Fast Aggregation Scheduling in Wireless Sensor Networks

Hamed Yousefi, Marzieh Malekimajd, Majid Ashouri, and Ali Movaghar, *Senior Member, IEEE*

***Abstract*—Data aggregation is a key, yet time-consuming functionality introduced to conserve energy in wireless sensor networks (WSNs). In this paper, to minimize time latency, we focus on aggregation scheduling problem and propose an efficient distributed algorithm that generates a collision-free schedule with the least number of time slots. In contrast to others, our approach named FAST mainly contributes to both tree construction, where the former studies employ Connected 2-hop Dominating Sets, and aggregation scheduling that was previously addressed through the Competitor Sets computation. We prove that the latency of FAST under the protocol interference model is upper-bounded by $12R + \Delta - 2$, where R is the network radius and Δ is the maximum node degree in the communication graph of the original network. Both the theoretical analysis and simulation results show that FAST outperforms the state-of-the-art aggregation scheduling algorithms.**

***Index Terms*—Wireless sensor networks, data aggregation, minimum latency scheduling.**

I. Introduction

ENERGY efficiency is an important issue for data collection in WSNs. Data aggregation is a fundamental operation aiming to conserve energy by reducing the number of packet transmissions through the network [3], [12], [16], [19]. To aggregate data, every intermediate node combines all the received data with its own record into a single packet according to some aggregation functions, and then forwards it to upper nodes on a spanning inward architecture rooted at the sink.

Besides the energy efficiency, the requirement of real-time communication is becoming more and more important in emerging applications [10], [24], [27]–[29]. Here, outdated information would be irrelevant and even lead to negative effects on the system monitoring and control. Thus, it is crucial to provide a guarantee on the delivery time and consider the total latency involved in aggregating data. In a previous study [1], we addressed the problem of maximizing the quality of data aggregation in a real-time, tree-based WSN. Indeed, we investigated how to maximize both the number of source participating nodes and their spatial dispersion subject to a deadline constraint. However, here, we focus on another fundamental question stated as: how fast can information be aggregated from a tree-based WSN rooted at the sink of aggregation?

Manuscript received April 13, 2014; revised September 8, 2014; accepted February 4, 2015. Date of publication February 20, 2015; date of current version June 6, 2015. The associate editor coordinating the review of this paper and approving it for publication was J. Huang.

The authors are with the Department of Computer Engineering, Sharif University of Technology, Tehran 11365-9363, Iran (e-mail: hyousefi@ce.sharif.edu; malekimajd@ce.sharif.edu; ashuri@ce.sharif.edu; movaghar@sharif.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TWC.2015.2405060

Communication collision is a primary reason for long latency in data aggregation. It occurs when two or more sensor nodes transmit data to a common node at the same time. Unfortunately, the collision problem is normally left to the MAC layer, which could incur a large amount of energy consumption and time latency during aggregation. Hence, we concentrate on the TDMA scheduling problem above the MAC layer, while the main objectives are to minimize aggregation latency and ensure the collision-free transmissions. This is known as the Minimum Latency Aggregation Scheduling (MLAS) problem in the literature [15], [22], [25], where the latency of a schedule is the number of time slots needed to aggregate data from the sensor nodes to the sink. It is worth mentioning that in this paper our focus is on the basis of MLAS itself, where the protocol interference model is employed as in the literature.

The MLAS problem was proved to be NP-hard and an approximation algorithm was proposed in [5]. It investigates the fundamental question of how the aggregation transmissions can be scheduled in a tree-based WSN, such that no collision may occur and the total number of time slots is minimized. To achieve this, the MLAS problem is typically approached in two steps: (i) data aggregation tree construction, and (ii) link transmission scheduling. Recently, there has been a surge of interest in studying this problem and several approximation algorithms have been proposed generally classified into two categories: centralized and distributed. Although the centralized approaches [5], [11], [21] have an important contribution in theory, they are not practical in real applications, where the network suffers from frequent topology changes (e.g., node failures); therefore, the sink has to gather new network topology information, recompute a schedule, and disseminate it into the network. These processes waste lots of energy resources, which make centralized algorithms inherently inefficient.

To overcome the above-mentioned problems, the distributed algorithms are employed to construct an aggregation tree and then schedule transmissions [15], [22], [25]. However, the time latencies of the existing scheduling techniques are still high. In this paper, we propose FAST, a collision-Free minimum latency Aggregation Scheduling algorithm for Tree-based WSNs. We make the following four main contributions.

1) We construct a novel data aggregation tree which is different from the commonly used Connected 2-hop Dominating Sets-based approaches. Furthermore, a key design feature which is simultaneous execution of aggregation tree construction and scheduling is employed.
2) We propose an efficient distributed collision-free TDMA schedule, such that the aggregation latency is minimized. Our algorithm differs from all the previous schemes in which Competitor Sets are computed.

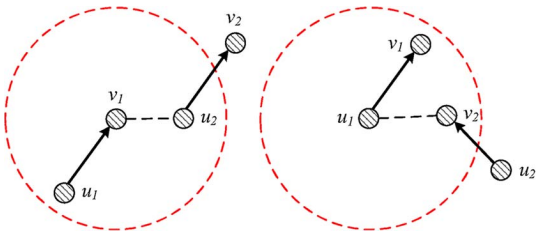


Fig. 1. Two cases of a collision.

3) We theoretically prove a new upper bound for aggregation latency at $12R + \Delta - 2$ time slots under the protocol interference model. To the best of our knowledge, FAST has so far had the minimum latency for data aggregation in tree-based WSNs.
4) We conduct extensive simulations to evaluate the performance of FAST. The simulation results show that, besides the latency upper bound, FAST is the current best distributed algorithm for the MLAS problem in the literature.

The rest of the paper is organized as follows. Section II formulates the MLAS problem. Section III outlines the related work. Our proposed algorithm is presented in Section IV and is analyzed in Section V. Simulation results are presented and discussed in Section VI. Finally, Section VII concludes our work and discusses some future directions.

II. Problem Definition

We consider that sensor nodes are distributed randomly into a two-dimensional area in a static WSN, and there exists one sink node that collects information from the sensors. The transmission coverage of any node is a circle of radius r. Let $G(V, E)$ be the topology graph of the network, where V is the set of all the nodes and E is the edge set of G. An edge $(u, v) \in E$ indicates that u and v can form a communication link if and only if u is in the transmission area of v (i.e., $|u - v| \leq r$). We also assume that G is connected.

As in the state-of-the-art [15], [22], [25], we consider the interference modeled by protocol interference model [9] in our analysis. In each time slot, a sensor node can either send or receive data. Furthermore, if a node hears more than one message at the same time, it can receive none of them correctly, so this causes a collision. Formally, two transmissions $u_1 \rightarrow v_1$ and $u_2 \rightarrow v_2$ are conflicting whenever $|u_1 - v_2| \leq r_I$ or $|u_2 - v_1| \leq r_I$, where $r_I \geq r$ is the interference range. In the rest of the paper, to simply analyze the time latency as in the state-of-the-art [15], [22], [25], we will assume that $r_I = r = 1$, i.e., the underlying communication graph is a Unit Disk Graph (UDG). Fig. 1 shows two cases in which collisions occur at v_1 and v_2, respectively.

An approximation of MLAS is a sequence of sender sets $U_1, U_2, \ldots, U_l$, such that data packets are simultaneously transmitted from the senders in U_i to their parents in time slot i, for each $i = 1, 2, \ldots, l$, and finally aggregated to the sink node v_s in l time slots. The sets satisfy the following conditions:

1) $U_i \bigcap U_j = \emptyset, \forall i \neq j$.
2) $\bigcup_{i=1}^{l} U_i = V - v_s$.
3) Data are aggregated from U_k to $V - \bigcup_{i=1}^{k} U_i$ in time slot $k, \forall k = 1, 2, \ldots, l$.

The MLAS object is to suggest a sequence of collision-free schedules $U_1, U_2, \ldots, U_l$, such that the total aggregation latency l is minimized.

It is worth mentioning that the root of the aggregation tree is assumed to be the sink node v_s in general. However, to reduce the total aggregation latency to a function of network radius R instead of network diameter D, as in the previous studies [15], [22], we choose the topology center of a given UDG as the root node. Note that in most networks, the topology center is different from the real sink node. Here, $v_c = \text{argmin}_v\{\max_u\{d_G(u, v)|u \in V\}|v \in V\}$ is called the topology center, where $d_G(u, v)$ denotes the hop distance between u and v in communication graph $G(V, E)$. Moreover, the maximum distance between v_c and any other node v is the network radius, i.e., $R = \max_v\{d_G(v_c, v)|v \in V\}$. As in the state-of-the-art [15], [22], we assume that v_c is known and static. After the data are aggregated with minimum latency from all nodes in v_c, it will forward the result to the sink node v_s via the shortest path. This will impose an additional delay $d_G(v_c, v_s)$ of at most R time slots. Note that we relax the requirement $U_i \bigcap U_j = \emptyset$, $\forall i \neq j$ for the case when the topology center is different from the real sink node. When the sets $U_i, 1 \leq i \leq l$ are not disjoint, in the actual data aggregation, a node which appears multiple times in $U_1, U_2, \ldots, U_l$ (it just happens on the shortest path from v_c to v_s) will participate in the data aggregation only once (say the smallest i when it appears in U_i), and then it will only serve as a relay node in later appearances.

III. Related Work

Aggregation is an important and essential operation in many WSN applications [2], [6], [18], [23], [26]. Recently, there has been a surge of interest in studying the time latency and collision prevention problem for efficient aggregation scheduling in WSNs. The relevant works for the MLAS problem are classified into two categories: centralized and distributed.

Chen *et al.* [5] proved the MLAS problem to be NP-hard and proposed an approximation algorithm with a ratio of Δ. Huang *et al.* [11] designed a centralized aggregation scheduling method based on Connected Dominating Sets (CDS) with the latency bound $23R + \Delta - 18$. Wan *et al.* [21] proposed three algorithms to further improve the aggregation time bound up to $15R + \Delta - 4$, $2R + O(\log R) + \Delta$ and $(1 + O(\log R/\sqrt[3]{R}))R + \Delta$, respectively. Although the centralized approaches have an important contribution in theory, they are not practical in WSNs, especially when the network topology changes frequently.

