Aggregation Convergecast Scheduling in Wireless Sensor Networks

Baljeet Malhotra
Computing Science Department
University of Alberta, Canada
Email: baljeet@cs.ualberta.ca

Ioanis Nikolaidis
Computing Science Department
University of Alberta, Canada
Email: yannis@cs.ualberta.ca

Mario A. Nascimento
Computing Science Department
University of Alberta, Canada
Email: mn@cs.ualberta.ca

***Abstract*—We consider the problem of aggregation convergecast scheduling in wireless sensor networks. Aggregation convergecast differs from regular convergecast in that it accommodates transmission dependencies that allow in-network aggregation to be performed. We formulate the abstract problem of aggregation convergecast and review the existing literature. We observe that existing schemes adopt essentially a two phase approach, consisting of, first, a tree construction and, second, a scheduling phase. Following a similar approach, we propose two new improvements, one to each of the two phases. Starting with a new lower bound on the schedule length, we make use of it in the tree construction step. The tree construction step consists of solutions to instances of bipartite graph semi-matching. The scheduling step is a weight-based priority scheme that obeys dependency (tree) and interference constraints. The combination of both improvements is demonstrated to outperform all existing solutions in the literature. We also study the impact of each of the two improvements alone and pay attention to cases where the tree construction is obviated due to the existence of extrinsic dependency constraints.**

I. Introduction

Data gathering is a basic capability expected of any wireless sensor network. The usual means of performing data gathering is to have all nodes send their measurements (possibly over multiple hops) to a particular node, the *sink*. The corresponding many-to-one "funnel" type of communication is called *convergecast*. Convergecast usually operates by building a logical tree on top of the physical topology with the sink located at the root, and subsequently by routing packets along this tree. The problem thus becomes how to schedule transmissions to avoid interference and collisions. One approach is to rely on an underlying Medium Access Protocol to handle contention and retransmissions. Another is to construct a TDMA schedule (including possible combinations of TDMA with CDMA scheduling etc.). The advantage of a TDMA schedule for wireless sensors is that the transceivers can be turned on only when the schedule stipulates that they must either receive or transmit, thus scheduling saves energy that would have gone to *idle listening*. It is well known that idle listening, as required by many MAC protocols, consumes significant amount of energy [1], therefore, TDMA scheduling is seen as the energy efficient alternative.

There exist several algorithms for convergecasting in multihop radio networks [2] that can be used for wireless sensor networks. A common trait of most of those algorithms is the decomposition of the problem into two and independent subproblems: first a logical tree construction, followed by the scheduling of transmissions along the constructed tree. The common objective of scheduling algorithms is to use the least number of time slots. We will see a similar approach is followed in *aggregation convergecast*.

Aggregation convergecast is defined as the routing and the en-route aggregation of data as they travel to the sink (plus of course interference constraints as in regular convergecast). Aggregation is a means to achieve energy efficiency by reducing the transmitted traffic volume. TAG is one such aggregation approach that can be used for either data or message aggregation [3]. Another example of using aggregation is top-k query processing to determine the k highest values sensed across all the network nodes [4]. In its simplest definition, aggregation operates by ensuring that a node receives a specific number of incoming (fan-in) messages (from a correspondingly specific number of nodes from its neighbors), then combines the received data along with its own, and it generates a *single* output message that describes collectively the received and its own data together. Naturally, this definition can be applied recursively all the way to the sink. Aggregation convergecasting has been studied under various names such as Data Aggregation Monitoring and TAG, and it possesses subtle, but important, differences compared to regular convergecasting.

Specifically, a regular convergecast solution cannot be applied to the aggregation convergecast problem. On the surface, a regular convergecast requires more slots than an aggregation convergecast, since there is no aggregation and therefore the volume of traffic is not reduced en-route to the sink. A more important difference (assuming we did not mind as much about the extra slots) that makes aggregation convergecast intrinsically different, is that the order of transmissions in regular convergecast is not observing any dependency constraints. As one extreme example consider a regular convergecast solution in which the nodes close to the sink happen to transmit early in the schedule. This very same schedule solution (assuming it was applied to accommodate aggregation convergecast) would mean that the nodes close to the sink have no chance of aggregating their data prior to sending them to the sink, because they did not get the chance to receive data from nodes from further away. Thus, using a

solution from a regular convergecast scheduler to schedule aggregation convergecast seriously curtails the potential for aggregation. Clearly, aggregation convergecast deserves its own crisp problem formulation and scheduling algorithms.

The aggregation order, expressed as tree–structured dependency constraints, is part of the overall solution to the aggregation convergecast scheduling problem. In other words, it is up to the aggregation convergecast schedule construction to *also* determine the order of aggregation. While this approach is acceptable for simple aggregation operations, e.g., such as finding the maximum of all sensed values in a network, there could be reasons to force a *particular* aggregation order. Such *extrinsic* constraint on the dependency tree is application–dependent, but not uncommon. For example, a specific tree consisting of connected clusters may be required because of the cluster formation logic, e.g., by the need to rotate the role of clusterheads, and hence have a specific cluster structures over periods of time. Another reason to force the use of a particular tree is because it might be more effective in terms of the traffic volume reduction, i.e., more effective in terms of aggregation.

Fortunately, same with regular convergecast, most algorithms for aggregation convergecast consists of two phases, and the second one (scheduling) can be used even when a specific, externally provided, dependency constraint (in the form of a tree) is provided. The use of externally provided aggregation trees to the scheduling problem has been noted in the data management literature, [5], but the applicable literature is much wider because any paper following a two–phase approach is trivially applicable (using only its second phase) to the case where a specific tree is given as part of the input. For this reason, the performance study presented in this paper considers "mix-and-match" combinations of first and second phases to evaluate our new original contributions on the tree construction and scheduling phases.

In this paper, through a study of the structure and limitations of existing aggregation convergecast schemes, we propose a new competitive aggregation convergecast scheduling algorithm which outperforms the existing schemes. Specifically, in section II, we formally define the aggregation convergecast problem and provide some insights into its solution process. In section III we review the shortcomings of the existing literature and argue that there is potential for improvement, leveraging also observations already made by other authors. In section IV we address the tree construction phase (when no extrinsic constraint on the aggregation tree is supplied). By first proving a lower bound on the schedule length, we show how to use this bound to guide the construction of an aggregation tree. In section V we describe a simple, yet powerful, weight-based priority scheme to schedule transmissions subject to dependency (tree) and interference constraints. The combination of both improvements is studied in Section VI by presenting performance results. A special quality of Section VI is that, in our simulations, we study the behavior of the competing schemes using the *widest range* of node density values compared to what has been used in the literature so far.

