)$. ■

Theorem 6: By using Algorithm 3, the sink can receive all the aggregated data in at most $17R + \Delta - 16$ time-slots.

Proof: Every dominatee's data can be aggregated to a dominator within Δ time-slots from Lemma 2. Observe that every dominator, except the root of the data aggregation tree T , connects to at least one dominator in the upper level within 2-hops. Then Algorithm 3 ensures that every dominator's data can be aggregated upwards the root finally. For each level of BFS, every dominator u including the root of data aggregation tree T , can collect aggregated data from all dominators in $D_2(u)$ within at most 16 time-slots by Lemma 5. Since there is no dominator in level 1, after at most $16(R-1)$ time-slots, every dominator's data can be aggregated to the root. The root then uses at most R time-slots to transmit data to the original sink node via the shortest path. Therefore within $17R + \Delta - 16$ time-slots, all the data can be be aggregated to the sink node. ■

Next, we provide a revised schedule that only need 15 time-slots for dominators in level i ($i \geq 2$) to aggregate data from some dominators within 2-hops, which can also ensure that data will be aggregated to the root finally. This means that we can reduce our latency by $R - 1$ time-slots totally.

For a dominator u other than the root, we denote all dominators within 2-hops of u as $B_2(u)$. Notice that $B_2(u)$ includes at least one dominator v located in upper level of u . By Lemma 4, u needs at most 12 connectors to connect to $B_2(u)$, we denote the set of at most 12 connectors as $C(u)$. There must exists a connector $w \in C(u)$ which connects u to v . Then all dominators in $B_2(u)$ that are connected to w are also 2-hop neighbors of the dominator v , we denote the set of these dominators as $B'_2(u)$, thus $B'_2(u) \subset B_2(v)$. Clearly all data in $B'_2(u)$ can be collected by v , it is not necessary for them to be collected by u . So we let u only collect the data in $B_2(u) \setminus B'_2(u)$. It requires at most 11 connectors (all the connectors in $C(u) \setminus \{w\}$) to connect to the dominators in $B_2(u) \setminus B'_2(u)$. So at most 15 ($= 4 + 11$) time-slots is required for u to aggregate the data from $B_2(u) \setminus B'_2(u)$. If every dominator u other than the root aggregate the data from $B_2(u) \setminus B'_2(u)$, all the data can be aggregated to the root.

Theorem 7: By using Algorithm 3, the sink can receive all

the aggregated data in at most $16R + \Delta - 14$ time-slots.

Proof: Similar to the proof of Theorem 6, we need Δ time-slots for dominators to aggregate data from dominatees. After that, for each level of BFS, every dominator u , other than the root of the data aggregation tree T , can collect aggregated data from all dominators in $B_2(u) \setminus B_2'(u)$ in at most 15 time-slots as stated above. Thus it costs at most $15(R - 2)$ for data to be aggregated to the dominators in level 2. The root r_T can collect the aggregated data from dominators in level 2 within 16 time-slots. Therefore within $15(R - 2) + 16$ time-slots, every dominator's data can be aggregated to the root. The root then transmit the result to the original sink node in R time-slots. In all, within $16R + \Delta - 14$ time-slots, all the data can be aggregated to the sink node. ■

Observe that, although our analysis is based on the centralized method, it is easy to show that all results carry to the distributed implementation (Algorithm 4). Consequently, we have

Theorem 8: By using Algorithm 4, the sink can receive all the aggregated data in at most $16R + \Delta - 14$ time-slots.

V. SIMULATION RESULTS

In this section, we present the simulation results which evaluate our Data Aggregation Algorithms. We implemented DAS on TOSSIM of TinyOS 2.0.2. In our simulation, we randomly deploy a number of wireless sensor nodes in a 2-D square region, all nodes have the same transmission range. Each node will generate a random 16-bits non-negative number as its own datum. The objective of the sink node is to report the aggregation (*max*, *min*, *sum* or *average*) result of all data (totally n data, n is the network size) correctly.

In order to evaluate the efficiency of DAS, we also implemented another data aggregation algorithm by combining BFS tree and CTP (Collection Tree Protocol, which is provided by TinyOS 2.0.2) using TOSSIM. We call this method *BFS+CTP* method for simplicity. The main idea of *BFS+CTP* method is to construct a BFS tree rooted at the sink node based on the link quality. In other words, during the procedure of constructing BFS, the link quality computed by CTP will be considered as the link weight. Notice that, the original CTP method (components) provided in TinyOS 2.0.2 is used to collect data to the sink node. To enable CTP to support data aggregation rather than to collect all data to the sink, we modified CTP in the upper layer such that each node will not send data to its parent (on the BFS tree) until it aggregates all necessary data from all children (on the BFS tree).