Realizing the disadvantages of the centralized approaches, Yu *et al.* [25] proposed a new distributed algorithm, proved that there is no collisions in their aggregation scheme, and computed a new upper bound at $24D + 6\Delta + 16$. Xu *et al.* [22] designed a distributed aggregation scheduling method generating collision-free schedules which guarantee that the aggregation time does not exceed $16R + \Delta - 14$. They also derived the overall delay lower bounds of $\max\left\{R, \frac{\Delta\left\lfloor \arcsin\frac{r_I - r}{2r_I}\right\rfloor}{2\pi}\right\}$ and $\max\{R, \Delta\}$ under the protocol interference model when $r < r_I < 3r$ and $r_I \geq 3r$, respectively. Both [22] and [25] employ a distributed approach of constructing Connected 2-hop

Dominating Sets (C2DSs), where the distance between any pair of nodes in a maximal independent set is exactly two hops [20]. More recently, a distributed scheduling algorithm on a cluster-based constructed tree has been developed in [15] to minimize scheduling with an upper bound on delay of $4R + 2\Delta - 2$; however, we proved that this bound is incorrect (this will be discussed in Section V-C in detail). To achieve a distributed collision-free aggregation scheduling in all previous works, each node needs to determine its Competitor Set. Given a node u in a constructed aggregation tree T, Competitor Set $CS(u)$ consists of those nodes which cannot transmit at the same time as u due to the collision. It is formally denoted as $CS(u) = N(p(u)) \bigcup (\bigcup_{v \in N(u) \setminus CHD(u)} CHD(v)) \setminus \{u, p(u)\}$, where $p(u)$, $CHD(u)$, and $N(u)$ are u's parent in T, u's children set in T, and u's 1-hop neighbor set, respectively.

In contrast to others, our distributed approach mainly contributes to both tree construction and aggregation scheduling, where the latency is upper-bounded by $12R + \Delta - 2$.

Some previous literatures [4], [13], [14], [17] have studied data aggregation under the more realistic physical (also known as SINR) interference model, yet by incurring a higher computational complexity. Moreover, many recent works consider not only latency but also throughput under a joint optimization framework. For instance, a multi-channel design is employed in [7], [8] as a promising technique to alleviate interference (which is the primary reason for long latency of aggregation scheduling). Furthermore, in recent years there has been a surge of interest in maximizing the quality of data aggregation under delay constraints by establishing bi-objective optimization problems [1], [10]. However, the importance of the MLAS problem itself motivated us to focus on optimizing the basis, where the protocol interference model is employed.

IV. FAST

A key contribution behind FAST is to construct a balanced Connected 3-hop Dominating Sets (C3DS)-based structure. The result is a tree containing some dominators, called `Parent`, among which transmissions are concurrent and collision-free; thus, avoiding conflicting transmissions can reduce the aggregation latency. Next, FAST decides a novel collision-free aggregation schedule in a distributed way. Another key (and unique) design feature is that the aggregation tree construction is partly simultaneous with the execution of aggregation scheduling. It is against the previous approaches where tree construction and aggregation scheduling are two consecutive and separated phases for the MLAS problem. In this section, we present the processes of 3-hop dominating sets selection, incomplete tree construction, and aggregation scheduling, respectively.

A. 3-Hop Dominating Sets Selection

Here, we propose a special method to select a 3-hop dominating set and assign different roles to nodes. In a graph $G(V, E)$, a subset S of V is a 3-hop dominating set if for each node u in V, it is either in S or at a distance of at most 2 hops from a node v in S. Nodes from S (dominators) and the others adjacent to them are assigned `Parent` and `Child` roles, respectively. The remaining nodes are identified as Grand-childs (`Gchilds`) constituting a part of the leaves in the aggregation tree.

Each node u has a specific rank $(level_u, ID_u)$, given by the ordered pair of its level (i.e., the smallest hop distance from u to center v_c in G) and its ID. Such rankings are compared using a lexicographic order. Furthermore, each node u has an array list L_u of all its lower-ranked neighbors within two hops. It is used to guarantee that dominators are separated by at least 3 hops.

The details of 3-hop dominating set selection are shown in Algorithm 1. A node takes `Parent` role if all its lower-ranked neighbors within two hops have been removed from L_u (i.e., $L_u = \emptyset$). Definitely, the algorithm starts from center v_c, where $L_{v_c} = \emptyset$. Each node u taking $role_u$ broadcasts a message $DSC(role_u, ID_u, ID_x)$ to inform its neighbors of its role. This message can also be used to remove both nodes u and x from u's higher-ranked non-role neighbors' lists (we employ this to remove `Child` and `Gchild` nodes from their 2-hop-away and 1-hop-away neighbors' lists, respectively). A message $DSC2(None, ID_u, ID_x)$ is also used only to remove node x from u's higher-ranked non-role neighbors' lists (we employ this to remove `Gchild` nodes from their 2-hop-away neighbors' lists). According to the last part of Algorithm 1, `Gchilds` are the only nodes which can change their roles (to `Childs`) during the role assignment process, where NRN_u is the number of u's non-role neighbors. This happens if a `Gchild`'s none-role neighbor takes `Parent` role. The algorithm finishes when each node u is assigned a role (i.e., $role_u =$ `Parent`/`Child`/`Gchild`) and also knows all its neighbors roles. All `Parent` nodes (dominators) form a 3-hop dominating set. The rest of the nodes are `Childs`, as the adjacent nodes to the dominators, and `Gchilds`.

Algorithm 1: 3-hop Dominating Set Selection and Role Assignment

```
while role_u = None do
    if L_u = ∅ then
        Set its role to Parent;
        Send message DSC(Parent, ID_u, ∅) to N(u);
        Break;
    Receive a message M
    if M = DSC(role, ID1, ID2) then
        if role = Parent then
            Set its role to Child;
            Send message DSC(Child, ID_u, ID1) to N(u);
        if role = Child then
            Set its role to Gchild;
            Send message DSC(Gchild, ID_u, ID1) to N(u);
        if role = Gchild then
            Remove both ID1 and ID2 from L_u;
            Send message DSC2(None, ID_u, ID1) to N(u);
    if M = DSC2(role, ID1, ID2) then
        Remove ID2 from L_u;
if role_u = Gchild then
    while NRN_u ≠ 0 do
        Receive a message M
        if M = DSC(role, ID1, ID2) then
            Decrease NRN_u by 1;
            if role = Parent then
                Set its role to Child;
                Send message DSC(Child, ID_u, ID1)
                to N(u);
                Break;
```

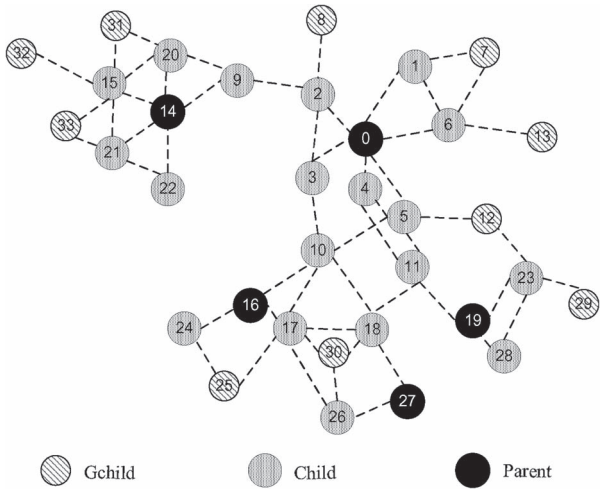


Fig. 2. Assigned roles in an example graph.

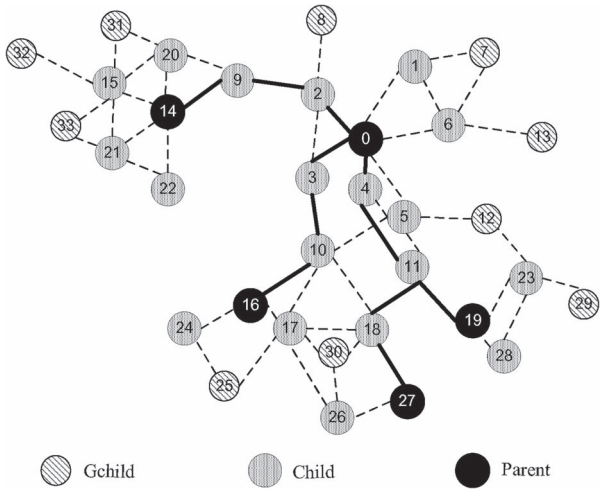


Fig. 3. Constructed backbone in the example graph.

The following lemma shows the property of Parent nodes.

Lemma 1: There are at least 3 hops between any two Parent nodes in a 3-hop Dominating Set.

Proof: Consider a node P becomes Parent during the role assignment process. Without loss of generality, suppose that P includes in L_u (lower-ranked neighbors) of a non-role node u within two hops. According to Algorithm 1, u changes its role to Parent if $L_u = \emptyset$; however, the instructions in our pseudo code can only remove Childs and Gchilds, not a Parent node (like P), from L_u. Therefore, u, as a P's 2-hop-away neighbor, cannot take Parent role. □

Generally, according to the algorithm, for any Parent, its 1-hop neighbors are assigned Child role, while the 2-hop-away neighbors are Gchilds (unless a Gchild changes its role to Child during the role assignment phase if one of its none-role neighbors takes Parent role). Fig. 2 shows the assigned roles for an example network graph, where the number inside each circle is the node ID. Node 20 is the sink and Node 0 is the network center. The dashed lines represent edges in the communication graph.

B. Incomplete Tree Construction

The structure of an incomplete aggregation tree is formed in two phases: First, by concatenating dominators, a Connected 3-hop Dominating Set (C3DS) is constructed as the virtual backbone of a WSN. Second, the Gchild nodes join the structure.

The backbone is actually a subgraph induced by Parent nodes as dominators. The Parents are then connected by some Child members to construct the network backbone. A Parent always has at least one lower-level Parent in exactly 3 hops and chooses the lowest-ranked one as the next Parent node in its route toward v_c.

Lemma 2: Each Parent (except the root) has at least one lower-level Parent in its 3 hops toward v_c.