II. System Model and Problem Statement

Throughout this paper, we assume a slotted system and that all nodes have adequate synchronization capabilities to follow a slot-by-slot schedule. The purpose of the schedule is to instruct each node when to: receive, transmit, or deactivate the transceiver (thus avoiding idle listening). The schedule is defined for a single round/epoch and we can assume that the same schedule is repeated for each round. The suggested mode of operation is consistent with the execution model of, the so called, *continuous queries*. Extensions to multiple (pre-computed) schedules are possible but outside the scope of the paper.

The important characteristic of *aggregation* convergecast is that a node transmits only once per round. This transmission can happen only after the node has collected data from other nodes on which to perform aggregation (including its own data). The aggregated data is subsequently received by another node that performs further aggregation along with those from other nodes, and so on, until the aggregated data arrives at the sink. We assume that a single slot period is sufficient to transmit data in its original (source) form or in its aggregated form. For example, in the case of top-k query processing some nodes may forward exactly k values and some nodes may forward less than k values depending on the number of values collected [4]. The slot size should be tailored to "fit" the worst case size of k values (inclusive of header and other overheads). For the purpose of this paper we will assume that all slots have the same fixed duration, which is sufficient to transmit/receive the *largest* packet/message required by the application.

We consider a network of N sensor nodes, with node 1 in the role of the sink. For each node i, the set of nodes denoted by $\mathcal{N}(i)$ are the "neighboring" nodes, i.e., within one hop of node i. To formally define the aggregation convergecasting problem, we assume that we are given T time slots and we introduce the binary decision variables $a^t_{(i,j)}$ to denote *link activation* events. Namely, $a^t_{(i,j)}$ denotes whether a transmission takes place from node i in timeslot t destined for node j. Hence, $a^t_{(i,j)}$ is the "activation" of the (directed) link (i,j) at time t. For notational convenience, for each node i (other than the sink) we define two additional variables $r(i)$ and $t(i)$ to stand for the transmission's recipient and the time slot which has been assigned for the transmission by node i, i.e., $a^{t(i)}_{(i,r(i))} = 1$ (with all the remaining $a^t_{(i,j)} = 0$). The decision problem is whether for a given T the following four constraints can be satisfied.

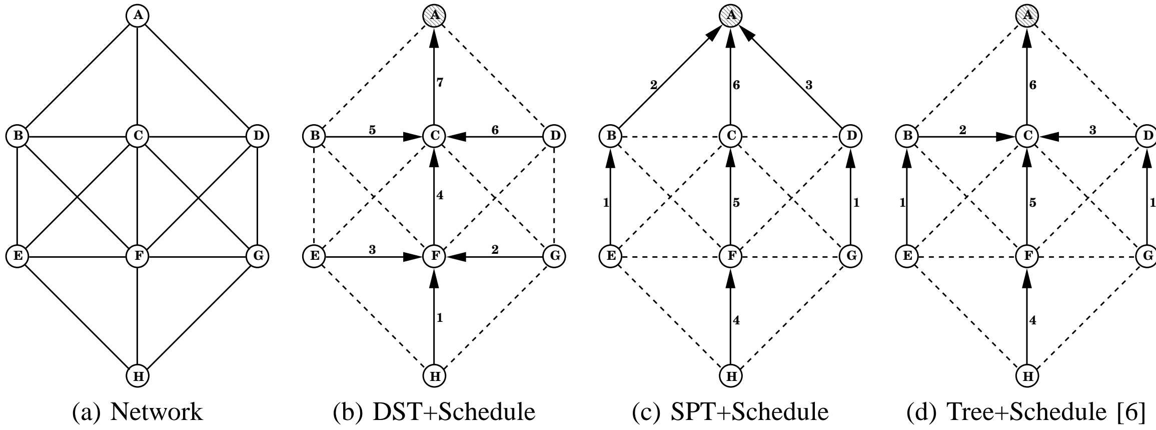
C1: $\sum_{k\in\mathcal{N}(i)} \sum_{t=1}^{T} a^t_{(i,k)} = 1, \quad i \in \{2,\ldots,N\}$

C2: $\sum_{k\in\mathcal{N}(i)} \sum_{t=t(i)}^{T} a^t_{(k,i)} = 0, \quad i \in \{2,\ldots,N\}$

C3: $\sum_{j\in\mathcal{N}(r(i))} a^{t(i)}_{(r(i),j)} = 0, \quad i \in \{1,\ldots,N\}$

C4: $\sum_{j\in\mathcal{N}(r(i)),j\neq i} \sum_{k\in\mathcal{N}(j)} a^{t(i)}_{(j,k)} = 0, \quad i \in \{1,\ldots,N\}$

Constraint C1 enforces a single transmission per node (except for the sink, for which $a^t_{(1,j)} = 0\ \forall j,t$). Constraint C2 ensures that, once a node transmits, it can no longer be the destination for any transmission (since transmission implies



(a) Network (b) DST+Schedule (c) SPT+Schedule (d) Tree+Schedule [6]

Fig. 1. Example network and three aggregation convergecast schedules. $a^{t(i)}_{(i,r(i))}$ is represented by an arrow $(i, r(i))$ labeled with the corresponding $t(i)$.

completion of the aggregation performed by that node in the round). Constraint C3 ensures that a recipient does not transmit in the same slot in which it has been assigned to receive, i.e., half–duplex operation. Constraint C4 captures the requirement that there be no interference at the recipient node.

Unfortunately, the problem defined above has been found to be NP-complete [6] even if restricted to Unit Disk Graphs (UDGs). Therefore, the existing solutions in the literature are heuristic approximations, and will be reviewed in the next section. An important observation is that the combination of C1 and C2 means that the edges (i, j) activated (that is, those where $a^t_{(i,j)} = 1$), regardless of when they are activated, form a subgraph of the network topology graph which is a tree. The property follows from the fact that this subgraph (a) contains exactly $N - 1$ edges (to satisfy C1) and (b) is directed and acyclic (to satisfy C2). As it is well-known, a subgraph satisfying those two characteristics must be a tree. In other words, a byproduct of solving the aggregation convergecast is a tree that represents a dependency of transmissions dictating the order in which aggregation is performed. This fact has been recognized by algorithms that structure their heuristics as consisting of two phases: first a phase to build a dependency tree (based on a variety of criteria), and then a second phase to construct a schedule. Let us capture the first phase (of tree construction) as an association between each node i and its parent node, $p(i)$, as its recipient, i.e., $r(i) = p(i)$. Then, the second phase (scheduling sans tree construction) consists of constraints C1-C4 plus a fifth constraint:

$$\text{C5:} \quad a^t_{(i,j)} = 0, \quad \begin{array}{l} i \in \{2, \ldots, N\}, \\ j \in \{1, \ldots, p(i) - 1, p(i) + 1, \ldots, N\} \end{array}$$

We observe that C5 restricts the set of feasible solutions that were previously achievable according to C1-C4 as there is no longer freedom in the choice of $r(i)$ *but* the timing, $t(i)$, of those transmissions is still open to optimization.

