

Distributed Minimal Time Convergecast Scheduling for Small or Sparse Data Sources

Ying Zhang
Palo Alto Research Center Inc.
3333 Coyote Hill Rd
Palo Alto, CA 94304, USA
Emails: yzhang@parc.com

Shashidhar Gandham
xG Technology, LLC
240 South Pineapple Avenue
Sarasota, FL 34236, USA
Email: shashig@xgtechnology.com

Qingfeng Huang
Palo Alto Research Center Inc.
3333 Coyote Hill Rd
Palo Alto, CA 94304, USA
Emails: qhuang@parc.com

Abstract—Many applications of sensor networks require the base station to collect all the data generated by sensor nodes. As a consequence many-to-one communication pattern, referred to as convergecast, is prevalent in sensor networks. In this paper, we address the challenge of fast and reliable convergecast on top of the collision-prone CSMA MAC layer. More specifically, we extend previous work by considering the following two situations: (1) the length of the packets generated by nodes is much smaller than the maximum length of a data frame that can be transmitted in one time slot and (2) not every node in the network has data to transmit and for those that have, may have lots of data that require more than one packet. The first situation leads to the possibility of improvement by data piggybacking/aggregation; the second scenario arises in networks where nodes locally store the data and serves query request on-demand. We present distributed minimal time scheduling algorithms for both the cases. Simulation results have shown significant performance improvements of our new approaches over existing solutions.

I. INTRODUCTION

Many applications of sensor networks involve collection of all the data generated by sensor nodes at the base station. Examples of such applications include: collecting global snapshot of the network state, monitoring the residual energy of nodes [19] and localizing a sniper [22] in urban environment. Hence, many-to-one communication pattern, referred to as convergecast, is prevalent in sensor networks. Convergecast in sensor networks involves a large number of the nodes forwarding data to the base station in a relatively short interval of time. This “bursty” nature of convergecast leads to high probability of collision and data loss in the network, particularly when one adopts contention-based MAC protocols like CSMA [1] for its simplicity and low overhead. Recovery methods such as retransmissions increase the convergecast latency and drain the scarce energy reserves of the nodes. Moreover, retransmissions might lead to further collisions in high data rate scenarios [6]. Radially coordinated transmission [12] was proposed to address the issues associated with collisions. Reliable bursty convergecast [8] employs retransmission-based error control and scheduling of retransmissions. However, the latency incurred by both approaches is far from the optimal. As a result, bounded convergecast latency and reliable convergecast packet delivery, like those in real-time one-to-one communication scenarios [10], [23], are challenges of paramount importance

in mission critical applications, e.g., surveillance and security applications.

In our earlier work [20], we developed an optimal distributed convergecast scheduling algorithm based on the following assumptions: (1) MAC layer data-frame is long enough to forward exactly one packet in a time slot and (2) every node has exactly one packet to be forwarded to the base station. We have proved that our algorithm is optimal for tree networks and requires at most $\max(3n_k - 1, N)$ time slots. Here, N represents the number of nodes in the network and n_k represents the maximum number of nodes in any subtree. Although our algorithm requires at most $3N$ time slots for a network of general topology, simulation results have shown that the number time slots actually used is about $1.5N$. Furthermore, the sleep schedule we proposed conserves at least 50% of the energy spent in convergecast.

In this paper, we consider the following generalizations: (1) packet aggregation when the size of data is much smaller than the size of the data frame, and (2) some nodes have several packets to be forwarded to the sink-node/base-station and the rest do not have any data to report to the base station.

The first generalization is applicable for many sensor network applications, where the size of data is about a couple of bytes and the maximum data-frame length supported by the MAC layer is much larger. For example, the acoustic sensor reading used in the shooter localization application [22] requires six bytes and the header size of a data frame in TinyOS MAC is six bytes [13]. As a result, 50% overhead is incurred in forwarding a packet at every hop on its way to the base station. The total size of the data frame supported by TinyOS MAC layer is 36 bytes. By aggregating multiple packets into a single data-frame the overhead can be reduced significantly.

The second generalization is applicable to sensor network deployments wherein nodes maintain a log of the data corresponding to events detected. Nodes forward the data to the base station only on receiving a query. Note that in this on-demand service scenario, some nodes could have lots of data to report and some may have none. This is also true in situations where sensor tasking (sampling rate) varies across the network for energy efficiency purposes.

The important contributions of this paper are:

- A distributed convergecast scheduling algorithm with packet aggregation. The proposed algorithm significantly decreases the number of time slots required for convergecast in linear and mesh networks.
- A distributed convergecast scheduling when not all nodes in the network have packets to be forwarded to the base station. The proposed algorithm is applicable even when nodes have multiple packets to be sent to the base station. We show that the proposed algorithm is optimal for tree networks. For a network of general topology, this algorithm significantly improves the performance over the previous scheduling algorithms.

The remaining part of this paper is organized as follows: System model considered for this work and the problem formulation are described in Section II. Section III reviews convergecast scheduling algorithms for cases where every node generates one data packet. The scheduling algorithm with packet aggregation for small data packets is described in Section IV. The scheduling algorithm for sparse but large number of data packets is discussed in Section V. Section VI compares the performance of these two new convergecast scheduling algorithms with the previous algorithms. Section VII concludes the paper.