Proof: Without loss of generality, suppose that u is a Parent node at level k. It is evident that u has a neighbor (say v) at level $k-1$. Unless v has a role during the role assignment process, it has a higher priority (lower rank) than u and picks up the Parent role instead. v cannot be a Child of any other higher-ordered Parent$w$, but can be its Gchild in a 2-hop distance. Finally, for each Parent$u$, there is at least one lower-level Parent$w$ exactly in 3 hops. Moreover, after u being a Parent, v changes its role to Child; therefore, two Parents are connected through two Childs to form the backbone. □

Here, to build a parent-child relationship in the backbone, each Parent$u$ searches for another Parent (say w) whose level (and then ID for two nodes at the same level) is the smallest among all the lower-ranked Parents in its 3 hops. To achieve this, each Child$c$ of a Parent$u$ sends message BA (Backbone Announcement) including u's ID and u's level to all its Child neighbors. It also receives BAs from the other Childs, but only forwards the lowest-ranked one to its own Parent. Finally, among all BAs received from u's Childs, it chooses Parent$w$, then sends message BBC (BackBone Construction) through the backward way to set up the backbone. Furthermore, to distinguish the nodes in the backbone from all the others, they set a binary variable $BFlag$ to one (i.e., $BFlag = 1$).

Fig. 3 illustrates the constructed backbone in the example communication graph, where solid lines are the edges in final tree T.

In the second phase, Gchilds join the structure. Distributing Gchilds connections among their neighbor Childs for an efficient tree construction strongly helps the reduction of aggregation latency by providing a higher potential of parallel transmissions. Fig. 4 shows an example graph, where the number in brackets is the target time slot. As is evident from the figure, node c can send the aggregated result after 3 slots in the balanced structure in Fig. 4(b) instead of 5 slots in Fig. 4(a).

Here, to construct a balanced structure and reduce aggregation latency, each Gchild$u$ broadcasts a message including the number of adjacent Childs and waits to receive replies from the neighbor Childs. Then, u selects the Child node with the minimum number of common Gchild neighbors as its parent (if it is equal for some Child nodes, one with the minimum ID has a higher priority). For example, Gchild 30 in Fig. 5 has three choices (Childs 17, 18, and 26) to select one as its parent. Among these nodes, Child 17 has another common Gchild neighbor and Child 26 has the highest ID. Thus, Gchild 30

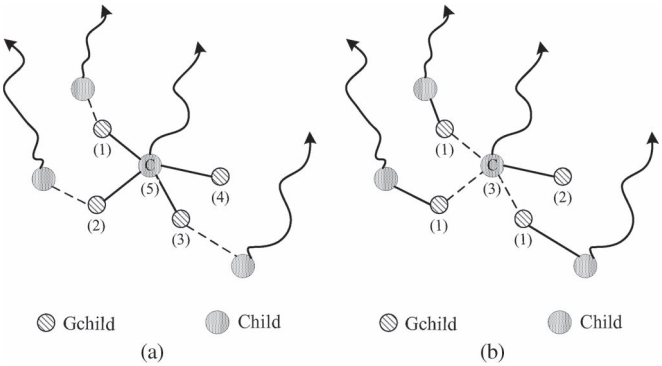


Fig. 4. An example schedule (a) unbalanced structure (b) balanced structure.

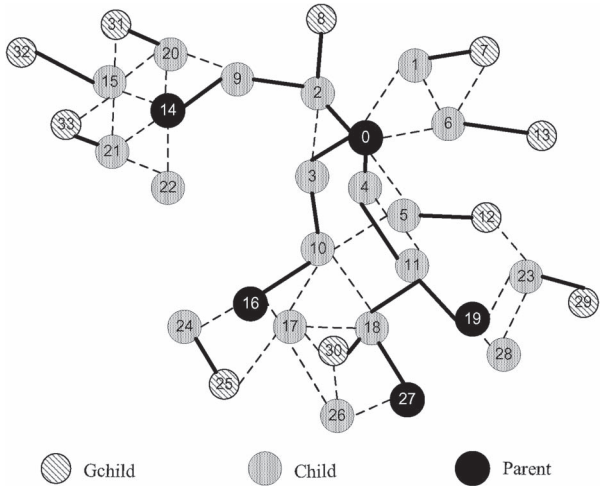


Fig. 5. `Gchilds` join the structure in the example graph.

connects to `Child` 18 as its parent. All the `Gchild` nodes become leaves in the aggregation tree. Fig. 5 illustrates our incomplete tree after the `Gchilds` join the structure.

To achieve much of the potential of parallelism in data transmissions, the not-in-the-backbone `Childs` (i.e., those having $BFlag = 0$) join the structure during the schedule assignments (see Section IV-C). Hence, a key design feature of FAST is that completing the tree construction is simultaneous with the execution of aggregation scheduling.

C. Aggregation Scheduling

In this section, we design a special collision-free aggregation scheduling algorithm simultaneous with completing the tree construction. We set up the complete tree and extract aggregate data information from a WSN with minimum latency to the sink. In contrast to others, FAST mainly contributes to aggregation scheduling that was previously addressed through Competitor Sets computation in each node. In our approach, considering that collisions occur at the receivers, each node negotiates with its parent in the aggregation tree. Here, parents are responsible for scheduling their children and deciding on time slots for transmissions.

In our proposed algorithm, each node schedules their children according to their priorities. First, `Gchilds` are scheduled.

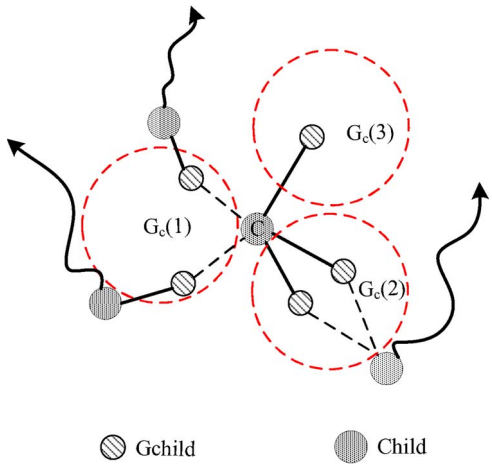


Fig. 6. Different types of `Gchilds`.

Next, FAST schedules not-in-the-backbone `Childs`. Third, the backbone is scheduled. The scheduling is actually a bottom-up process from `Gchilds` to the topology center and is assigned to the network nodes in three different phases:

1) In the first phase, all `Gchilds` are scheduled. According to the `Gchilds` connection policy, each `Child`c may have three types of neighbor `Gchilds` in the vicinity which are grouped into sets $G_c(1)$, $G_c(2)$, and $G_c(3)$ (where $G_c = \bigcup_{i=1}^{3} G_c(i)$ is the set of c's neighbor `Gchilds`) as shown in Fig. 6.

- Set $G_c(1)$ includes those c's neighbor `Gchilds` which are the children of some other `Childs` in the data aggregation structure.
- Set $G_c(2)$ includes those c's children `Gchilds` which are in the vicinity of some other `Childs` in the data aggregation structure.
- Set $G_c(3)$ includes those c's children `Gchilds` which are in the vicinity of no other `Child` in the data aggregation structure.

FAST makes each `Child`c wait for the first type of neighbor `Gchilds` (i.e., $G_c(1)$) to be allocated transmission slots by the other `Childs`, before it starts the assignment of schedules to its own children. Then, node c assigns time slots to $G_c(2)$ and $G_c(3)$ as its children, respectively. As mentioned before, this mainly results in the potential of a large number of parallel transmissions in common `Gchilds` (i.e., $G_c(1)$ and $G_c(2)$), thus minimizing aggregation latency.

Theorem 3: Given a set of `Childs` $\{c_1, c_2, \ldots, c_n\}$, the waiting policy for scheduling of the first type of their neighbor `Gchilds` is loop-free.

Proof: Here, we prove the theorem by contradiction. Suppose that each c_i waits for scheduling a common `Gchild` neighbor by c_{i+1} (as denoted by $c_i \leq c_{i+1}$). We know that this is true if c_{i+1} has a higher priority (i.e. fewer number of common `Gchilds` or a smaller ID with the same number of common `Gchilds`) than c_i. At the end, node c_n waits for c_1. Loop $c_1 \leq c_2 \leq \ldots \leq c_n \leq c_1$ needs all `Childs` to have the same number of common `Gchilds` ($c_1 = c_2 = \ldots = c_n = c_1$), but again we have $ID_{c_1} > ID_{c_2} > \ldots > ID_{c_n} > ID_{c_1}$ that contradicts the assumption. □

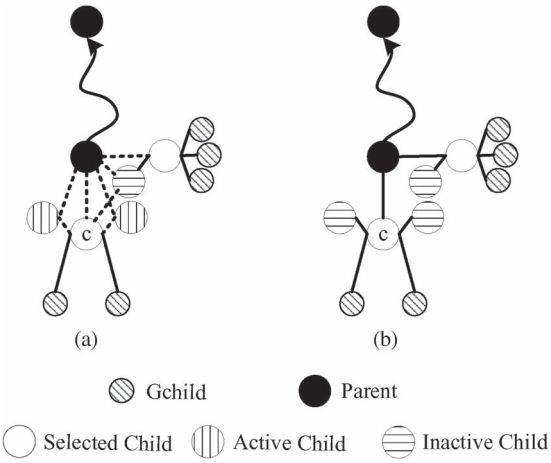


Fig. 7. Different types of Childs in (a) incomplete structure (b) complete structure.

To provide a collision-free schedule, each Child$c$ maintains an array of size Δ (as the maximum number of neighbors) to record the unavailable time slots (those allocated to the nodes in the vicinity). As a matter of fact, a transmission from any neighbor node at unavailable time slots will lead to a collision in c. Thus, c searches in its array and assign the available transmission slots to its own children in an efficient manner. Indeed, the first available slot is assigned to the lowest-ranked child. In this way, we try to use all vacant slots, thus minimizing aggregation delay.