The tacit assumption of two–phase heuristics is that a "well–chosen" tree would be quite close (ideally, the exact same) as the one that would have resulted by C1-C4. Yet another benefit of decomposing the problem into two phases is that the first phase could be obviated if there are reasons to use a specific tree to begin with, i.e., if a particular tree is *required*, and hence provided as input to the aggregation convergecast problem.

III. Previous Work

To put the question of aggregation convergecast scheduling in some concrete perspective, we will consider the example of Figure 1. The one hop communication of the eight nodes is depicted by Figure 1(a). Let node A be the sink. Figures 1(b) to 1(d) depict alternative aggregation/dependency trees and corresponding schedules. Specifically, $a^{t(i)}_{(i,r(i))}$ are represented as directed edges from i to $r(i)$ and each edge is annotated with the timeslot $t(i)$ in which it will be activated in order for dependency and interference constraints to be met. We show in Figure 1(b) a dependency tree constructed as a Dominating Set Tree (DST) [4], akin to what would have been created to observe cluster-based neighborhood formations (here C and F are clusterheads). We also show, in Figure 1(c) the much more common and well known Shortest Path Tree (SPT) [3], [6] used as an aggregation tree. As a first observation, note that DST is not necessarily providing a short schedule, compared to SPT. DST requires seven slots instead of the six of the SPT example. This is partly to be expected because the DST's paths from nodes to sink are not necessarily optimal. On the other hand, given sufficiently rich connectivity, there also exist multiple alternative SPTs for the same physical topology. Hence, we would like to know what features make an SPT better than another SPT; a question we address in section IV.

If the scheduling algorithm is defined independently of the aggregation tree, then naturally the performance will vary greatly depending on the aggregation tree supplied. Such is the case of the scheme called PAS by Yu et al. [5]. Another (and more often) approach is for a scheme to prescribe both the tree construction as well as the scheduling algorithm. Within this category, there are those algorithms that retain the tree constructed in the first phase, and others that do not. In the latter category we find SDA by Chen et al. [6] and First–Fit by Huang et al. [7]. Unfortunately, the second of those papers produces schedules with possible conflicts, the resolution of

which is not addressed in [7]. This leaves SDA as the main example in this category.

SDA constructs an SPT in the first phase. It then incrementally schedules the nodes but also assigns them a parent, possibly different than in the one in the original SPT. The examples shown in Figures 1(c) and 1(d) are in fact two possible (distinct) trees/schedules that are generated by SDA algorithm with the input of SPT shown in Figure 1(c). Note the change of parents for nodes E and D in Figure 1(d) that is different than the original parents as shown in Figure 1(c). As we will later see, SDA is a well performing scheme but its drawback is the "distortion" caused by the scheduling phase, which means we cannot apply SDA's scheduler when we need to retain the aggregation tree exactly as supplied.

In contrast to SDA, the scheme called DAS by Yu et al. [8] retains the tree constructed in the first phase. The shortcoming of DAS is that its first phase constructs a DST (based on [9]) An example of such a tree is shown in Figures 1(b). Unfortunately, a DST is not necessarily a good selection for producing a better schedule, which has already been demonstrated through our examples. The reason is that DST tends to *cluster* many (children) nodes, with the rest of the nodes being a (smaller) set of clusterhead (parent) nodes, resulting in a large number of dependency constraints. That is, a clusterhead cannot aggregate before it receives from its children, and therefore its transmission cannot take place earlier than all of its (typically many) children transmissions. The result is long schedules.

Finally, other examples of two-phase approaches are the DST-based (via Maximal Independent Set construction) scheme by Wan et al. [10], and the SPT-based scheme by Annamalai et al. [11] (which has been exceeded in performance by [6]). Certain other efforts, such as [12], result in schedules that are potentially not conflict free, and have been left out of consideration for obvious reasons.

IV. Bounds and Tree Construction

We note that a tree construction phase, whether SPT, DST or any other kind of tree, ought to be guided by the potential it has to generate a short schedule. Until now, the two–phase schemes produced a tree based on topological properties alone. To the best of our knowledge, we are the first to perform the tree construction in a manner that is "informed" by the potential it has to result in a short schedule. To this end, and in contrast to the existing approaches, our tree construction specifically targets at "relaxing" the logical dependency constraints of the tree. The intuition being that the less constrained in terms of dependencies is the aggregation tree, the fewer the additional constraints (on top of the conflict freedom constraints), hence the potentially shorter the schedule.

We start by establishing, for a given tree, a lower bound on the schedule length. Then, we restrict ourselves to SPT trees and produce an SPT that follows this lower bound. As a side note, [10] provides an *upper* bound on the schedule length. But from the performance of their algorithm it is evident that this upper bound is far too pessimistic compared to the typical *practical* behavior of their algorithm. We take a different view of trying to squeeze the performance as close as possible to the lower bound corresponding to the constructed tree. Nevertheless, for comparison purposes, we will also evaluate the performance of [10] in our performance study Section VI.

A lower bound specifies the minimum number of slots that are required for convergecasting. Chen et. al. proposed the lower bound to be $max\{h, log_2 N\}$, where h is the longest path in a logical tree [6]. Similarly, Huang et. al. also argue that the data aggregation latency cannot be less than the network radius [7]. It basically means that the lower bound is the longest shortest path between the sink and a node in the network, i.e., h. Unfortunately, these lower bounds are loose. Consider the DST shown in Figure 1(b). Chen's and Huang's lower bound for this particular DST is 3, i.e., this particular DST can not be scheduled with less than three slots. However, it is very clear from the DST structure that node F will need at least 5 slots to send across its data to the root. In particular, three different slots are required by its children, i.e., nodes E, G and H. (That is because all of them have the same parent and therefore, there cannot be any concurrent transmissions among themselves.) Finally, since F is two-hop away from the root, it will need at-least two more slots (after receiving data from its children) to send across the aggregated data to the root, e.g., node F can use fourth slot and node C can use fifth slot. To generalize this observation, we introduce the following theorem.