II. MODEL AND PROBLEM FORMULATION

We consider sensor networks wherein the nodes and the associated base station are static. All the nodes are equipped with a single omni-directional transceiver¹. Hence, the nodes (including the base station) cannot transmit and receive at the same time. All the communications are carried over a single frequency band. The bandwidth of every wireless link in the network is assumed to be the same. We assume that the network connectivity is fixed over time, i.e., no power control such as in FPAS [16] is applied. No specific assumptions, like unit-disk radio range model, are made about the propagation characteristics of the wireless medium. However, we consider only symmetric links for scheduling. Initially, we assume that transmission range of a node is equal to its interference range. Relaxation of this assumption will be discussed. We assume that the maximum length of a packet is fixed. We consider applications wherein the base station has to receive all the data generated by the nodes, no data aggregation is considered. However, we do consider *packet aggregation* where a set of packets are assembled into one packet before transmission. Just like most TDMA schedules (e.g. [7], [17], [18], [9]), we assume global time synchronization and a time slot is the time to send one packet over the radio.

Problem Formulation: Let $G(V, E)$ represent the sensor network where (i) $V - \{s\}$ represents the set of sensor nodes, (ii) s represents the base station or sink and (iii) $E \subset V \times V$ represents the set of wireless links. Let $p_v(0)$ be the number of packets that originate at a node v , and $p_v(j)$ be the number of packets at node v in time slot j . Let $f_v : T \rightarrow V$ represent

the action of node v at different time slots; $f_u(j) = v \wedge u \neq v$ implies that node u transmits a packet to v at time slot j ; $f_u(j) = u$ implies that u does not transmit in time slot j . We need to identify a schedule $f_v, v \in V - \{s\}$ such that:

- 1) $p_s(l) = \sum_{v \in V - \{s\}} p_v(0)$. All the packets should reach the sink by the last time slot l .
- 2) If $f_u(j) = v$ (u transmits a packet to v in time slot j) then:
 - $(u, v) \in E$. The edge (u, v) exists in the network.
 - $p_u(j-1) > 0 \wedge p_u(j) = p_u(j-1) - 1 \wedge p_v(j) = p_v(j-1) + 1$. Node u has at least one packet at the end time slot $j-1$ and exactly one packet is transmitted along the edge (u, v) in time slot j .
 - $f_v(j) = v$ and $f_w(j) = w \forall (w, v) \in E, w \neq u$. Node v and the neighbors of v , except u , cannot transmit in time slot j .

The objective is to minimize l , the total number of time slots used for convergecast. The convergecast scheduling problem is known to be NP-complete [11]. Our approach for this problem is first to consider networks of simple topology, such as linear, multi-line and trees, for which distributed optimal-time algorithms can be derived. For a network of arbitrary topology, a BFS tree is built first, and the algorithms applied to trees are used with the filtering of conflicting branches. Although the algorithms are no longer optimal for general networks, they have a provable time-bound that is independent to the density of the network.

Our approach is inspired by and is very similar to PDASN [4], [5], except that we provide distributed algorithms for *data collection*, while PDASN only discusses the algorithms for the base station for *data distribution*. Although data distribution is the opposite to data collection, the distributed algorithms require computing only limited information at each node. Our schedule is generated *on-line* during data transmissions, although parameters used in the schedule are computed distributedly in the initialization process. No node, including the base station, has the complete topology of the network.

Another related work to this research is FPS [3], [2]. Unlike FPS where it assumes that the demands change over time and are not predicable, we assume each node knows its own demand at the initialization, or at the time receiving a query. Therefore, our scheduling is for a fixed demand distribution, and time optimal or near-optimal for that distribution. After initialization, our scheduling has no other control packets such as advertisement, request and confirmation in FPS. Although data loss is inevitable due to link failure, our algorithms guarantee *no data loss due to collisions* and perform well for any network topology.

In the rest of this paper, we first review the distributed minimal-time scheduling algorithm that assumes every node generates exactly one packet. Then we focus on the major topics of this paper: (1) convergecast scheduling with packet aggregation, and (2) convergecast scheduling when nodes generate none or multiple packets.

¹Song [25] recently discovered the similar bound as the one in [20], although his network assumptions are different from ours.

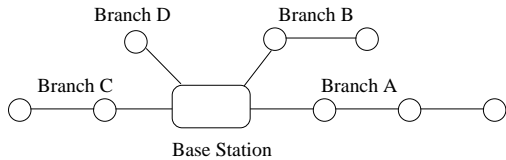


Fig. 1. A Multi-line Network.

III. BASIC CONVERGECAST SCHEDULING

The scheduling algorithm for each node having one packet is derived by first considering a *linear* network of nodes and then developing a scheduling for a *multi-line* network that can be reduced into multiple linear networks with the sink as the point of intersection. A *tree* network can be reduced into an equivalent multi-line network, thus, resulting a convergecast schedule for tree networks. For a general network, a breath-first tree rooted at the sink will be built first and then the scheduling algorithm for tree networks will be applied.

A. Linear Networks

A linear network consists of a linearly connected set of nodes, with the sink at one end. Given N nodes, the hop-count from a node to the sink ranges from 1 to N . During initialization, the hop-count to the sink is obtained at each node. During convergecast a node will be in one of the following states: (1) T : the node can transmit, (2) R : the node may receive from a neighboring node, and (3) I : the node neither transmits nor receives.

Every node is assigned an initial state based on its hop-distance from the base station. Starting with the first node every third node (i.e., 1^{st} , 4^{th} , 7^{th} , ...) is in initial state T . Starting with the second node every third node (i.e., 2^{nd} , 5^{th} , 8^{th} , ...) in the network is assigned I as its initial state. Starting with third node from the base station every third node (i.e., 3^{rd} , 6^{th} , 9^{th} , ...) is assigned state R before the start of the convergecast. All nodes move from one state to another in the subsequent time slots according to the state transition $T \rightarrow I \rightarrow R \rightarrow T$.