2) In the second phase, not only are all not-in-the-backbone Childs scheduled, but also they join the structure. Each Parent$u$ maintains an array to record the last time slots (T_{final}) that its Childs have assigned to their children. Indeed, when the last transmission slots are successfully determined, the Childs send them to neighbor Parent$u$. Next, u selects a neighbor c in its array with the maximum $\{T_{final}(c), c \in u'$s neighbor Childs$\}$ and picks it up as its child by sending a message. Once c receives the message, it tries to find all the active u's neighbor Childs in its vicinity (by active, we mean those Childs that have not already been scheduled by the others and wait to receive the transmission slots), builds a parent-child relationship, assigns them the time slots exactly after $T_{final}(c)$, and finally inactive them. u also finds to remove it from its array. It keeps selecting the next Childs with the maximum T_{final}, completing tree construction, and assigning time slots. Finally, according to the Childs connection policy, each Parent$u$ may have two types of neighbor not-in-the-backbone Childs which are grouped into sets $C_u(1)$ and $C_u(2)$ (see Fig. 7).

- Set $C_u(1)$, named selected Childs, includes those u's neighbor Childs which join u as their parent in the data aggregation structure.
- Set $C_u(2)$, named non-selected Childs, includes those u's neighbor Childs which join the other selected Childs as their parent in the data aggregation structure.

In this way, we bound the number of u's connections (selected Childs) by 5 (see Section V), so reduce aggregation latency.

3) In the third phase, the backbone is scheduled. Each node u in the backbone starts to schedule all its children when (1) all u's children are in state $Ready$ and (2) u receives schedule information from all its higher-ranked neighbors except its children and its parent. As a matter of fact, u cannot transmit data at the same time as its neighbors' children. It also needs to assign time slots, different from the neighbors' schedules, to its own children. Once u finalizes its children scheduling, it goes to state $Ready$ and requests scheduling. As mentioned, each node waits as long as there is a higher-ranked node which has not yet been scheduled in the vicinity. This makes a collision-free data aggregation, where each node knows the unavailable slots in its neighborhood.

To ensure a loop-free waiting policy, we assign new ranks to Child nodes in the backbone. Since Parents have no neighbor in the backbone except their parents and children, they just keep their previous ranks. We know that each Parent$u$ will progressively route data to another one (say Parent$w$) in exactly 3 hops toward v_c. According to the backbone construction policy, in the worst case, u and w are at two consecutive levels (i.e., $level_u - level_w = 1$). Thus, new ranks $(level_u - \frac{1}{3}, ID_u)$ and $(level_u - \frac{2}{3}, ID_u)$ are assigned to the two intermediate connectors (i.e., Child c_2 and Child c_1, respectively, in a tuple $\langle u, c_2, c_1, w\rangle$).

Theorem 4: Given a set of Childs $c_1, c_2, \ldots, c_n$ in the network backbone, the waiting policy in scheduling is loop-free.

Proof: Here, we prove that there is no loop in our waiting process by contradiction. Suppose that there exists a loop $\langle c_1, c_2, \ldots, c_n, c_1\rangle$, where Child c_{i+1} waits for Child c_i for $1 \leq i < n$. It means both $rank(c_1) < rank(c_n)$ and $rank(c_1) > rank(c_n)$ are true. Definitely, this contradicts the assumption because even if the levels are equal for different nodes, they have different IDs. □

In this phase, without loss of generality, we first remove all the not-in-the-backbone nodes from communication graph G and then start to schedule the backbone. In the backbone, leaves have Parent role. According to the C3DS-based tree construction approach, these leaves have no neighbor except their parents. Hence, the process starts without any waiting in leaves. Totally, to produce a collision-free schedule, each node u in the backbone sends both $T_{final}(u)$ and $ST(u)$ to all its neighbors, where $T_{final}(u)$ is the last time slot allocated to u's children and $ST(u)$ denotes the schedule time targeted by u. It also waits to receive the target schedules from all its higher-ranked neighbors to successfully determine the applicable transmission slots for its children. This provides the potential of consecutive slot assignment to the children. Moreover, u needs all of the last schedules of its neighbors to compute new $T_{final}(u)$. Finally, it goes to state $Ready$, sends $T_{final}(u)$ to its parent, and requests scheduling. For each child node u of parent p, $ST(u)$ is calculated as $ST(u) = Min(t : t > T_{final}(u)\ and\ t \in N^* - US(p))$, where $US(p)$ is the set of unavailable transmission slots in p. The scheduling process ensures that the time slot assigned to each node is always larger than those of all its children.

Algorithm 2 concludes FAST distributed three-phase scheduling for a collision-free time slot assignment. It shows

Algorithm 2: Aggregation Scheduling and Tree Construction

```
if u is a Gchild then
    Send message Ready(u, T_final(u)) to its parent;
    while (ST(u)= Null) do
        Nothing;
    Send message Gcomplete(u, ST(u)) to all its neighbors;
if u is a Child then
    while (NGchild_1(u) > 0) do
        Receive message Gcomplete(v, ST(v)) from a neighbor Gchild v
            Record the schedule time and decrease NGchild_1(u) by 1;
    Start scheduling all its Gchild children;
    if u is not in the backbone then
        Send message Ready(u, T_final(u)) to its parent;
        while (ST(u)= Null) do
            Receive MAXChild(u)
                Connect to all its active Child neighbors and starts
                scheduling as well as inactivating them;
                Update T_final(u) and send message
                Ready2(u, T_final(u)) to its Parent;
        Send message Ccomplete(u, ST(u)) to all its neighbors;
    else
        while (ST(u)= Null) do
            Receive a message M
                if M = Bcomplete(v, ST(v), T_final(v)) from a higher-
                ranked neighbor v then
                    Record the schedule times and decrease HRN(u) by
                    1;
                if M = Ready3(v, T_final(v)) from its child in the backbone
                then
                    Decrease Nbchild(u) by 1;
            if HRN(u) = 0 and Nbchild(u) = 0 then
                Start scheduling all its children in the backbone, update
                T_final(u), and send message Ready3(u, T_final(u)) to
                its parent;
        Send message Bcomplete(u, ST(u), T_final(u)) to all its neigh-
        bors;
if u is a Parent then
    Set MAXflag(u) = False;
    while NChild > 0 do
        if MAXflag(u) = True then
            Find a neighbor Child v having MaxT_final(v),
            send it MAXChild(v), and set MAXflag(u) =
            False;
        Receive a message M
            if (M = Ccomplete(v, ST(v)) from a non-selected
            Child v) then
                Record the schedule time, remove v from its list, and
                decrease NChild by 1;
            if M = Ready2(v, T_final(v)) from a selected Child v
            then
                Set MAXflag(u) = True and decrease NChild(u)
                by 1;
            if M = Ready(v, T_final(v)) from a neighbor not-in-the-
            backbone Child v then
                Decrease NChild(u) by 1;
                if (NChild(u) = 0) then
                    Set MAXflag(u) = True and reset NChild(u);
    Connect to all its selected Childs and start scheduling them;
    while (Nbchild(u) > 0) do
        Receive message Ready3(v, T_final(v)) from its child v in
        the backbone
            Decrease Nbchild(u) by 1;
    Start scheduling all its children in the backbone and send
    message Ready3(u, T_final(u)) to its parent;
```

what exactly each Gchild/Child/Parent node does (as well as sending/receiving control messages) until it is finally scheduled in details. We assume each node u maintains some variables as follows:

- $T_{final}(u)$: the time slot at which the latest u's children will send its data; it is initialized to 0.
- $ST(u)$: the schedule time targeted by node u.
- $NGchild_1(u)$: the number of Gchilds in $G_u(1)$ which are not yet scheduled.
- $NChild(u)$: the number of u's Child neighbors which are not in the backbone of communication graph G.
- $Nbchild(u)$: the number of u's children in the backbone of communication graph G.
- $HRN(u)$: the number of u's higher-ranked neighbors except its parent and
- its children in the backbone of communication graph G.

Algorithm 2 outputs a complete tree in which each node has allocated a transmission slot. Fig. 8 shows the complete aggregation tree as well as the assigned schedules in the example graph, where the number in each bracket is the target time slot. As is evident from the figure, the network center (i.e., Node 0) can gather all the data in 9 time slots. Finally, it sends the final results to the sink via the shortest path. It is worth mentioning that we further describe the algorithm while analyzing the latency bound.

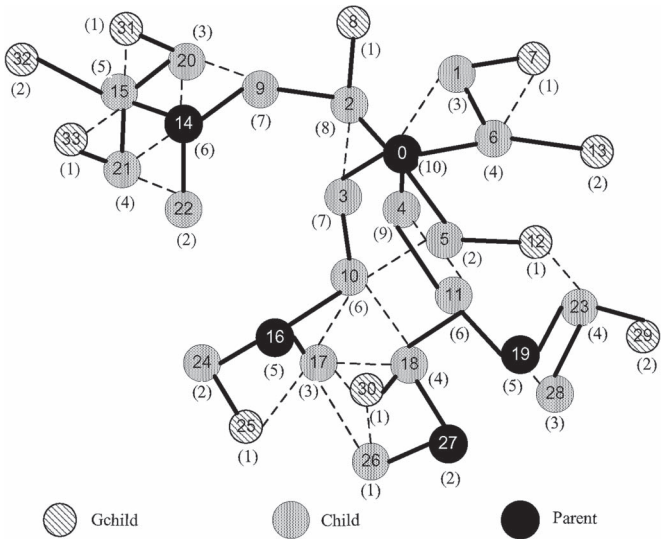


Fig. 8. Complete tree and schedules in the example graph.

V. Performance Analysis

In this section, we theoretically prove that the latency upper bound of FAST is $12R + \Delta - 2$. Moreover, a more general protocol interference model, i.e., when the interference range r_I is higher than transmission range r (i.e., $r_I > r$), is also considered. We then present an example to show the problem in [15] for aggregation latency analysis. At last, the message complexity of our proposed algorithm is analyzed.

A. Delay Bound Analysis

To collect all the data from the network nodes to topology center v_c, as mentioned before, aggregation scheduling is performed in three independent phases: (1) Gchilds to Childs scheduling, (2) not-in-the-backbone $Chlid$s to Parents scheduling, and (3) backbone scheduling. Each phase requires a set of slots to terminate. First, we prove that aggregating data to the backbone costs at most $\Delta + 4$ time slots. Next, an upper bound at $\frac{2\pi R}{\frac{\sqrt{3}}{2}} + 3R - 6$ is computed to complete the data aggregation process through the backbone to v_c. Here, we look at the phases separately.

(1) Gchilds to Childs scheduling.