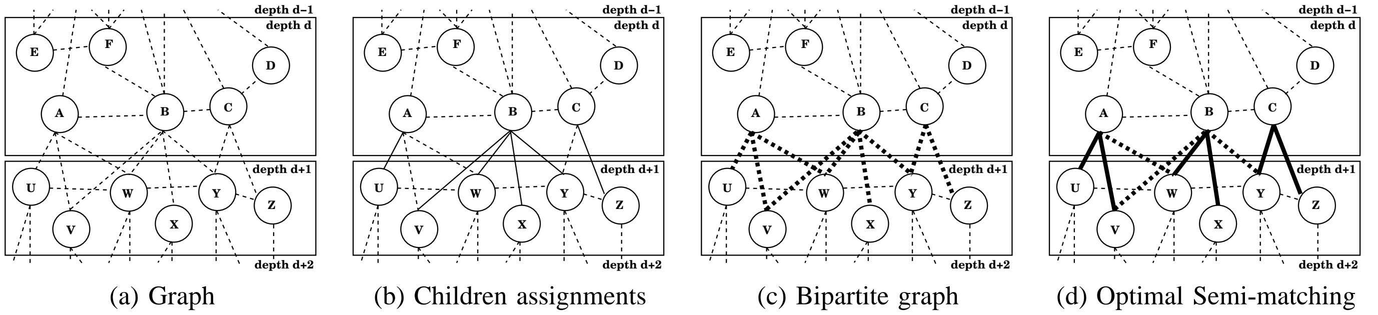
Theorem 1: Given a logical tree the lower bound, T_{min}, for the aggregation convergecast scheduling problem is $max\{\xi_i + h_i : i = 1, 2, ...N\}$, where ξ_i and h_i, respectively, are the children-count and hop-count (from the root) of node i in the given tree.

Proof: Let k be a node having the maximum sum of the children and hop count, i.e., $\xi_k + h_k = max\{\xi_i + h_i : i = 1, 2, ...N\}$. We will prove by contradiction that $T \geq max\{\xi_i + h_i : i = 1, 2, ...N\}$ for any schedule[1], $\mathcal{T}$, of the given tree.

Assume that $T < max\{\xi_i + h_i : i = 1, 2, ...N\}$. If k has been assigned the slot number $t(k)$, then it must be less than or equal to the total number of slots used i.e., $t(k) \leq T$. Since our assumption is $T < max\{\xi_i + h_i : i = 1, 2, ...N\}$, therefore, $t(k) < max\{\xi_i + h_i : i = 1, 2, ...N\}$. It also means that $t(k) < \xi_k + h_k$, which we get by replacing $max\{\xi_i + h_i : i = 1, 2, ...N\}$ with $\xi_k + h_k$. (Recall that k is the node with the maximum sum of the children and hop count, i.e., $\xi_k + h_k = max\{\xi_i + h_i : i = 1, 2, ...N\}$.) However, $t(k) < \xi_k + h_k$ contradicts the fact that k needs at least ξ_k slots for its children (for them to transmit first) and another h_k slots to send across its data to the root. It means that our assumption $T < max\{\xi_i + h_i : i = 1, 2, ...N\}$ must be incorrect. Therefore, $T \geq max\{\xi_i + h_i : i = 1, 2, ...N\}$, and hence the proof that $T_{min} = max\{\xi_i + h_i : i = 1, 2, ...N\}$. ∎

Next, we describe our procedure for constructing the aggregation tree based on the bound established by Theorem 1.

[1] That meets the criteria set-forth in Section II.



(a) Graph (b) Children assignments (c) Bipartite graph (d) Optimal Semi-matching

Fig. 2. Parent-children assignment during SPT construction. Rectangles contain nodes at a particular depth (from root) of the tree under construction. Dashed lines represent the graph (adjacency) edges. Solid lines represent parent-children assignments and also represent potential edges that can be selected in the tree construction. Dashed very-thick lines represent the edges in the bipartite graph formed between two consecutive depths of the tree. Solid very-thick lines represent the optimal semi-matching of the bipartite graph.

A. Balanced Shortest Path Tree

A tree that minimizes the lower bound potentially uses a lesser number of slots for scheduling N nodes. Consider the trees shown in Figures 1(b) and 1(c) that have a lower bound of 5 and 3 slots, respectively, as computed according to Theorem 1. In accordance with their respective lower bound, SPT used 6 slots as compared to 7 slots used by DST. This scenario exemplifies our observation that a tree, which "relaxes" the logical constraints, i.e., ξ_i and h_i, can indeed reduce the total number of slots that are required for scheduling the nodes. An obvious question now is how to construct such a tree that minimizes $max\{\xi_i + h_i : i = 1, 2, ...N\}$.

h_i for every node can be minimized by ensuring that every node is connected to the root using a shortest path, i.e., by constructing an SPT. A shortest path tree can be constructed using a standard Breadth First Search (BFS) algorithm. However, minimizing ξ_i is non-trivial. Consider the scenario of a shortest path tree construction as shown in Figure 2. A set of 12 nodes have been shown in Figure 2(a) at two consecutive "depths", d and $d+1$, of the tree. In particular, nodes from A to F are at distance d, and nodes from U to Z are at distance $d+1$ from the root (not shown in the figures). A possible scenario of parent-children assignments are shown in Figure 2(b). In this example children-count for nodes B and C have been minimized by assigning only one child to each one of them, i.e., nodes U and Z, respectively. However, this parent-children assignment has resulted in the assignment of four children to node B. In fact, the parent-children assignment shown in Figure 2(b) has maximized the "local" lower bound at depth d of the tree, i.e., $max\{\xi_A, \xi_B, \xi_C\} = 4$. (All the nodes at depth d will have the same hop-count, i.e., $h_A = h_B = h_C$, therefore, their respective children-count will actually determine the lower bound at depth d of the tree.) An increase in the lower bound at depth d of the tree may indeed result in the increased "global" lower bound of the tree. In general, we have the following sub-problem that needs to be solved.