It has been shown [20] that the above-described convergecast algorithm is near-optimal and requires exactly $3N - 2$ time slots (one slot more than the optimal $3N - 3$). An interesting property of the above algorithm is that nodes require to buffer at most two packets in any time slot, given that each node has one packet initially.

B. Multi-line Networks

A naive way to schedule transmissions in a multi-line network (Fig. 1) would be to schedule transmissions along one branch at a time. However, scheduling multiple branches *concurrently* would require relatively fewer number of time slots to complete convergecast. Note that the base station is equipped with a single transceiver. Hence, at most one packet can be received by the base station in a time slot. Convergecast scheduling in multi-line networks involves deciding the branch along which a packet can be forwarded to the sink in every time slot.

During the initialization, each node obtains its hop-count to the sink and determines its initial state as described in Section III-A. The sink assigns an unique ID to every branch. The information about the number of nodes in each branch is collected and disseminated to all the nodes by the sink. After the initialization phase, each node is aware of (1) its initial state (T , I , or R), (2) its branch ID and (3) the number of nodes, or equivalently, the number of time slots, in each branch of the network. Note that no node, including the sink, needs to be aware of the entire network topology. The total number of transmissions for this initialization process is $3N + k$ for k branches: $N + k$ for nodes to obtain the hop-counts and branch IDs, N for the sink to collect the number of slots S_i from each branch i , and N for the sink to distribute $\{S_i\}$ to each node.

Let S_i be the number of time slots left to schedule and L_i be the last time slot scheduled for branch i , respectively. Let S_i^0 be the initial value of S which is the total number of time slots for branch i and $L_i^0 = 0$. At any time slot t , a branch i is called *eligible* if $t > L_i$. An *eligible* branch with highest number of time slots left is given priority to be scheduled to forward the packets. In the event of a tie the branch with lowest ID is given preference. Let i be the branch selected, $S_i \leftarrow S_i - 3$ and $L_i \leftarrow L_i + 2$, i.e., each time three time slots will be scheduled for a branch. Every node updates these information and make decision independently at every time slot.

A node with branch ID i is said to be *active* at t , if $t \leq L_i$. An active node changes its state as per the state transition $T \rightarrow I \rightarrow R \rightarrow T$.

It has been shown in [20] that the number of time slots required by the above described convergecast-scheduling algorithm for multi-line networks is $\max(3n_k - 1, N)$, where N represents the number of nodes in the network and n_k represents the maximum number of nodes in a branch.

C. Tree Networks

In the rest of the paper we use the term *one-hop-subtree* to refer to any subtree that is rooted at a one-hop neighbor of the sink. The convergecast scheduling algorithm for tree networks is based on the observation that a tree network can be reduced to a multi-line network with each line represented as a combination of line branches of nodes. For example, the tree shown in Fig. 2(a) can be represented as a two-line network: (i) $b-c-d$ and $b-e-f$ rooted at b , and (ii) $g-h$ and $g-i-j$ rooted at g .

Let v be any node in the tree. Let W_v be the number of packets that have to be forwarded by the corresponding subtree before node v becomes *active*. Let v have k child nodes v_1, v_2, \dots, v_k . Let N_1, N_2, \dots, N_k respectively be the number of nodes in each subtree rooted at v . Then $W_{v_1} = W_v$; i.e., v_1 (the first child node) becomes active along with v . $W_{v_2} = W_v + N_1$; i.e., node v_2 becomes active after all the packets in subtree rooted at v_1 have reached the node v . Similarly, $\forall i \leq k$ $W_{v_i} = \sum_{j=1}^{i-1} N_j + W_v$. To schedule in a tree network, nodes coordinate to determine their respective W_v values in the initialization phase. For example, an in-order

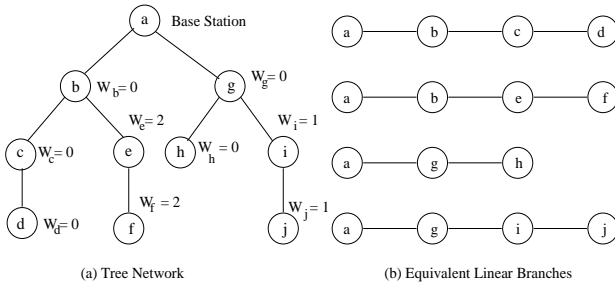


Fig. 2. Reduction of a Tree Network into Line Branches.

tree traversal [15] initiated by the sink can be used. Fig. 2 shows the value of W_v at each node. The *active time slot window* of a node v is $[3W_v + 1, 3(W_v + N_v)]$ which can be set at each node before the convergecast starts.

Note that all the one-hop-subtree (e.g. subtrees rooted at nodes b and g in Fig. 2) can be scheduled parallelly as in a multi-line network (see Section III-B). Each one-top-subtree will correspond to a branch in the multi-line network. An *eligible one-hop-subtree with highest number of time slots left will be scheduled in each time slot*; a one-hop-subtree i is said to be *eligible* in time slot t if $t > L_i$. A one-hop-subtree i is *active* if $t \leq L_i$. A node with active window $[w_1, w_2]$ is *active* if its one-hop-subtree is active and $w_1 \leq t_b^i \leq w_2$. Where, t_b^i is the branch time of this one-hop-subtree, which is increased by one every time this one-hop-subtree is active.

It has been shown in [20] that the number of time slots required by the above-described convergecast scheduling algorithm for tree networks is at most $\max(3n_k - 1, N)$, where n_k represents the number of nodes in the largest one-hop-subtree. For example, the network shown in Fig. 2 can be viewed as a two-line network, and 14 ($n_k = 5$) time slots are required to complete convergecast. For tree networks, in addition to the information required for multi-line networks, each node v has to know its active window. Again, no node has information about connectivity of the whole network, and the initialization cost is $3N + k$ transmissions.