Lemma 5: Let each Child$c$ have three types of neighbor Gchilds grouped into three sets $G_c(1)$, $G_c(2)$, and $G_c(3)$ (where $G_c = \bigcup_{i=1}^{3} G_c(i)$ is the set of c's neighbor Gchilds) as shown in Fig. 6. It takes time at most $|G_c(1)| + |G_c(2)|$ to aggregate data from both $G_c(1)$ and $G_c(2)$. In other words, if u is the last common Gchild scheduled in the vicinity of c, u's schedule time is not more than the number of c's common Gchilds (i.e., $\leq |G_c(1)| + |G_c(2)|$).

Lemma 6: For each Child$c$, all its Gchild nodes are scheduled at most within $|G_c|$ time slots.

Detailed proofs of Lemmas 5 and 6 can be found in Section A and Section B of Appendix, respectively.

(2) not-in-the-backbone $Chlid$s to Parents scheduling.

Lemma 7: FAST generates collision-free schedules with an upper bound on delay of $\Delta + 4$ time slots to completely collect data from all the not-in-the-backbone nodes.

Proof: We claim that it takes at most 5 time slots for each Parent$u$ to finish data collection from their associated not-in-the-backbone Childs. The reason lies behind this fact is that there are at most 5 independent points (selected Childs) within a disk of unit radius around u. In this way, according to the Childs connection policy (see Section IV-C as well as Fig. 7), u's not-in-the-backbone neighbors connect to the selected Childs as their parents and complete the aggregation tree. As mentioned before, a selected Child$c$ may have two types of u's Childs in the vicinity: (1) active nodes A_c having no schedule, and (2) inactive nodes I_c previously being scheduled by some other selected Childs. Finally, both nodes I_c and A_c have schedules in the interval $[(T_{final}(c) + 1), (\Delta - 1)]$, where $T_{final}(c) + |I_c| + |A_c| \leq \Delta - 1$. Here, $T_{final}(c) \leq |G_c|$ is the time in which Child$c$ finishes receiving data from all its Gchilds. It is also worth mentioning that one of Child$c$ neighbors is its parent. Thus, it takes at most $\Delta - 1$ time slots for each selected Child$c$ to schedule all its neighbors. Parent$u$ then assigns at most 5 time slots to the selected points. Hence, in total, data aggregation to the back backbone has a delay bound of $\Delta + 4$ time slots. □

(3) backbone scheduling.

Here, to analyze the latency bound of data aggregation through the backbone, we present a novel method using a critical path approach. The critical path is constructed hop-by-hop by adding the node with the last assigned schedule in each step from the sink to the leaves in the backbone. We first estimate the number of nodes on the critical path. Next, we count the number of disjoint paths which enter the critical path nodes through its 3-hop boundary. Finally, considering that the critical path imposes the maximum number of time slots on scheduling, we prove a delay bound of $\frac{2\pi R}{\frac{\sqrt{3}}{2}} + 3R - 6$ slots in the backbone.

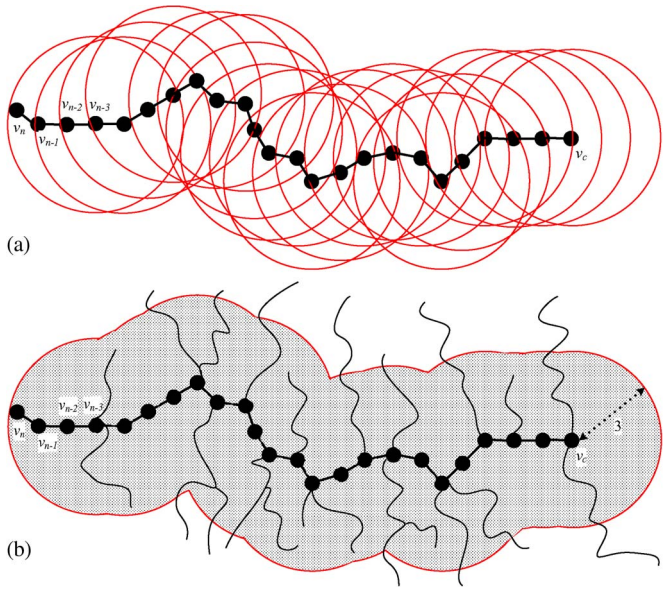


Fig. 9. (a) Critical path (b) 3-hop area boundary.

Consider a critical path $CP = v_n, v_{n-1}, \ldots, v_1, v_c$ from Parent v_n as a child node in the backbone to center v_c. As mentioned before, to build a parent-child relationship in the backbone, each Parent finds another one whose level is smaller. Hence, except for v_c, each Parent at level k may route to another at level $k - 1/k - 2/k - 3$ in 3 hops. In other words, Parents place at the consecutive levels on critical path in the worst case. Furthermore, v_n is at most at level R of the aggregation tree, where R is the network radius. Considering there is no entry to v_n in the backbone, we analyze the delay bound from the next Parent, i.e., v_{n-3}, at most at level $R - 1$ to center v_c. It is also worth noting that the closest Parent node to the center is at level 3. Finally, there exists a total number of $((R-1)-3) \times 3 + 4 = 3R - 8$ nodes from v_{n-3} to v_c in the critical path (see Fig. 9(a)). Moreover, to find the number of entries into the critical path through its 3-hop boundary, the circumference of the area made by the union of circles of radius 3 centered at nodes $v_{n-3}, v_{n-4}, \ldots, v_1, v_c$ is necessary to be computed (see Fig. 9(b)).

Lemma 8: Given a path $P = v_{n-3}, v_{n-4}, \ldots, v_1, v_0$ in the backbone, where v_{n-3} and v_0 have level $R - 1$ and 0, respectively, circumference C of the area made by the union of the circles of radius 3 centered at the path is at most $2\pi R$.

Proof: To prove Lemma 8, we compute area A made by the union of the circles of radius 3 centered at the path in two ways:

(1) From each disc of radius 3 centered at node $i(D(i,3)|i : 1 \leq i \leq n-3)$, the overlapped area of disk $D(i-1,3)$ is removed. What remains is an area A_i, where $A = \Sigma_{i=0}^{n-3} A_i$. As can be seen in Fig. 10, the area of sectors $S(arc\ ab, bv_i, v_ia)$ and $S(arc\ ab, bv_{i-1}, v_{i-1}a)$ are $\frac{180+\alpha}{360} \times \pi 3^2$ and $\frac{180-\beta}{360} \times \pi 3^2$, respectively, where $\alpha = \beta$ due to the parallelism of lines av_i and $v_{i-1}b$ in rhombus av_ibv_{i-1}. Therefore,

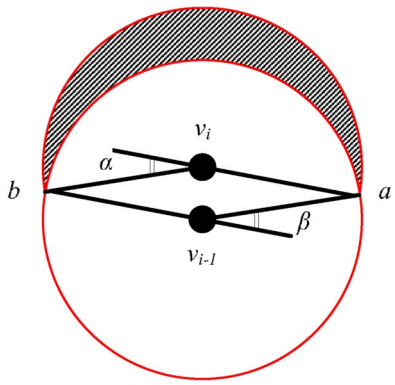


Fig. 10. Non-overlapped zone of 3-hop areas of two consecutive nodes in the critical path.

non-overlapped area A_i (hatched in Fig. 10) is computed as $A_i = \frac{180+\alpha}{360} \times \pi 3^2 - \frac{180-\alpha}{360} \times \pi 3^2 + x = \frac{2\alpha}{360} \times 9\pi + x$, where x is the rhombus area and $\widehat{\alpha} \leq \arccos\left(\frac{17}{18}\right) \leq 20$. It is worth mentioning that the closer v_i and v_{i-1} are, the smaller A_i is. Finally, in the worst case, we have $A_i = \pi + x$, so $A \leq (n-1)\pi + \Sigma x + 9\pi$, where 9π is the area covered by center v_0.

(2) Here, we again compute area A, but this time as a function of its circumference. For each disc of radius 3 centered at node $i : 0 \leq i \leq n-3$, we divide it into congruent sectors; therefore, the disc area can be estimated by $\frac{3}{2}\Sigma\varepsilon_i$, where a sector of a circle is a region enclosed by two radii and an arc of length ε. Hence, we have $A = \frac{3}{2}\Sigma\varepsilon_i + \Sigma x = \frac{3}{2}C + \Sigma x$, where x is the rhombus area.

Finally, the circumference of the area made by the union of the circles is computed by solving equation $(n-1)\pi + \Sigma x + 9\pi = \frac{3}{2}C + \Sigma x$. By replacing $n = 3R - 8$ in the equation, we have $C \leq 2\pi R$. □

Lemma 9: Given a communication graph G of the network, each two disjoint multi-hop paths which enter the backbone boundary have a distance not less than $\frac{\sqrt{3}}{2}$.

Proof: At first, we illustrate that distance d between each vertex v in UDG G and each chord $c(a,b)$ on the unit circular disk boundary centered at v cannot be less than $\frac{\sqrt{3}}{2}$. We fix vertices a and v as in Fig. 11(a) and move b on the boundary. The farther b goes, the smaller d is. Considering that $|a-b| \leq 1$, in the best case, we have an interior equilateral triangle vab with height of $\frac{\sqrt{3}}{2}$.

Next, we prove Lemma 9 by contradiction. Let $L(a,b)$ and $L(c,d)$ be two links on two disjoint paths in the backbone as in Fig. 11(b). In the closest case, node c can reach the intersect point of two black arcs centered at a and b. Now, suppose that there is a distance less that $\frac{\sqrt{3}}{2}$ between two points P_1, P_2 on the paths. We know angle $\widehat{P_1bP_2} \geq 60$; therefore, according to the triangle theorem (the longer edge lies opposite the larger angle), the requirements are not satisfied. As a matter of fact, although $\widehat{P_1bP_2} \geq 60$, we have $|b - P_2| \geq \frac{\sqrt{3}}{2}$ and $|P_1 - P_2| \leq \frac{\sqrt{3}}{2}$ that contradicts the assumption. Definitely, by changing the relative position of c and d to $L(a,b)$, it is easy to see that the minimum distance is fixed. Moreover, this is true if we reduce the intersection of circles by changing the Euclidian distance of centers a and b. □

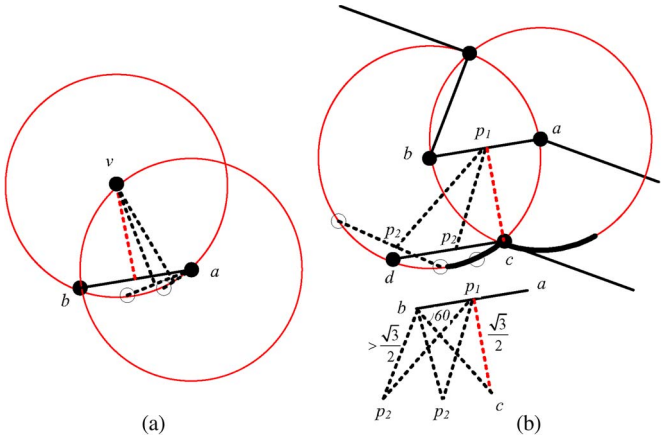


Fig. 11. Distance calculation between (a) a vertex and its unit circular disk boundary (b) two disjoint paths.