Definition 1: If $\xi_{k,d}$ is the children-count of a node at depth d, the parent-children assignment problem then is to assign children to the nodes at depth d from the nodes at depth $d+1$ such that $max\{\xi_{k,d} : \forall k \geq 1\}$ is minimum.

Interestingly, the parent-children assignment problem defined above is equivalent to the problem of finding an optimal[2] semi-matching (with respect to L_∞ norm) for bipartite graphs [13]. A bipartite graph consisting of nodes from A to C and nodes from U to Z is shown in Figure 2(c). (Since nodes D, E and F do not have any neighbors from the nodes at depth $d+1$, they can not become parents and hence they are ignored. However, they will eventually become leaves of the shortest path tree). An optimal semi-matching is shown in Figure 2(d). It is optimal with respect to the number of assigned children, i.e., $max\{\xi_A, \xi_B, \xi_C\} = 2$ is minimum.

Using the basic idea depicted in Figure 2(d) we construct a "special" SPT in which optimal semi-matchings are obtained by constructing bipartite graphs with nodes at every two consecutive depths of SPT. Of-course that results in the minimization of $max\{\xi_{k,d} : \forall k \geq 1\}$ at every depth of SPT, which we will prove shortly in this section. We call an SPT for which parent-children assignments are balanced using optimal semi-matchings as a Balanced SPT (BSPT).

The pseudo code for BSPT is shown in Algorithm 1. ConstructBSPT is essentially a breadth first search algorithm with two new additions. As shown in Algorithm 1 at line 12 a bipartite graph is created from the nodes of two consecutive depth of the tree. At line 13 an optimal semi-matching is found for the corresponding bipartite graph using an algorithm from Harvey et. al. [13]. Parent-children assignments obtained through the semi-matchings, which will eventually become edges of the desired tree, are then added in the edge set of the tree under construction at line 14. This procedure is repeated until all possible parents are exhausted, i.e., the while loop at line 5. Finally the desired tree, BSPT, is produced at line 16.

Lemma 1: Given a graph, $G(V, E)$, ConstructBSPT produces an SPT with a minimum lower bound (as defined in Theorem 1) across all SPTs of G.

Proof: ConstructBSPT traverses the graph in a breadth first search manner, therefore, it minimizes the hop-count for every node, and hence produces an SPT. All parents at depth d of the SPT are assigned children from nodes at depth $d+1$ through an optimal semi-matching for the bipartite graph

[2]In the rest of the paper whenever we mention an optimal semi-matching, we mean that the semi-matching is optimal with respect to L_∞ norm.

Algorithm 1: $ConstructBSPT()$

```
Input: G(V, E), s_r ∈ V
Output: G'(V, E')
begin
1    P = {s_r}; /*Initialize the parent set with root*/
2    E' = ∅; /*An empty edge set for tree graph*/
3    G' = (V, E'); /*Initialize an empty tree graph*/
4    ∀v ∈ V Mark(v) = False; /*Unmark all nodes*/
5    while P ≠ ∅; do
6        C = ∅; /*Initialize an empty children set*/
7        ∀m ∈ P Mark(m) = True; /*Mark nodes that
         are being assigned*/
8        for all m ∈ P do
9            for all n ∈ N(m) do
                 /*Include every unmarked node in the
                 children set*/
10               if Mark(n) = False then
11                   C = C ∪ {n};
         /*Create the bipartite graph with the set of
         parent and children nodes*/
12       G_b = BipartiteGraph(P, C);
         /*Find an optimal semi-matching [13]*/
13       Z = FindSemiMatchings(G_b);
14       E' = E' ∪ Z; /*Update tree's edge set*/
15       P = C; /*Make the current children set as the
         next parent set*/
16   G' = (V, E'); /*Output the BSPT*/
end
```

formed by the nodes at depth d and $d+1$ of the SPT. An optimal semi-matching with respect to the L_∞ norm minimizes the maximum load [13]. It also means that if $\xi_{k,d}$ is the load of a node at depth d then $max\{\xi_{k,d} : \forall k \geq 1\}$ is minimum at depth d of the SPT. Because the hop-count for every node at depth d is same, therefore, $max\{\xi_{k,d} + h_{k,d} : \forall k \geq 1\}$ is also minimum at depth d of the SPT. Since optimal semi-matching is performed at every depth of the SPT, $max\{\xi_i + h_i : \forall i \geq 1\}$ is minimum for the SPT produced, and hence the proof. ■

It is worth noting that for a given graph ConstructBSPT generates an optimal SPT that has a lower bound as defined in Theorem 1, which is guaranteed to be minimum with respect to all possible SPTs that can be generated from that given graph. However, it does not guarantee a minimum lower bound (as defined in Theorem 1) with respect to trees that are *not* SPTs. It is conceivable that a tree can be constructed in which the paths of the nodes can be *elongated*, hence increasing their hop-count (compared to the shortest possible path) while possibly *decreasing* their children-count to potentially achieve an optimal lower bound as defined in Theorem 1. A more detailed study on this topic is part of our future study.

V. A Ranking Based Scheduling Algorithm

In this section we present an algorithm for scheduling the tree produced in section IV. As a matter of fact our algorithm can be used for any given tree to produce a conflict free schedule. Unlike some other proposals, e.g., SDA [6], our algorithm retains the structure of the input tree.

Our scheduling algorithm is fairly simple as compared to many other proposals. It takes a tree as input for which a schedule is desired and starts by considering all nodes in the tree that are *eligible* to be scheduled. Let E_j denote the eligible nodes for slot j. It is easy to understand that for the first slot all leaf nodes of the input tree are eligible to be scheduled. However, only a subset of the eligible nodes can actually be chosen for that particular slot in order to meet the conflict-free criterion, C4, set-forth in section II. How to choose a subset of nodes from all eligible nodes, so that the total number of slots being used can eventually be reduced, is a key step in our scheduling algorithm. The basic idea is to rank all eligible nodes in *decreasing* order of *weight*. We denote the sorted set of eligible nodes by S_{E_j}. We then check every eligible node (in the order it appears in S_{E_j}) if it can be scheduled in the given slot in a conflict-free manner. The chosen nodes are then *removed* from the tree. This procedure continues with the new set of eligible nodes (which may contain the nodes that have recently become eligible nodes and also the nodes that were previously eligible nodes but not yet scheduled) on the *resulting tree* until all nodes are scheduled. The ability of this process to generate good schedules in terms of length depends of course on the way the weights are assigned. We have experimented with many alternative weights and in the following we suggest which one works the best and the reasons behind it.