D. General Networks

For networks of general topology, a breadth first search (B.F.S.) spanning tree is constructed first and then the convergecast scheduling algorithm described in Section III-C is applied to the spanning tree.

In a general network, we need to consider the interference due to edges that are not part of the spanning tree. Given a spanning tree, we classify the edges of a network as *spanning tree edges* and *non-spanning tree edges*. As the names suggest, spanning tree edges (resp. non-spanning tree edges) are part of (resp. not part of) the constructed spanning tree.

Two one-hop-subtrees, say A and B , are said to be *interfering* if there exist a non-spanning tree edge (a, b) such that a is in A and b is in B ; (A, B) is referred to as a *conflicting-subtree-pair* and the edge (a, b) is referred to as an *interfering-edge*. If multiple one-hop-subtrees are scheduled simultaneously, interfering-edges might result in collisions.

Hence, in a time slot t we schedule an *eligible one-hop-subtree* which has the maximum number of packets left and does not interfere with active one-hop-subtrees.

During the initialization, nodes announce their one-hop-subtree ID to all their neighbors. As a result, nodes can learn about existence of conflicting-subtree-pairs and report to the sink. For example, if node a in one-hop-subtree A discovers that its neighbor b belongs to a different one-hop-subtree B then a reports the conflicting subtree pair (A, B) to the sink. After the sink receives all the conflicting subtree pairs, a *conflict map* M is disseminated to each node in the network; $M(A, B) = 1$ if (A, B) is a conflict-subtree-pair and $M(A, B) = 0$ otherwise. Using the conflict map M , each node independently schedules its transmissions. Note that for general networks, in addition to the information for tree networks, each node has to know the conflict map M at the initialization phase, which introduces extra communication cost. However, again, no node knows the global connectivity of the network.

Let $\mathcal{J} = \{i | t \leq L_i\}$ be the set of active one-hop-subtrees at time slot t and let $\mathcal{L} = \{i | L_i < t \wedge \forall j \in \mathcal{J}, M(i, j) = 0\}$ be the set of *eligible* one-hop-subtrees to schedule. We will refer to this filtering (mechanism of not scheduling conflicting-subtree-pairs in parallel) as *conflicting resolution* (CR). As this filtering mechanism is conservative, the resulting schedule might not be optimal. However, it has been shown in simulations that for a network of N nodes about $1.5N$ time slots are actually needed using this convergecast algorithm. This is about half of $3N$, the upper bound on the number of time slots required by this algorithm [20].

E. Discussion

Note that the TDMA schedule is built *on-line* with data transmission at each node, not built and transmitted from the base station to each node. The initialization for distributing parameters for the schedule is relatively expensive, if the convergecast scheduling is used only once. However, in most cases, we assume the network is stable and the convergecast happens periodically or whenever events are detected.

Also, even data are not distributed uniformly across the network, such a collision-free schedule can still be used, although time-optimality cannot be hold.

Although the scheduling does not consider packet loss for unliable links, one can use implicit confirmation (i.e., over-hearing the transmission of its parent) to estimate packet loss and retransmit the packet at next time slot if no confirmation has been heard. For permanent node failures, e.g. out of battery, this approach will fail to work. How to repair a schedule (instead of building a new one from scratch) in permanent node failures is an interesting research for the future.

A sleep schedule can be added so that only nodes that are in active branches and within its active window are awake. With such sleep scheduling, more than 50% energy can be saved [20], assuming transmitting, receiving, and idling have similar energy consumption [3].

							Timeslot	↓
d	e	f	g	h	i		Status	
I	I	I	I	I	I	I	1	0
1	1	1	1	1	1	1	Num. Pkts	
R	I	T	→ R	I	T	→	Status	1
1	1	0	2	1	0	0	Num. Pkts	
T	→ R	I	T	→ R	I	T	Status	2
0	2	0	0	3	0	0	Num. Pkts	
I	T	→ R	I	T	→ R	T	Status	3
0	0	2	0	0	3	3	Num. Pkts	
R	I	T	→ R	I	T	→	Status	4
0	0	0	2	0	0	0	Num. Pkts	

Fig. 3. Scheduling in Line Networks with Packet Aggregation.

IV. PACKET AGGREGATION

For many sensor network applications, the amount of data generated by each node is small. To illustrate the advantages of packet aggregation, consider a linear network. We use the same scheduling algorithm described in Section III-A. However, instead of transmitting one packet in each time slot we aggregate up to *three* original packets and transmit in one time slot. Fig. 3 shows the first four time slots of such a scheduling. “Num.Pkts” in the Figure means the number of before-aggregation data “packet”. It can be observed that after receiving a packet, a node transmits in the next time slot. Hence, we can show that after the first three time slots, an aggregated packet traverses one hop in each time slot. For linear networks the following theorem holds:

Theorem 1: The convergecast scheduling algorithm using packet aggregation requires at most $N + 2$ time slots in a linear network with N nodes.

Proof: Note that the sink receives the first packet in the first time slot. Then in every three time slot it receives an aggregated packet which contains data from three original un-aggregated packets. Hence, the total number of required time slots is $1 + 3\lceil \frac{N-1}{3} \rceil \leq N + 2$. ■

For multi-line networks, the scheduling algorithm using aggregation is similar to that described in Section III-B with the following modifications: S_i is initialized to be the total number of time slots needed to finish the convergecast with aggregation, i.e., $S_i \leftarrow 1 + 3\lceil \frac{n_i-1}{3} \rceil$ given n_i as the total number of nodes in branch i .