Lemma 10: Let CP be the critical path in constructed tree T, then the aggregation latency in CP is upper-bounded by $\frac{2\pi R}{\frac{\sqrt{3}}{2}} + 3R - 6$.

Proof: The number of disjoint multi-hop paths which enter different points of the critical path (see Fig. 9(b)) is computed by dividing the circumference of the area made by the union of the circles of radius 3 centered at the critical path (i.e., $2\pi R$) by the minimum distance of entries on the boundary (i.e., $\frac{\sqrt{3}}{2}$). Definitely, the paths originated in area A within 3 hops have no latency imposition on CP, where they pick up the first vacant slots for scheduling; therefore, they are not in competition with CP. In the worst case for each node in the critical path, all the entries are ready at the same time and requests scheduling. Hence, data aggregation in the critical path has a delay bound of $\frac{2\pi R}{\frac{\sqrt{3}}{2}} + 3R - 6$, where $3R-6$ is the total number of links on CP from v_n to v_c. □

The following theorem estimates the total time latency.

Theorem 11: The time latency of FAST is at most $12R + \Delta - 2$.

Proof: Here, we estimate the total aggregation latency through the network to the sink node. The not-in-the-backbone nodes need $\Delta + 4$ time slots by Lemma 7. Furthermore, considering that the last transmission is scheduled in the critical path, it takes at most $\frac{2\pi R}{\frac{\sqrt{3}}{2}} + 3R - 6$ to aggregate data through the backbone to topology center v_c. Indeed, the data from the other paths are received in the topology center or critical path before we completely schedule the critical path. Finally, v_c uses at most R time slots to send data to the sink via the shortest path. Thus, the total time latency to receive the final aggregation result is $11.25R + \Delta - 2 \leq 12R + \Delta - 2$ time slots. □

B. A More General Protocol Interference Model

Here, to make our approach more realistic and practical, we provide the upper bound of delay under the protocol interference model when the interference range r_I is higher than transmission range r (i.e., $r_I > r$). The interference range is the maximum distance within which nodes in receiving mode will be disturbed by an unrelated transmitter, thus suffering a loss. Moreover, the interference graph of the network is defined as a graph $\acute{G}(V, \acute{E})$, where V is the set of all the nodes and $\acute{E}$ is the edge set of $\acute{G}$. An edge $(u,v) \in \acute{E}$ if and only if $|u - v| \leq r_I$.

Theorem 12: Under the protocol interference model when $r \leq r_I < 3r$, the time latency of FAST is at most $12R + \acute{\Delta} - 2$, where R is the network radius in communication graph G and $\acute{\Delta}$ is the node degree in interference graph $\acute{G}$.

Proof: After the incomplete tree construction on G, as mentioned before, aggregation scheduling is performed in 3 phases. However, considering a more limited number of transmissions forced by a higher number of interferences, it takes:

- at most $\acute{\Delta} - 1$ time slots to aggregate data from `Gchilds` to `Childs`. It is because a `Child` c needs to wait for the first type of neighbor `Gchilds` in its interference range (i.e., $\acute{G}_c(1)$) according to $\acute{G}$ before starting to schedule its own children `Gchilds` (i.e., $G_c(2)$ and $G_c(3)$).
- at most 5 time slots to aggregate data from not-in-the-backbone `Childs` to the backbone.
- at most $\frac{2\pi R}{\frac{\sqrt{3}}{2}} + 3R - 6$ time slots to aggregate data from the backbone. To handle the interference in the backbone scheduling, all the waiting policies need to be applied on interference graph $\acute{G}$ to make interference-free schedules. Unlike the case of $r_I = r$, here it is possible that the `Parents` cannot send their data simultaneously. Thus, like a `Child` node, a `Parent` needs to participate in waiting policy both for its children in G and higher ranked nodes in $\acute{G}$. However, it is worth mentioning that the latency bound does not change here due to the independency of our analysis approach (which uses hop-by-hop transmissions on the critical path) from the interference range. The only point is that for very high interference ranges a node outside the area boundary may affect the transmissions on the critical path. Considering that a margin of 3 is provided around the critical path in our analysis, our delay bound is just valid for $r \leq r_I < 3r$.

Therefore, according to Lemma 7, FAST generates collision-free schedules with an upper bound on delay of $\acute{\Delta} + 4$ time slots to completely collect data from all the not-in-the-backbone nodes. Moreover, according to Lemma 10, aggregation in backbone has a delay bound of $\frac{2\pi R}{\frac{\sqrt{3}}{2}} + 3R - 6$. Thus, it totally takes at most $12R + \acute{\Delta} - 2$ time slots for the sink to receive the aggregated data under the protocol interference model when $r \leq r_I < 3r$ (which is a wide enough range in practice). □

C. Discussion of Clu-DDAS Algorithm

It has been proved that Clu-DDAS [15] has a latency upper bound at $4R + 2\Delta - 2$ based on Lemma 13.

Lemma 13: Let u be a $BLUE$ node in a Rest-Clu-DAT. Then u has at most 4 $BLUE$ neighbors [15].

Here, we present a counter-example to show that Lemma 13 is invalid, so the provided aggregation latency upper bound is incorrect. According to Clu-DDAS algorithm, the tree in Fig. 12 is a typical form of a Rest-Clu-DAT, where there are at least 3 hops between any two $BLACK$ nodes. As is evident from our true example, $BULE$ node C have more than 4 $BLUE$ neighbors which contradicts with Lemma 13.

It is worth mentioning that although the latency analysis is not valid in [15], we compare our scheduling algorithm to Clu-DDAS and show how FAST outperforms it.

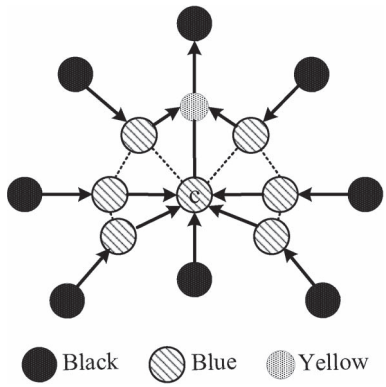


Fig. 12. An example Rest-Clu-DAT.

D. Message Complexity Analysis

Here, we analyze the communication cost in terms of the order of message transmissions through the network. During both 3-hop dominating sets selection and tree construction, each node sends messages just to its neighbors. Thus, the message complexity of either of these two functions is $O(n\Delta)$, where n is the number of nodes in the network. Next, the distributed aggregation scheduling algorithm needs $O(n\Delta)$ messages. The reason is that the number of messages transmitted by each node can be bounded by the number of a node's neighbors. Altogether, FAST requires $O(n\Delta)$ message transmissions as [15], [22].

VI. Simulation Analysis

In this section, we evaluate the performance of our algorithm via simulation in Matlab. The results are compared with IAS [22] and Clu-DDAS [15] as two previously known best minimum latency distributed scheduling algorithms.

In our simulation, we randomly and uniformly deploy a number of sensor nodes into a square region of 200 m $\times$ 200 m. All sensor nodes have the same transmission radius. Considering that the aggregation latency is strongly affected by the maximum node degree (Δ) and the network radius (R), we compare the number of time slots needed to aggregate data from the leaves to the sink in two different scenarios.

In the first scenario, we randomly generate the network topology (connected) with different number of network nodes (increasing from 300 to 700 with step 50) while ensuring the network density remains unchanged. To achieve this goal, the network deployment area increases with the increment of the number of nodes. By doing this, we actually fix Δ and investigate how increasing R affects aggregation latency. Fig. 13 shows the latencies for various transmission ranges (20 m, 25 m, and 30 m), so different network densities ($\Delta = 18$, $\Delta = 25$, $\Delta = 32$). As is evident from the figure, the aggregation latency is proportional to R (for each scenario, R is indicated by the value in the brackets right after the number of network nodes on x-coordinate). From Fig. 13(a), we can see that when the network density is not high, the latency difference among three methods is not significant. However, as the network density increases, FAST outperforms IAS and Clu-DDAS (on average by 22% and 16%, respectively, in Fig. 13(c)). The reason is a high potential of parallel transmissions in our proposed algorithm, especially when it is applied to a dense

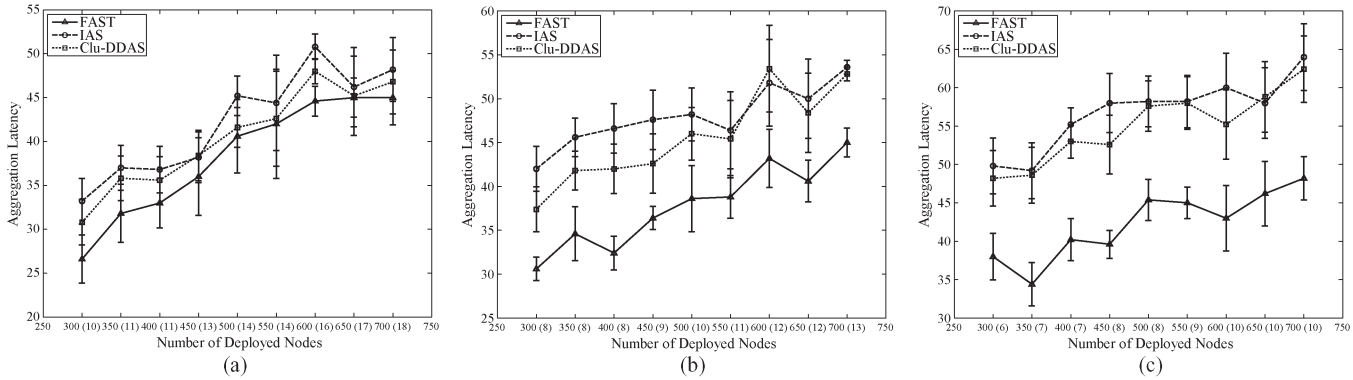


Fig. 13. Aggregation latency for fixed Δ in each scenario. (a) $\Delta = 18$; (b) $\Delta = 25$; (c) $\Delta = 32$.