Let us denote by $s_{k,j}$ the k–th node in sorted order with respect to weights in S_{E_j} in the j–th slot. Let $w(s_{k,j})$ represent its weight. The higher weight gives a higher relative priority to a node to be scheduled in the current slot over other eligible nodes. If S_j represents the senders in the j–th slot, then for each slot we start with $S_j = \emptyset$, and add to it first $s_{1,j}$ (the highest weight, i.e., priority). After that, the second node $s_{2,j}$ is considered and added to S_j: iff by adding it to S_j we do not violate the conflict freedom constraint, C4, set-forth in section II. All other nodes (up to the $|S_{E_j}|^{th}$ node) are checked in the same fashion and added to S_j if they do not violate the conflict-free criterion. Finally, all nodes from S_j are assigned to transmit in the jth slot. The scheduled nodes are then removed from the tree, and the process repeats with the next set of eligible nodes until all nodes are scheduled.

In the rest of the paper we will call this framework for ranking and incremental scheduling as Weighted Incremental Ranking for convergEcast with aggregation Scheduling (WIRES). A concrete implementation of WIRES requires that we define a particular means to assign weights/priorities to eligible nodes.

A particular weight assignment that we use in WIRES is that of *non-leaf neighbor count*, $\eta(s_{k,j}) \subseteq \mathcal{N}(s_{k,j})$ of an

Algorithm 2: $WIRES()$

```
Input: G'(V, E'), s_r ∈ V
Output: T
begin
1    V' = V; /*Initialize set of nodes in the current tree*/
2    ∀v ∈ V' t(v) = 0; /*Initialize slots in T*/
3    j = 1; /*Initialize slot number*/
4    while t(s_r) = 0; do
5        E_j = ∅; /*Initialization the set of eligible nodes*/
6        for all v ∈ V' do
7            if ξ_v = 0 then
8                E_j = E_j ∪ {v}; /*Update the set*/
         /*Compute the weights*/
9        for all e ∈ E_j do
10           η(e) = ∅;
11           for all e' ∈ N(e) do
12               if e' ∈ V' and ξ_e' ≠ 0 then
13                   η(e) = η(e) ∪ {e'};
14           w(e) = |η(e)|; /*Assign weight (see text)*/
15       S_Ej = SortDecreasing(E_j); /*Do ranking*/
16       S_j = ∅; /*Initialize senders for the j^th slot*/
17       R_j = ∅; /*Initialize receivers for the j^th slot*/
18       FlagC4a = True; FlagC4b = True;
20       for all e ∈ S_Ej do
21           for all s ∈ S_j do
22               if e ∈ N(p(s)) then
23                   FlagC4a = False;
24                   break; /*Exit "for" loop*/
25           if FlagC4a = True then
26               for all r ∈ R_j do
27                   if p(e) ∈ N(r) then
28                       FlagC4b = False;
29                       break; /*Exit "for" loop*/
             /*Schedule only if C4 is satisfied*/
30           if FlagC4a = True and FlagC3b = True
             then
31               t(e) = j; /*Update the schedule T*/
32               S_j = S_j ∪ e; /*Update the senders*/
33               R_j = R_j ∪ p(e); /*Update the receivers*/
34               V' = V' \ {e}; /*Update nodes*/
35               E' = E' \ {(e, p(e))}; /*Update edges*/
36       T = j;
37       j = j + 1;
38   return T;
end
```

eligible node, $s_{k,j}$ in the resulting tree. Recall that as the eligible nodes are scheduled they are removed from the tree creating a *new* tree in which the non-leaf neighbors of a node may change. If $G'(V', E')$ is the current new tree, then $\eta(s_{k,j}) = \{e : e \in \mathcal{N}(s_{k,j}),\ e \in V' \text{ and } \xi_e \neq 0\}$. Intuitively, $S_{E_j} = \{s_{k,j} : |\eta(s_{k,j})| \geq |\eta(s_{k+1,j})|\ \forall k \geq 1\}$ ranks the eligible nodes in such a way that the most "constrained" nodes are scheduled first enabling many other nodes to become eligible nodes for the next available slots. A simple yet powerful feature of WIRES is that it considers many (eligible) nodes simultaneously to schedule for particular slots while increasing the probability of concurrent transmissions, and hence increasing the probability of using a lesser number of slots.

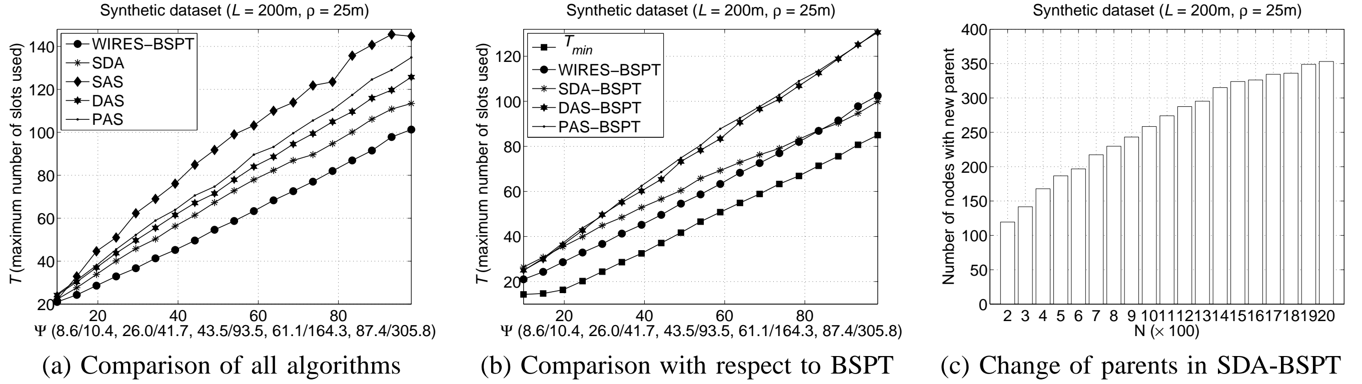
The pseudo code for WIRES is shown in Algorithm 2. All nodes are initialized with slot number 0 at line 2. The set of eligible nodes for the jth slot, and the weights are computed at lines 5∼14. After sorting the eligible nodes according to weight at line 15, each one of them is considered for the $j^{th} slot$ (lines 20∼35). Only those eligible nodes are finally allocated the jth slot (line 31) that do not violate the conflict-free criteria (line 30). The eligible nodes that are finally scheduled are removed from the tree (lines 34- 35). This procedure continues until the root is scheduled (line 4). (The root, s_r, transmission in the j–th slot is a pseudo-transmission, which is not needed, it just signifies that the schedule was completed in $j - 1$ slots.)