We tested and compared the efficiency (including latency and transmission times) for DAS method and *BFS+CTP* method in two different cases. For the first case, we randomly generated the network topology (connected) with different network size (increasing from 30 to 210 with step 30) while keeping the network density unchanged, *i.e.*, the network deployment area increases with the increment of the network size. Actually, by doing this, we fixed the maximum degree Δ (In our simulation, Δ is around 22) for each case, thus the radius of communication graph increases with the increment of network size. The latency performance of two methods, DAS

and *BFS+CTP*, is illustrated in Fig. 2(a). Notice that here, the definition of latency is the time duration from the first datum is transmitted heading for the sink node to the sink node reports the result finally. From the Fig. 2(a), we can see that when the network density is not big, the latency difference between two method is not so big. In most cases, our DAS method has better performance than that of *BFS+CTP*. The radius R for each case is indicated by the value in the brackets right after the network size on x-coordinate. The Fig. 2(b) describes the average transmission times per node with the increment of network size for both methods. Clearly, for both algorithms, the average transmission times are not high (around 1.1) and our method is better than the other in most cases.

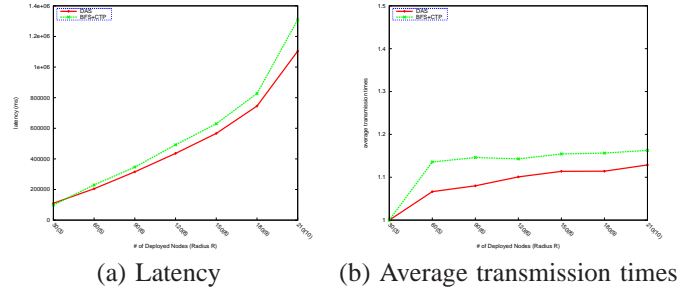


Fig. 2. Simulation results for two methods when Δ is fixed to 22. x -coordinate is the number of deployed nodes, the value in the bracket right after the node number is the radius R of the communication graph.

For the second case, we fix the deployment area as (300×300) and continue to increase the network size from 50 to 200 with step 30 while keeping the network is connected. By doing this, we can fix the radius R and test the performance of both algorithms with the increment of network density (maximum degree Δ). As we can see from Fig. 3, both latency and

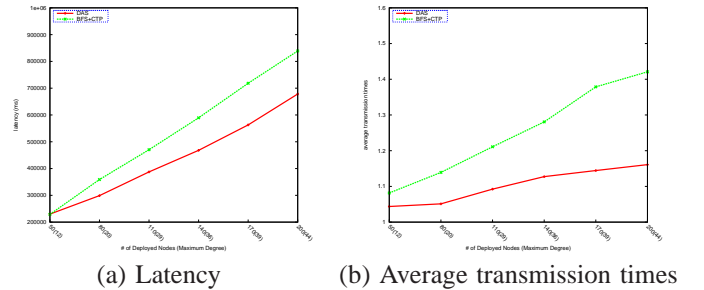


Fig. 3. Simulation results for two methods when the radius R is fixed to 6. x -coordinate is the number of deployed nodes, the value in the bracket right after the node number is the maximum degree Δ of the communication graph.

transmission times have big gaps between these two methods when the density (maximum degree Δ) continues increasing. That is because the interference will be greatly decreased after DAS gather all data to dominators. Hence the total latency and average transmission times decrease significantly. However, for *BFS+CTP* method, the number of relay nodes will continue to increase with the increment of network size such that the latency and average transmission times increase greatly due to the interference. From the simulation results, we can see that in most cases, DAS has better performance

than *BFS+CTP* method. Especially, the denser the network is, the more efficient our DAS algorithm is.

VI. RELATED WORK

Data aggregation in sensor networks has been well studied recently [2], [10], [13], [20]. In-network aggregation means computing and transmitting partially aggregated data rather than transmitting raw data in networks, thus reducing the energy consumption [15].

There are a lot of existing researches on in-network aggregation in the literature [6], [16]. Suppression scheme and model-driven approach were proposed in [5], [7] towards reducing communication cost. The tradeoff between energy consumption and time latency was considered in [20]. A heuristic algorithm for both broadcast and data aggregation was designed in [1]. Another heuristic algorithm for data aggregation was proposed [17], aiming at reducing time latency and energy consumption. Kesselman *et al.* [11] proposed a randomized and distributed algorithm for aggregation in wireless sensor networks with an expected latency of $O(\log n)$. Their method are based on two assumptions: One is that sensor nodes can adjust their transmission range without any limitation. The other is that each sensor node has the capability of detecting whether a collision occurs after transmitting data. Both assumptions pose some challenges for hardware design and is impractical when the network scales. A collision-free scheduling method for data collection is proposed in [12], aiming at optimizing energy consumption and reliability. All these work did not discuss the minimal-latency aggregation scheduling problem.

In addition, the minimum latency of data aggregation problem was proved *NP*-hard and a $(\Delta - 1)$ -approximation algorithm was proposed in [4], where Δ is the maximum degree of the network graph. Another aggregation scheduling algorithm was proposed in [9], which has a latency bound of $23R + \Delta + 18$, where R is the network radius and Δ is the maximum degree. All the algorithms mentioned above are centralized. In many cases centralized algorithms are not practical, especially when the network topology changes often in a large sensor network.