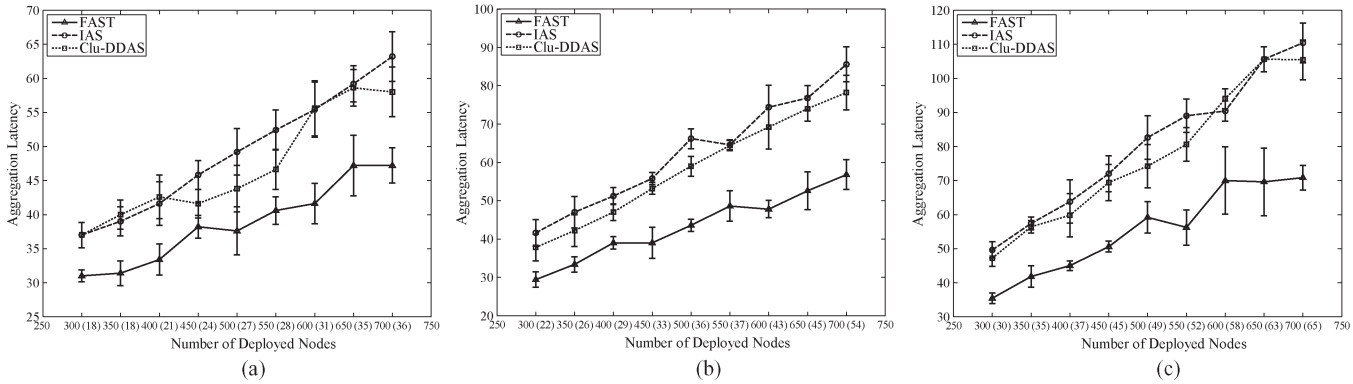


Fig. 14. Aggregation latency for fixed R in each scenario. (a) $R = 9$; (b) $R = 7$; (c) $R = 6$.

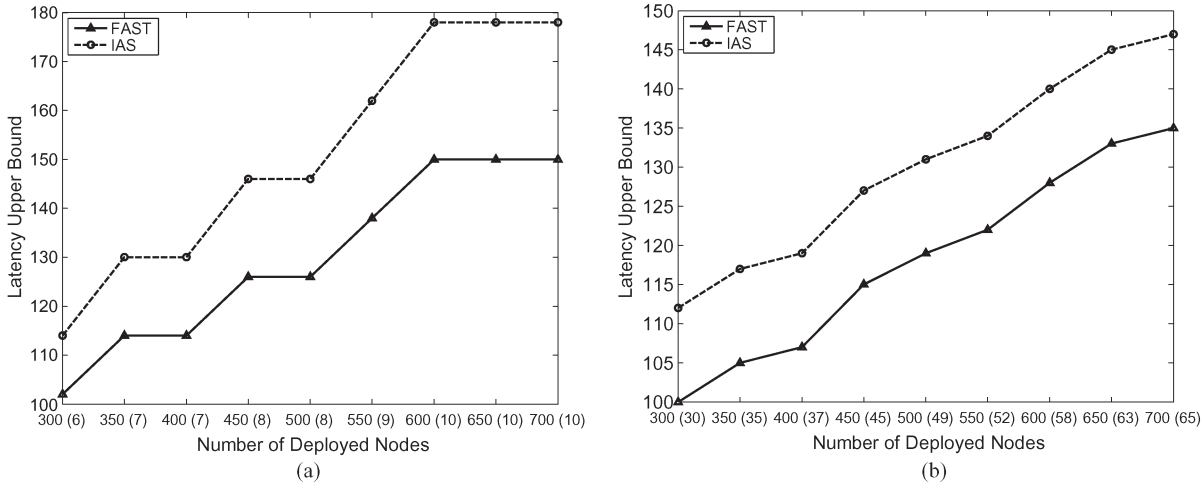


Fig. 15. Comparison of latency upper bounds. (a) $\Delta = 32$; (b) $R = 6$.

network, due to the construction of a special aggregation tree simultaneous with our scheduling method.

In the second scenario, we fix the network area and continue to increase the number of network nodes (from 300 to 700 with step 50) while keeping the network connected. By doing this, we actually fix R for each scenario and investigate how increasing Δ affects aggregation latency. As we can see in Fig. 14, aggregation latency is proportional to Δ. As the number of nodes or the transmission radius increases, the average size of the competitor nodes also increases. Thus, each node has to compete with more nodes, which costs more time slots. However, FAST has much lower latency than IAS and Clu-DDAS (on average, 29% and 21%, respectively, in Fig. 14(c)), especially in large networks. It is worth mentioning that we have carried out the experiments 10 times for each setting, and the average performance along with 95% confidence interval was reported.

Besides, considering that in our paper and all related studies the performance analysis part mainly focuses on the latency upper bound, we also need to evaluate the worst-case performance of the algorithms. Here, since [15] cannot give a correct latency bound to the scheduling problem, we only compare FAST with IAS. Fig. 15 compares the latency upper bounds of the schedules on the constructed aggregation trees when the transmission radius is fixed to 30 m (see Figs. 13(c) and 14(c)).

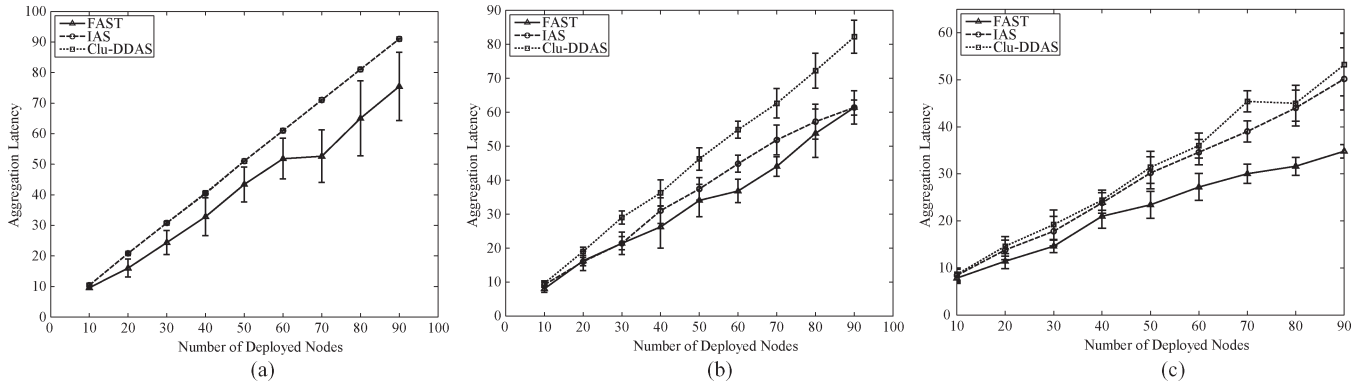


Fig. 16. Aggregation latency for $R \leq 3$. (a) $R = 1$; (b) $R = 2$; (c) $R = 3$.

To the best of our knowledge, FAST has so far had the minimum latency bound for data aggregation in tree-based WSNs. It is worth mentioning that the time latencies in the experiments are much more better than the theoretical latency bounds. The reason is that we magnify the latency too much when computing the upper bound, which makes it to be rather loose.

Moreover, one may be concerned about how FAST would perform when radius of the graph is very small, say when $R \leq 3$, and $12R + \Delta - 2$ has no advantage over $16R + \Delta - 14$ [22]. To address this concern, we randomly and uniformly deploy a number (10–90) of sensor nodes into a square region of 50 m $\times$ 50 m. Our simulation results in Fig. 16 shows that, even in the case of $R \leq 3$, our proposed algorithm outperforms the others. This is mainly because of a better distribution of connections among their neighbors for an efficient tree construction which strongly helps the reduction of latency by providing a higher potential of parallel transmissions.

VII. Conclusion

Minimizing latency is of primary concern for efficient data aggregation in WSNs. In this paper, we designed FAST as a novel collision-free minimum latency data aggregation scheduling algorithm for tree-based structures. To achieve an optimal time slot assignment, FAST simultaneously execute the aggregation scheduling and balanced C3DS-based tree construction. Next, we gave a new upper bound on aggregation latency at $12R + \Delta - 2$ time slots. According to our knowledge, this is the best result of latency upper bound for data aggregation and much better than those of previous approaches. The simulation studies and theoretical analysis indicated that FAST significantly outperforms other existing aggregation scheduling algorithms.

The algorithm could be modified to take into account some aspects that have not been addressed in this work. For instance, studying the MLAS problem in other interference models as well as in multi-sink WSNs can be considered in future studies.

Appendix

A. Proof of Lemma 5

Proof: We prove this by induction on the total number of c's common `Gchilds`, $|CG_c| = |G_c(1)| + |G_c(2)|$, where $|G_c(1)|$ and $|G_c(2)|$ denote the cardinality of c's common neighbors.

Base case: Consider $|CG_c| = 1$, so `Child` c has only one common `Gchild` (say g). According to the `Gchilds` connection policy, if g joins c, it is assigned Slot 1 as its schedule time; otherwise, there is another `Child` (say x) as g's parent (where $|CG_x| = |CG_c|$ and $ID_x < ID_c$). Thus, $|CG_x| = 1$ and, again, g is scheduled to transmit its data in Slot 1.

Inductive hypothesis: For $|CG_c| < k$, Lemma 5 holds.

Inductive step: Assume the inductive hypothesis is true for $|CG_c| < k$. We need to show Lemma 5 is true for $|CG_c| = k$. The proof is by induction on c's ID.