Lemma 2: A schedule produced by WIRES is conflict-free.

Proof: Every slot in WIRES is allocated in an incremental fashion, i.e., S_j is initiated as an empty set and then eligible nodes are added into it incrementally. Since any node is added to S_j only if that node meets the conflict-free criteria, S_j will contain the set of nodes that do not interfere with each other's transmissions. Because this procedure is repeated for every slot, $j \geq 1$, S_j will always contain nodes that do not interfere with each other's transmissions, and hence the proof. ∎

VI. Performance Evaluation

To evaluate our proposal we implemented SDA [6], PAS [5], DAS [8], SAS [10] and First-Fit [7] algorithms to compare their performance with WIRES-BSPT. We discovered that First-Fit algorithm does not produce a conflict-free schedule, which has also been noted in [8]. Therefore, in our evaluations we omit the results of First-Fit algorithm. It is interesting to note that all existing proposals have been tested under drastically different simulation setups. For example, Chen et. al. [6] assumed a network of 100 nodes in a 200m×200m area. Various topological scenarios were created by varying the transmission range of nodes between 21.7m and 40m. In contrast, Yu et. al. [8] used 1000 to 2000 nodes, with a radio range of 25m, in an area of 200m×200m area. These two scenarios exemplify the extreme variations in the simulation setups being used by various studies. Choosing a particular "representative" setup for our study was problematic. To solve this problem we used the *density* metric to provide a "common platform" to test all algorithms as fairly as possible. More specifically we define the density to be: $\Psi = \frac{\pi\rho^2 N}{L^2}$, where N is the number of nodes, ρ is the transmission range of nodes and L is the length of a square area. By varying density we have essentially captured the "essence" of various setups



(a) Comparison of all algorithms (b) Comparison with respect to BSPT (c) Change of parents in SDA-BSPT

Fig. 3. Synthetic dataset. In the "( )" with Ψ we have provided average-degree/degree-variance of the nodes.

being used in other studies. In our experiments we kept ρ and L fixed at 25m and 200m, respectively, while varied N to change the density. The reported results are an average of 10 runs. A unique node is chosen randomly as the root in each of these runs. As a side note, even though we conduct the performance evaluation using UDGs, the choice is purely motivated for comparison purposes with existing literature that already assumes UDGs. With very few exceptions the algorithms in the literature are applicable to general topology graphs and not to UDGs alone.

We also performed experiments with real data by using the Intel Berkeley dataset (http://db.csail.mit.edu/labdata/labdata.html). This dataset contains of approximately 3.5 million readings from 54 sensors deployed in the Intel Berkeley Research lab. Though the sensor readings are of no use for this study, we used sensors' positions, which were available along with the original dataset, to create various topologies. A few sensors for which the location was not know were removed from the dataset. Since the number of sensors was fixed in the Intel dataset, ρ is varied to create scenarios with various densities. In each of these scenarios one unique node is chosen randomly as the root. Again, the reported results are an average of 10 runs. To the best of our knowledge none of the previously proposed algorithms have been evaluated using real sensor location positions, such as the Intel Berkeley setup.

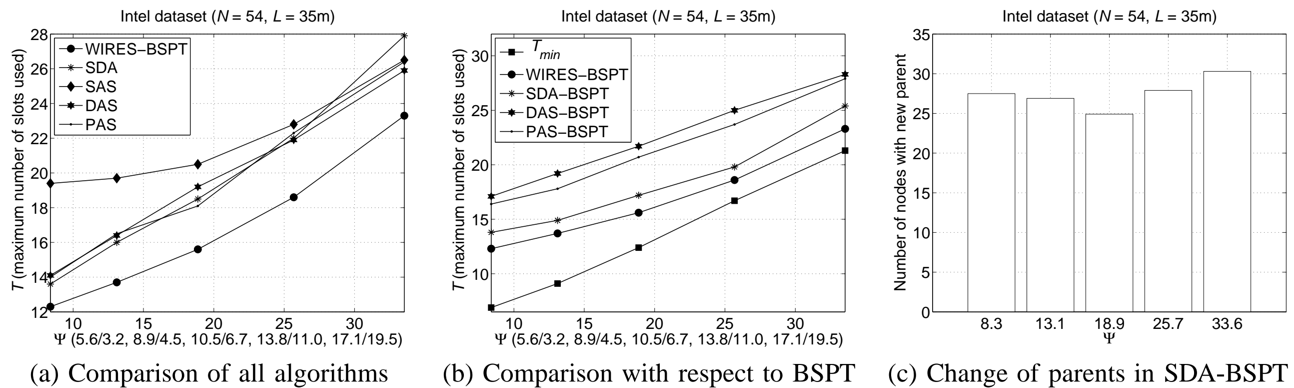
The first set of results from the synthetic dataset are shown in Figure 3. As shown in Figure 3(a) we can see that WIRES-BSPT outperforms all other solutions by 10$\sim$30%, which means that our solution will require that much less time for its schedule. More interestingly, as the density increases, the performance of WIRES-BSPT is improved compared to other approaches. The reason for this improved performance is that as the density increases the underlying tree, BSPT, becomes more "bushy". It also means that, on average, the number of children per parent is increased substantially. The way BSPT is constructed it tends to spread the load (children) among the parents evenly which in turn "relaxes" the logical constraint for these parents. That results in two prominent affects for WIRES. (1) It increases the probability that a parent may become eligible node much faster. (2) As more parents become eligible, they significantly increase the "pool" of eligible nodes providing more opportunity for concurrent transmissions.

Since SDA, DAS and PAS can work independently as standalone algorithms for scheduling, we ought to evaluate their performance with respect to BSPT. We provided BSPT as input tree to each one of these algorithms, i.e., SDA-BSPT, DAS-BSPT and PAS-BSPT are compared with WIRES-BSPT. The results are shown in Figure 3(b). These result confirm that the choice of the tree is important. We can see that SDA, DAS and PAS are able to schedule the nodes using a lesser number of slots. Their performance is improved by just replacing the tree with which they were originally proposed and evaluated, by BSPT ! In addition, the combined WIRES-BSPT still performs better than the all the rest. However, as Ψ increases the performance gap between WIRES-BSPT and SDA-BSPT shrinks. At a very high density, i.e., $\Psi \geq 80$ the difference between WIRES-BSPT and SDA-BSPT becomes negligible. However, SDA-BSPT has its own limitation, i.e., it does not retain the input tree.