Theorem 2: If N represents the number of nodes in the network, k is the number of branches and n_k represents the maximum number of nodes in a branch, then the number of time slots required by this aggregation-enabled convergecast-scheduling algorithm for multi-line networks is given by

$$\max(n_k + 3, \lceil \frac{N + 2k}{3} \rceil).$$

Proof: Let n_i represent the number of nodes in branch i , and $n_k \geq n_{k-1} \geq n_{k-2} \dots \geq n_1$. When there are at most two branches in the network, it is easy to show that the schedule determined by our algorithm requires at most $n_k + 3$ time slots,

since the longer branch can always be active and the shorter one will be active after first slot. If both have equal length, it is $n_k + 2 + 1$. From hereon, we consider networks which have at least three branches.

Let $\hat{n}_k = 1 + 3\lceil \frac{n_k-1}{3} \rceil$. If $\hat{n}_k > \lceil \frac{1}{3}(\sum_{i=1}^k \hat{n}_i) \rceil$, branch k will be always active, and at most one more packet to send after branch k finishes. Therefore maximum is $\hat{n}_k + 1 \leq n_k + 3$.

If $\hat{n}_k \leq \lceil \frac{1}{3}(\sum_{i=1}^k \hat{n}_i) \rceil$, there are three active branches at every three time slots, therefore the maximum is

$$\lceil \frac{1}{3}(\sum_{i=1}^k \hat{n}_i) \rceil \leq \lceil \frac{N + 2k}{3} \rceil$$

Here we see that when the number of branches k is small relative to the number of nodes N , aggregation is effective.

For tree networks, the scheduling algorithm with aggregation is similar to the one described in Section III-C with a modification of the counter W_v . Let v be any node in the tree and let W_v be the number of time slots that have to be passed by the corresponding subtree before node v becomes active. Let v have k child nodes v_1, v_2, \dots, v_k . Let n_1, n_2, \dots, n_k respectively be the number of time slots for each subtree rooted at the child nodes of v forwarding all its packets. Then $W_{v_1} = W_v$; i.e., v_1 (the first child node) becomes active along with v . And $\forall i \leq k$ $W_{v_i} = \sum_{j=1}^{i-1} 3\lceil \frac{n_j}{3} \rceil + W_v$.

Theorem 3: Let N be the number of nodes and L be the number of leaf nodes in the network. Let n_i be the number of nodes and l_i be the number of leaf nodes in the i^{th} one-hop-subtree. The maximum number of time slots required for aggregation-enabled convergecast in tree networks is $\max(\hat{n}, \lceil \frac{N+2L+2(L-k)}{3} \rceil)$ where k is the number of one-hop-subtrees and $\hat{n} = \max_i(n_i + 4l_i - 2)$.

Proof: Note that our convergecast scheduling algorithm reduces the tree network into equivalent linear branches. Also, the number of equivalent linear branches in a tree is equal to the number of leaf nodes in that tree. From theorem 1, we know that at most $n+2$ time slots are required for convergecast in a linear network with n nodes. Thus in a one-hop-subtree, say i , we need at most $n_i + 2l_i$ time slots. Whenever we switch scheduling from one branch to another it should be ensured that nodes which are common to both the branches are in their initial states. As a result, at most two additional time slots might be required for each switch. Moreover, we have $l_i - 1$ branch switches in the i^{th} one-hop-subtree. Hence, we need $n_i + 2l_i + 2(l_i - 1)$ time slots to schedule the transmissions in the i^{th} one-hop-subtree. The result follows from Theorem 2. ■

We see that when the number of leaf nodes are large, the aggregation enabled convergecast scheduling might not result in reduced latency comparing to the original scheduling. However, if the connectivity is relatively sparse, such as in linear or mesh networks, the improvement is significant. Observations from our simulation experiments support this result.

V. SPARSE DATA SOURCES

The algorithm in Section III can be easily *extended* to cases where nodes generate more than one packets by adding $n - 1$ nodes for a node with $n > 1$ packets [20]. However, if many nodes do not have any packet, the scheduling would not be optimal for tree networks. Consider the example of a linear network in Fig. 3. If only node d has a packet, the optimal schedule has 6 time slots, while the convergecast schedule in Section III needs $6 \times 3 - 2 = 16$ time slots. In general, the extended scheduling algorithm requires at most $3P$ time slots where $P = \sum_{i=1}^N \max(1, p_i)$ and p_i is the number of packets in node i . In this section, we will present a scheduling algorithm that uses fewer time slots. Note that although PDASN [4], [5] also obtains optimal scheduling for the sparse data case, it computes the scheduling at the base station, while ours is distributed computation at each node, without global network connectivity information for any node.

Similar to the methods we use in Section III, we start with linear networks and then extend to multi-line and tree-networks, finally add conflict resolution to general networks.

A. Linear Networks

Given a linear network of N nodes, let node i be the node i -hops from the sink. Let each node associate with three variables: F_i, M_i, O_i , where

- F_i : first time node i transmits a packet.
- M_i : last time node i transmits a packet.
- O_i : last time node i transmits its own packet.