Base case: Consider c has the minimum ID. All the c's common `Gchild` neighbors in $G_c(1)$ have been connected to other `Childs` denoted by $x_i : i \leq |G_c(1)|$, where $\forall i |CG_{x_i}| < |CG_c|$ (i.e., x_i has definitely fewer number of common `Gchilds` than c). Therefore, they are all scheduled within $|CG_c| < k$ slots, although the assigned slots may not be consecutive. Furthermore, as a key feature of our method, some nodes in $G_c(1)$ connected to different `Childs` get the potential of simultaneous transmission in just one slot. Hence, all nodes in $G_c(1)$ requires at most $|G_c(1)|$ number of total slots. Finally, by scheduling the remaining nodes in interval $[1, |CG_c|]$, all common `Gchilds` can send their data in time slots not more than $|CG_c|$.

Inductive hypothesis: For each `Child` c with ID_c less that N, Lemma 5 holds.

Inductive step: Assume the inductive hypothesis is true for each `Child` u with ID_u less than N. We need to show Lemma 5 is true for `Child` c with ID_c equal to N. The proof is the same as for the base case. Some `Child` $x_i : i \leq |G_c(1)|$ are actually responsible for schedule assignment while (1) $|CG_{x_i}| < |CG_c| = k$ or (2) $|CG_{x_i}| = |CG_c|$ and $\forall i ID_{x_i} < N$. Hence, according to the inductive hypothesis, all the x's common neighbors in $G_c(1)$ are assigned slots in interval $[1, k]$. Finally, it costs at most $|CG_c| = k$ time slots to schedule all common `Gchilds`. □

B. Proof of Lemma 6

Proof: Without loss of generality, suppose that u is the last `Gchild` scheduled in the constructed structure, where c is its parent. If $u \in CG_c$, it takes at most $|CG_c| \leq |G_c|$ time slots when u finishes its data transmission to c; otherwise, it is assigned a time in interval $[(k+1), |G_c|]$, where the first $k \leq |G_c|$ slots are assigned to c's common `Gchilds`. □

REFERENCES

[1] B. Alinia, H. Yousefi, M. S. Talebi, and A. Khonsari, "Maximizing quality of aggregation in delay-constrained wireless sensor networks," *IEEE Commun. Lett.*, vol. 17, no. 11, pp. 2084–2087, Nov. 2013.

[2] S. Bahrami, H. Yousefi, and A. Movaghar, "DACA: Data-aware clustering and aggregation in query-driven wireless sensor networks," in *Proc. 21st IEEE ICCCN*, 2012, pp. 1–7.

[3] J. Beaver, M. A. Sharaf, A. Labrinidis, R. Labrinidis, and P. K. Chrysanthis, "Location-aware routing for data aggregation in sensor networks," in *Proc. Geosensor Netw. Workshop*, 2003, pp. 189–210.

[4] D. M. Blough, G. Resta, and P. Santi, "Approximation algorithms for wireless link scheduling with sinr-based interference," *IEEE/ACM Trans. Netw.*, vol. 18, no. 6, pp. 1701–1712, Dec. 2010.

[5] X. Chen, X. Hu, and J. Zhu, "Minimum data aggregation time problem in wireless sensor networks," in *Proc. 1st Int. Conf. MSN*, 2005, pp. 133–142.

[6] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: A survey," *IEEE Wireless Commun.*, vol. 14, no. 2, pp. 70–87, Apr. 2007.

[7] F. Ghods, H. Yousefi, A. M. A. Hemmatyar, and A. Movaghar, "MC-MLAS: Multi-channel minimum latency aggregation scheduling in wireless sensor networks," *Comput. Netw.*, vol. 57, no. 18, pp. 3812–3825, Dec. 2013.

[8] A. Ghosh, O. Incel, V. Kumar, and B. Krishnamachari, "Multichannel scheduling and spanning trees: Throughput-delay trade-off for fast data collection in sensor networks," *IEEE/ACM Trans. Netw.*, vol. 19, no. 6, pp. 1731–1744, Dec. 2011.

[9] P. Gupta and P. R. Kumar, "The capacity of wireless networks," *IEEE Trans. Inf. Theory*, vol. 46, no. 2, pp. 388–404, Mar. 2000.

[10] S. Hariharan and N. B. Shroff, "Maximizing aggregated information in sensor networks under deadline constraints," *IEEE Trans. Autom. Control*, vol. 56, no. 10, pp. 2369–2380, Oct. 2011.

[11] 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 *26th IEEE INFOCOM*, May 2007, pp. 366–372.

[12] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann, "Impact of network density on data aggregation in wireless sensor networks," in *Proc. 22nd ICDCS*, 2002, pp. 457–458.

[13] H. Li, Q. S. Hua, C. Wu, and F. C. M. Lau, "Minimum-latency aggregation scheduling in wireless sensor networks under physical interference model," in *Proc. 13th ACM Int. Conf. MSWIM*, 2010, pp. 360–367.

[14] X.-Y. Li *et al.*, "Efficient data aggregation in multi-hop wireless sensor networks under physical interference model," in *Proc. 6th Int. Conf. MASS*, 2009, pp. 353–362.

[15] Y. Li, L. Guo, and S. Prasad, "An energy-efficient distributed algorithm for minimum-latency aggregation scheduling in wireless sensor networks," in *Proc. 30th IEEE ICDCS*, Jun. 2010, pp. 827–836.

[16] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TAG: A tiny aggregation service for Ad-Hoc sensor networks," in *Proc. USENIX Symp. OSDI*, 2002, pp. 131–146.

[17] T. N. Nguyen, N. X. Lam, D. T. Huynh, and M. K. An, "Scheduling problems in interference-aware wireless sensor networks," in *Proc. ICNC*, 2013, pp. 783–789.

[18] R. Rajagopalan and P. K. Varshney, "Data aggregation techniques in sensor networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 8, no. 4, pp. 48–63, 2006.

[19] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis, "TiNA: A scheme for temporal coherency-aware in-network aggregation," in *Proc. 3rd ACM Int. Workshop MobiDe*, 2003, pp. 69–76.

[20] P. J. Wan, K. Alzoubi, and O. Frieder, "Distributed construction of connected dominating set in wireless Ad Hoc networks," in *Proc. 21st IEEE INFOCOM*, 2002, vol. 3, pp. 1597–1604.

[21] P. J. Wan, S. C. H. Huang, L. Wang, Z. Wan, and X. Jia, "Minimum-latency aggregation scheduling in multihop wireless networks," in *Proc. 10th ACM Int. Symp. MobiHoc*, 2009, pp. 185–194.

[22] X. Xu, X. Y. Li, X. Mao, S. Tang, and S. Wang, "A delay-efficient algorithm for data aggregation in multihop wireless sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 1, pp. 163–175, Jan. 2011.

[23] X. Xu, X. Y. Li, P. J. Wan, and S. Tang, "Efficient scheduling for periodic aggregation queries in multihop sensor networks," *IEEE/ACM Trans. Netw.*, vol. 20, no. 3, pp. 690–698, Jun. 2012.

[24] H. Yousefi, M. H. Yeganeh, N. Alinaghipour, and A. Movaghar, "Structure-free real-time data aggregation in wireless sensor networks," *Comput. Commun.*, vol. 35, no. 9, pp. 1132–1140, May 2012.

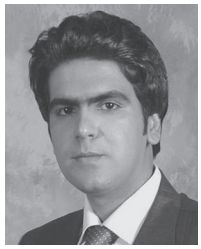
[25] B. Yu, J. Li, and Y. Li, "Distributed data aggregation scheduling in wireless sensor networks," in *Proc. 28th IEEE INFOCOM*, Apr. 2009, pp. 2159–2167.

[26] M. Yu, K. K. Leung, and A. Malvankar, "A dynamic clustering and energy efficient routing technique for sensor networks," *IEEE Trans. Wireless Commun.*, vol. 6, no. 8, pp. 3069–3079, Aug. 2007.

[27] Y. Yu, B. Krishnamachari, and V. K. Prasanna, "Energy-latency tradeoffs for data gathering in wireless sensor networks," in *Proc. 23th IEEE INFOCOM*, Mar. 2004, vol. 1, pp. 244–255.

[28] J. Zhang, X. Jia, and G. Xing, "Real-time data aggregation in contention-based wireless sensor networks," *ACM Trans. Sensor Netw.*, vol. 7, no. 1, pp. 1–25, Aug. 2010.

[29] J. Zhu, S. Papavassiliou, and J. Yang, "Adaptive localized qos-constrained data aggregation and processing in distributed sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 9, pp. 923–933, Sep. 2006.



Hamed Yousefi received the M.Sc. degree from the Sharif University of Technology, Tehran, Iran, in 2009. He is currently pursuing the Ph.D. degree in computer engineering at the same university. From 2013 to 2014, he was a visiting Ph.D. student at the Department of Computer Science, University of Michigan, Ann Arbor, USA. He is a member of Iranian Inventors Association. His research interests include wireless networks, performance and dependability modeling, and real-time communications.

Marzieh Malekimajd received the B.Sc. and M.Sc. degrees in computer engineering from Sharif University of Technology in 2009 and 2011, respectively. She is currently pursuing the Ph.D. degree in computer engineering at Sharif University of Technology. She was a visiting Ph.D. student with the Department of Electronics, Information, and Bioengineering, Politecnico di Milano, Milan, Italy, in 2014. Her research interests include graph theory and algorithmic aspects of cloud computing, MapReduce, and networks.

Majid Ashouri received the M.Sc. degree in computer engineering from Sharif University of Technology in 2012. His research interests are in the analysis and design of algorithms/protocols for wireless networks, particularly for *ad hoc* and sensor networks and in the performance evaluation of computer networks.

Ali Movaghar (SM'07) received the B.Sc. degree in electrical engineering from the University of Tehran in 1977, and the M.Sc. and Ph.D. degrees in computer, information, and control engineering from University of Michigan, Ann Arbor, MI, USA, in 1979 and 1985, respectively. He is a Professor in the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran where he joined first as an Assistant Professor in 1993. He visited the INRIA in France in 1984 and the Department of Electrical Engineering and Computer Science, University of California, Irvine, CA, USA, in 1984 and 2011, respectively, worked at AT&T Information Systems in Naperville, IL, USA, in 1985–1986, and taught at the University of Michigan, Ann Arbor, in 1987–1989. His main areas of interest are performance and dependability modeling, verification and validation, wireless networks, and distributed real-time systems. He is a senior member of the ACM.