The distributed algorithms for convergecast scheduling were proposed in [8], [11] and [3]. [8], [11] focused on the scheduling problem for data collection in sensor networks. In data collection, since data cannot be merged, the sink must receive N packets from all the nodes, 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. [3] proposed a distributed scheduling algorithm generating collision-free schedules that has a latency bound of $24D + 6\Delta + 16$, where D is the network diameter which is the best result for minimum latency of data aggregation so far.

VII. CONCLUSIONS

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 paper we study the problem of distributed aggregation scheduling in sensor networks and propose a distributed scheduling algorithm with latency bound $16R + \Delta - 14$. This is a nearly constant approximate algorithm which significantly reduces the aggregation latency. The theoretical analysis and the simulation results show that our algorithm outperforms previous algorithms.

REFERENCES

- [1] ANNAMALAI, V., GUPTA, S., AND SCHWIEBERT, L. On tree-based convergecasting in wireless sensor networks. *IEEE WCNC*, 2003.
- [2] BEAVER, J., SHARAF, M., LABRINIDIS, A., AND CHRYSANTHIS, P. Location-Aware Routing for Data Aggregation in Sensor Networks. *Geosensor Networks* (2004).
- [3] BO YU, J. L., AND LI, Y. Distributed Data Aggregation Scheduling in Wireless Sensor Networks. In *submitted for publication* (2008).
- [4] CHEN, X., HU, X., AND ZHU, J. Minimum Data Aggregation Time Problem in Wireless Sensor Networks. *LECTURE NOTES IN COMPUTER SCIENCE* 3794 (2005), 133.
- [5] CHU, D., DESHPANDE, A., HELLERSTEIN, J., AND HONG, W. Approximate data collection in sensor networks using probabilistic models. In *International Conference on Data Engineering (ICDE)* (2006).
- [6] CONSIDINE, J., LI, F., KOLLIOS, G., AND BYERS, J. Approximate aggregation techniques for sensor databases. In *Data Engineering, 2004. Proceedings. 20th International Conference on*, pp. 449–460.
- [7] DESHPANDE, A., GUESTRIN, C., HONG, W., AND MADDEN, S. Exploiting Correlated Attributes in Acquisitional Query Processing.
- [8] GANDHAM, S., ZHANG, Y., AND HUANG, Q. Distributed Minimal Time Convergecast Scheduling in Wireless Sensor Networks. In *IEEE ICDCS* (2006).
- [9] HUANG, S., WAN, P., VU, C., LI, Y., AND YAO, F. Nearly Constant Approximation for Data Aggregation Scheduling in Wireless Sensor Networks. In *IEEE INFOCOM* (2007), pp. 366–372.
- [10] INTANAGONWIWAT, C., ESTRIN, D., GOVINDAN, R., AND HEIDEMANN, J. Impact of Network Density on Data Aggregation in Wireless Sensor Networks. In *IEEE ICDCS* (2002), vol. 22, pp. 457–458.
- [11] KESSELMAN, A., AND KOWALSKI, D. Fast distributed algorithm for convergecast in ad hoc geometric radio networks. *Journal of Parallel and Distributed Computing* 66, 4 (2006), 578–585.
- [12] LEE, H., AND KESHAVARZIAN, A. Towards Energy-Optimal and Reliable Data Collection via Collision-Free Scheduling in Wireless Sensor Networks. In *IEEE INFOCOM* (2008), pp. 2029–2037.
- [13] MADDEN, S., FRANKLIN, M., HELLERSTEIN, J., AND HONG, W. TAG: a Tiny Aggregation Service for Ad-Hoc Sensor Networks. Proceedings of the USENIX Symposium on Operating Systems Design and Implementation, 2002.
- [14] P.-J. WAN, K. A., AND FRIEDER, O. Distributed construction of connected dominating set in wireless ad hoc networks. In *IEEE INFOCOM* (2002).
- [15] SHARAF, M., BEAVER, J., LABRINIDIS, A., AND CHRYSANTHIS, P. TINA: a scheme for temporal coherency-aware in-network aggregation. In *Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access* (2003), ACM New York, NY, USA, pp. 69–76.
- [16] SHRIVASTAVA, N., BURAGOHAIN, C., AGRAWAL, D., AND SURJ, S. Medians and beyond: new aggregation techniques for sensor networks. In *ACM SenSys* (2004), pp. 239–249.
- [17] UPADHYAYULA, S., ANNAMALAI, V., AND GUPTA, S. A low-latency and energy-efficient algorithm for convergecast in wireless sensor networks. In *IEEE GLOBECOM* (2003), vol. 6.
- [18] WAN, P.J. AND YI, C.W. AND JIA, X. AND KIM, D. Approximation algorithms for conflict-free channel assignment in wireless ad hoc networks, *Wiley Wireless Communication and Mobile Computing* (2006), vol. 6, pages 201.
- [19] WANG, W. AND WANG, Y. AND LI, X.Y. AND SONG, W.Z. AND FRIEDER, O. Efficient interference-aware TDMA link scheduling for static wireless networks, In *IEEE MOBICOM* (2006), pp. 262–273.
- [20] YU, Y., KRISHNAMACHARI, B., AND PRASANNA, V. Energy-latency tradeoffs for data gathering in wireless sensor networks. In *IEEE INFOCOM* (2004), vol. 1.