Figure 3(c) summarizes the "side affects" of SDA-BSPT solution. In this figure we show the average number of nodes that have been assigned new parent, which are different from the parents in the original BSPT. In particular, when the total number of nodes are 200, then more than 50% of the nodes are assigned new parent. When the node density increases (as the number of nodes increases) the total number of nodes with new parents also increase. These results suggest that the changes inflicted by SDA algorithm on the input trees are substantial.

The second set of results from Intel dataset are summarized in Figure 4. Here again the qualitative behavior of the results remain same as seen in the results from the synthetic dataset. However, their quantitative behavior has changed. As shown in Figure 4(a) we can see that WIRES-BSPT outperforms all other solutions but by slightly smaller margin, i.e., by 10$\sim$18%. Figure 4(c) summarizes the performance of the algorithms when BSPT is the input tree.

An interesting result that did not appear in the synthetic dataset is that performance gain of SDA with respect to DAS and PAS is negligible. As a matter of fact it is outperformed, though slightly, by DAS and PAS when density is beyond



(a) Comparison of all algorithms (b) Comparison with respect to BSPT (c) Change of parents in SDA-BSPT

Fig. 4. Intel dataset. In the "( )" with Ψ we have provided average-degree/degree-variance of the nodes.

25. This experiment reveals SDA's sensitivity towards particular topologies. However, WIRES-BSPT consistently performs better than all other approaches. Figure 4(c) summarizes the performance of SDA with respect to the changes in the output tree. As shown in these results, SDA retains only 50% (on average 27 nodes out of total 54 nodes) of the original parents in the output tree.

The summary of these results is that if the underlying tree is not important and the only objective is to reduce the number of slots, then WIRES-BSPT performs better in most of the cases. Though at higher densities WIRES-BSPT and SDA-BST offered equally good solutions. Nevertheless, if the underlying tree is important (i.e., if we have constructed a tree that is important from the application's perspective), then WIRES is the most efficient solution. Another note here is that Figure 3(b) and Figure 4(b) also show the results of the lower bound on BSPT computed in accordance with Theorem 1. These experimental results verify the correctness of the proof.

VII. Conclusion

Previous aggregation convergecast scheduling solutions rely on ad-hoc approaches to create the aggregation dependency tree before applying their scheduling algorithms. Some of the proposed solutions even change the tree, hence their usefulness is not obvious to applications that wish to retain the aggregation tree intact. Some of the previously proposed solutions are not even conflict-free. We have presented several contributions in this paper. First, we proposed a tighter lower bound to the tree scheduling problem, and proved its correctness. Second, we proposed an algorithm to construct a logical tree, BSPT, guided by the lower bound, allowing the generation of schedules with fewer slots. Third, we proposed a ranking/priority–based scheduling algorithm, WIRES, that produces schedules, that are guaranteed to be conflict-free. Our proposal was evaluated extensively using synthetic and real datasets. Our proposed algorithms are efficient and can save up to 15% of the scheduling time. Future directions include increasing the robustness of in-network aggregation, i.e., to produce reliable aggregation convergecast. Another extension is that of considering aggregation queries on subsets of the nodes, instead of on all the nodes in the network.

Acknowledgment

Thanks are due to Xujin [6], Bo [8], Jianming [12], Scott [7] and Nicholas [13] for providing some important clarifications regarding their work to the authors of this paper. This research is partially supported by NSERC.

References

[1] W. H. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," *Proc. of the 33rd Annual Hawaii Intl. Conf. on System Sciences*, vol. 2, pp. 1–10, 2000.

[2] A. Kesselman and D. R. Kowalski, "Fast distributed algorithm for convergecast in ad hoc geometric radio networks," *Journal of Parallel and Distributed Computing*, vol. 66, no. 4, pp. 578–585, 2006.

[3] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TAG: a tiny aggregation service for ad-hoc sensor networks," *Proc. of the 5th symposium on Operating systems design and implementation (OSDI'02)*, vol. 36, pp. 131–146, 2002.

[4] B. Malhotra, M. A. Nascimento, and I. Nikolaidis, "Monitoring exact top-k values in wireless sensor networks using dominating set trees," Computing Science, Univ. of Alberta, Tech. Rep. TR09-01, 2009.

[5] X. Yu, S. Mehrotra, and N. Venkatasubramanian, "Sensor scheduling for aggregate monitoring in wireless sensor networks," *Proc. of the 19th Int. Conf. on Scientific and Statistical Database Management (SSDBM'07)*, p. 24, 2007.

[6] X. Chen, X. Hu, and J. Zhu, "Minimum data aggregation time problem in wireless sensor networks," *Lecture Notes in Computer Sciences*, vol. 3794, pp. 133–142, 2005.

[7] 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," *Proc. of the 26th Conf. on Computer Communications (INFOCOM'07)*, pp. 366–372, 2007.

[8] B. Yu, J. Li, and Y. Li, "Distributed data aggregation scheduling in wireless sensor networks," *Proc. of the 28th Conf. on Computer Communications (INFOCOM'09)*, 2009.

[9] P. J. Wan, K. M. Alzoubi, and O. Frieder, "Distributed construction of connected dominating set in wireless ad hoc networks," *Mobile Networks and Applications*, vol. 9, no. 2, pp. 141–149, 2004.

[10] P.-J. Wan, S. C.-H. Huang, L. Wang, Z. Wan, and X. Jia, "Minimum-latency aggregation scheduling in multihop wireless networks," *Proc. of the 10th ACM Int. Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC'09)*, pp. 185–194, 2009.

[11] V. Annamalai, S. Gupta, and L. Schwiebert, "On tree-based convergecasting in wireless sensor networks," *Proc. of the IEEE Conf. on Wireless Comm. and Networking (WCNC'03)*, vol. 3, pp. 1942–1947, 2003.

[12] J. Zhu and X. Hu, "Improved algorithm for minimum data aggregation time problem in wireless sensor networks," *Jour. of Systems Science and Complexity*, vol. 21, no. 4, pp. 626–636, 2008.

[13] N. J. A. Harvey, R. E. Ladner, L. Lovász, and T. Tamir, "Semi-matchings for bipartite graphs and load balancing," *Jour. of Algorithms*, vol. 59, no. 1, pp. 53–78, 2006.