Let p_i be the number of packets generated by node i for transmitting to the sink. These variables can be set during initialization as follows:

$$F_1 = 1 \quad (1)$$

$$G_i = \begin{cases} F_i + (p_i - 1) & \text{if } i = 1 \\ F_i + 2(p_i - 1) & \text{if } i = 2 \\ F_i + 3(p_i - 1) & \text{if } i > 2 \end{cases} \quad (2)$$

$$F_{i+1} = \begin{cases} \max(1, G_i + 1) & \text{if } i = 1 \\ \max(1, G_i + 2) & \text{if } i > 1 \end{cases} \quad (3)$$

$$O_i = \begin{cases} 0 & \text{if } p_i = 0 \\ G_i & \text{if } p_i > 0 \end{cases} \quad (4)$$

$$M_N = \begin{cases} 0 & \text{if } p_N = 0 \\ G_N & \text{if } p_N > 0 \end{cases} \quad (5)$$

$$M_{i-1} = \begin{cases} O_{i-1} & \text{if } M_i = 0 \\ M_i + 1 & \text{if } M_i > 0 \end{cases} \quad (6)$$

Eq. (1) means that the first node should send a packet at time slot 1, if it has any. Eq. (2) indicates the time that the node finishes sending its own packets, where the node with hopcount 1 sends at every time slot, the node with hopcount 2 sends every other time slot, and the rest nodes send at every three time slot, after sending its first packet. Note that G_i may be negative if $p_i = 0$. Eq. (3) shows that if a node with hopcount i finishes sending its own packet at time G_i , when should the node with hopcount $i + 1$ start sending its first packet? Eq. (4) gives the ending time for sending its own

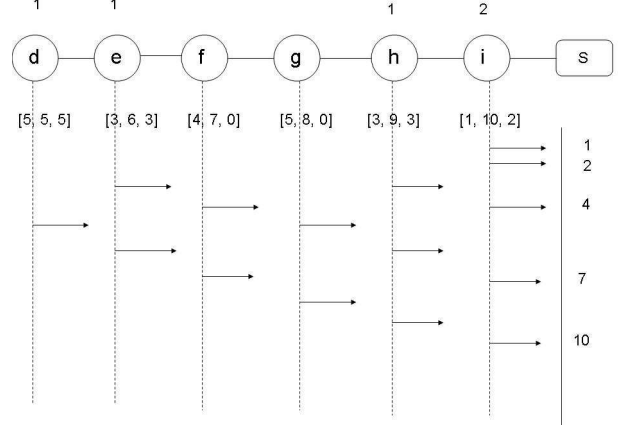


Fig. 4. Scheduling for a linear network: $[F_i, M_i, O_i]$ are displayed below each node.

packets. Note that $O_i = 0$ if $p_i = 0$, otherwise it is G_i . Eq. (5) says the last time the last node transmits is the time it finishes sending its own packet, since the last node has no packet to forward. Eq. (6) computes the last time the node with hopcount i transmits according to the last time the node with hopcount $i + 1$ transmits. Here we notice that every node just forward a packet the next time slot after it receives.

To obtain these parameters distributedly, G_i is computed at each node and passed down the links to the end and M_i is computed at each node and passed up to the first node. The total number of time slots is M_1 , i.e., the last time node 1 transmits to the sink. The initialization cost is $2N$ for N nodes.

During convergecast, similar to the scheduling in Section III, each node keeps a time slot t , which increases by 1 at each step. Node i is *active* if $F_i \leq t \leq M_i$. During the time that node i is active, it is *sending* if $F_i \leq t \leq O_i$ and it is *passing* if $\max(F_i, O_i) < t \leq M_i$. Unlike the scheduling in Section III, nodes start transmit at F_i . During *sending*, node 1 transmits every time slot, node 2 transmits every other time slot, and node $i, i > 2$, transmits every three time slots. During *passing*, every node transmit the packet received from the last time slot.

Consider a linear network shown in Fig. 4. The total number of time slots using this scheduling algorithm is 10. Five packets will arrive in time slots 1, 2, 4, 7, and 10. If using the extended scheduling algorithm in Section III, the total number of time slots is $3 \times 7 - 2 = 19$.

Theorem 4: This scheduling algorithm is correct, i.e., all packets shall arrive to the sink at $M_1 + 1$, where M_1 is the last time slot node 1 transmits.

Proof: All we need to show is that every transmission is successful. We do this by induction. When there is only one node, it is trivially true. When there are two nodes, if node 1 has no packet, it is trivially true; if node 1 has its own packets, node 2 starts transmitting the first packet after one time slot node 1 finishes its own packet, then node 1 passes

the packet, node 2 transmits its second packet, and so on, no collision will occur. In cases of more than two nodes, assume that up to node i no collision occurs. If node i has its own packets, node $i + 1$ start transmitting its first packet two time slots after node i finishes its own packet. The last packet sent by node i is now at node $i - 2$. Simultaneous transmissions of node $i - 2$ and node $i + 1$ will not cause collision. If node i has no packet of its own, node $i + 1$ will start transmit at $F_{i+1} = F_i - 3 + 2 = F_i - 1$, so that node i will pass the packet at F_i , according to the assumption, no collision occurs either. ■

This algorithm is optimal in terms of memory, i.e., at any time, the number of packets at each node is no more than its original number. Also the following theorem shows that it is time-optimal as well.

Theorem 5: The total number of time slots is

$$\max_{1 \leq i \leq N} (i - 1 + p_i + 2p_{i+1} + 3 \sum_{j \geq i+2} p_j) \quad (7)$$

and it is optimal.

Proof: The total number of time slots is M_1 . We use induction to prove that $M_1 = \max_{1 \leq i \leq N} (i - 1 + p_i + 2p_{i+1} + 3 \sum_{j \geq i+2} p_j)$. For one and two nodes, it is trivial to see it is true. Assuming for n nodes it is true. Let's consider the cases for $n + 1$ nodes. If node $n + 1$ has no packet, it is trivially true. If node $n + 1$ has $p_{n+1} > 0$ packets. Let $i \leq n$ be the last node $p_i > 0$. From Eq. 2-6, $M_1^n = M_i + (i - 1)$, $F_{i+1} = M_i + 2$, $F_{n+1} = \max(F_{i+1} - (n - i), 1)$, $M_{n+1} = F_{n+1} + 3(p_{n+1} - 1)$ and $M_1^{n+1} = M_{n+1} + n$. If $F_{n+1} = F_{i+1} - (n - i)$, $M_1^{n+1} = M_1^n + 3p_{n+1}$; otherwise, it is easy to see that both $p_{n-1} = 0$ and $p_n = 0$, and $M_1^{n+1} = n + 1 + 3(p_{n+1} - 1) = (n - 1) - 1 + p_{n-1} + 2p_n + 3p_{n+1}$.

The proof of optimal has been derived in [4]. ■

This algorithm can be easily extended to cases when some interference links are larger than transmission links. For example, in Fig. 4, if node g can hear from node i (but cannot transmit to i), i and f cannot transmit at the same time, since g may not receive from f due to interference from i . In general, we let $F_{i+1} = \max(1, G_i + d)$ where d is the minimum number that node i and node $i - d$ has no interference. For asymmetric links, we will use only the symmetric links for scheduling [24] and use the asymmetric links for interference checking.

B. Multi-line Networks

For the scheduling algorithm for multi-line networks in Section III, each node has to know the number of packets in each branch to do scheduling independently. However, if there are nodes with no data packets at all, scheduling depending on the total number of packets will not achieve optimality. A branch may have few packets but takes longer to finish due to longer distances from sources to the sink. Furthermore, the packet arrival time from each branch is no longer with the fixed interval (i.e. every three time slots). For example, in Fig. 4, packets arrive at times 1,2,4,7,and 10. The packet arrival times can be obtained using F_i . For k 'th packet of node i , the arrival

time is $F_i + 3(k - 1) + i - 1$ if $i \geq 2$ and $F_i + 2k - 1$ if $i = 2$. Each branch passes the arrival times to the sink node and then the sink node distribute them to every node. The total cost of initialization is $3N + k$ for N nodes and k branches. After initialization, each node, in addition to F, M, O values, knows the arrays of arrival times of all the branches.

The convergecast scheduling is similar to that in Section III-B, except the following:

- Let a_i be the array of arrival times for branch i . The initial last assigned time slot L_i^0 is set to be $a_i(1) - 1$ where $a_i(1)$ is the first packet arrival time for branch i , and S_i^0 is set to be the last element of a_i , i.e., the total number of time slots.
- Let j be the branch to be selected to transmit at time slot t , $L_j \leftarrow L_j + (a_j(2) - a_j(1) - 1)$, $S_i \leftarrow S_i^0 - a_j(1)$ and $a_j(k) = a_j(k + 1)$.
- If this branch is active, the node checks if it currently belongs to *sending* or *passing* state using the F, M, O parameters of this node. If it is sending, its transmit state changes according to the number of hops to the sink, i.e., 1 hop: every time slot, 2 hops: every other time slot, and more than 2 hops: every three time slots. If it is passing, it always transmits the packet received in the last time slot.

C. Tree Networks

The scheduling for tree networks is the same for multi-line networks, by reducing each one-hop-subtree to a branch. The process of reducing is the same as Section III-C, except the computation of F and M parameters for each node in a one-hop-subtree. Let v be a node in a one-hop-subtree. Assume v has k children, v_1, v_2, \dots, v_k , and assume \bar{M}_{v_i} be the last active time slot for node v_i if $F_{v_i} = 1$, then

$$\begin{aligned} F_{v_1} &= G_v + d \\ M_{v_i} &= \begin{cases} F_{v_i} - 1 + \bar{M}_{v_i} & \text{if } \bar{M}_{v_i} > 0 \\ 0 & \text{otherwise} \end{cases} \\ F_{v_{i+1}} &= \begin{cases} M_{v_j} + 1 + d & \text{if } j = \max_{l \leq i} \{l | M_{v_l} > 0\} \\ G_v + d & \text{otherwise} \end{cases} \\ M_v &= \max(O_v, \max_i M_{v_i} + 1) \end{aligned}$$

where $d = 1$ if v is the root of a one-hop-subtree and $d = 2$ otherwise. Each node obtains $[F, M, O]$ values during the initialization by traversing the tree. For each one-hop-subtree, the traverse of a tree costs $2n$ (each node is visited twice) for n nodes in the subtree. The total cost is again $3N + k$ for N nodes with k one-hop-subtrees.

Consider a tree network shown in Fig. 5. The branch rooted at b has arrival times: 1,2,4 and 7. The branch rooted at c has arrival times 1,3 and 6. The total number of time slots using this scheduling algorithm is 8, slots 1,2,4 and 7 for branch b , slots 3,5,and 8 for branch c . If using the scheduling algorithm in Section III, the total number of time slots is $3 \times 7 - 2 = 19$.

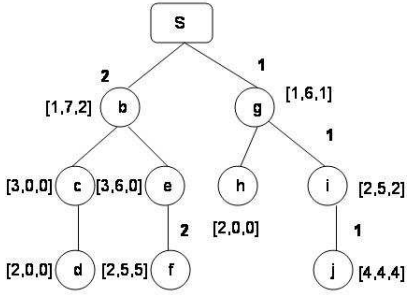


Fig. 5. Scheduling for a tree network: $[F_i, M_i, O_i]$ are displayed near each node.

D. General Networks

Similar to the scheduling in Section III-D, a BFS spanning tree is built at first and the tree scheduling algorithm is applied to the tree. In addition, a filtering mechanism, *conflict resolution* (CR), can be added so that no two branches that have potential interference may be active at the same time.

VI. PERFORMANCE EVALUATIONS

To evaluate the performance of these convergecast scheduling algorithms we conducted several simulation experiments using Rmase and Prowler [26].

The default deterministic radio model in Prowler [21] is used in our simulation. All the communication links in the network are symmetric and deterministic. Packets are lost only when there is a collision. The data rate of wireless links is set to be 40Kbps. The maximum length of a packet is fixed as 960 bits. All algorithms except our convergecast scheduling algorithm use the default CSMA MAC implemented in Prowler. Our convergecast scheduling algorithm uses CSMA during initialization. Nodes switch to TDMA after the initialization. The duration of each time slot is set to 1/40 seconds, which is slightly greater than the time required to transmit one packet of size 960 bits.

We considered a square sensor field of $n \times n$ nodes. Nodes were placed on points of a uniform grid. The connectivity of nodes is a mesh, i.e., each node has a maximum four neighbors. Note that the algorithm works for arbitrary connectivity, however, the packet aggregation performance degrades for the high density networks (Theorem 3).

A. Performance Metrics

We use the following metrics to evaluate the performance of our convergecast scheduling algorithm:

Latency: Latency is a measure of the time taken by a packet to reach the sink from its source node. We report the average latency for convergecast given by $\sum_i d_i/n$. Here, n represents the number of packets received by the sink and d_i represents the latency of the i^{th} packet.

Throughput: Throughput of a network measures the number of data packets received at the sink per second. Note that

the concept of throughput is identical to good-put since only desired data packets are counted.

Success rate: The success rate measures the fraction of packets successfully received by the sink. It is defined as a ratio between total number of data packets received by the sink and the total number of data packets generated.

Each simulation experiment was repeated for ten times, both average and standard deviation are plotted.

B. Small Data

Assume that each data frame can pack at least three data packets. We tested four variants of our convergecast scheduling algorithm as listed below. These variants are distinguished based on whether nodes employ packet aggregation and conflict resolution between different subtrees.

- 1) **wo. Agg/CR:** Nodes employ neither packet aggregation nor conflict resolution.
- 2) **wt. Agg:** Nodes employ packet aggregation. However, possible collisions between nodes of different subtrees (i.e., conflict resolution) are ignored.
- 3) **wt. CR:** Nodes do not aggregate packets. However, collisions between different subtrees are eliminated. i.e., conflict resolution is employed.
- 4) **wt. Agg/CR:** Nodes employ both packet aggregation and conflict resolution.

Note that in the first two variants some packets might be lost due to collisions. We also implemented the following convergecast algorithms to compare with the performance of our convergecast scheduling algorithms:

- **Directed Flooding:** Directed flooding [14], [22] uses *hop-count* to flood the packets towards the sink.
- **Radial Coordination:** In radial coordination for convergecast [12] each node waits for an additional time based on the *hop-count* before transmitting.

Both directed flooding and radial coordination employ packet aggregation. At most four packets can be aggregated.

We first report the number of time slots required by all the four variants of our algorithm. Then, we present the results comparing our convergecast scheduling algorithm with directed flooding and radial coordination convergecast.

1) *Number of Time Slots:* The number time slots used to complete convergecast by all the variants of our scheduling algorithm can be seen in Fig. 6. It can be noticed that the number of time slots required grows linearly with the number of nodes. The variants without conflict resolution require 50% fewer time slots comparatively. However, not all packets are guaranteed to arrive at the sink without employing conflict resolution. The variants with aggregation requires significantly fewer time slots in either cases: with or without CR.

2) *Performance Comparisons:* Fig. 7 compares the performance of all the six convergecast mechanisms under consideration. In these simulations every node generated exactly one data packet at the beginning. It can be noticed that the four scheduled convergecast variants outperform both directed flooding and radial coordination on all the three metrics.

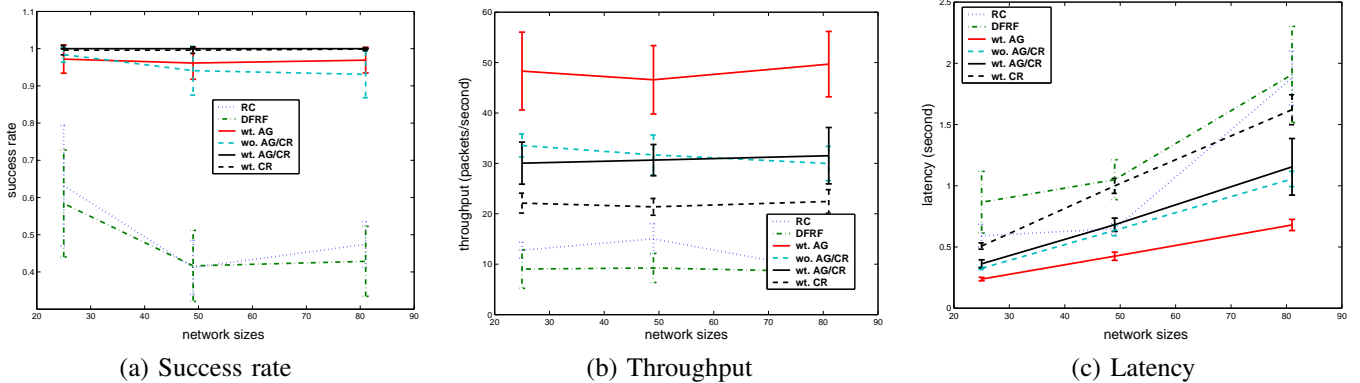


Fig. 7. Performance Comparisons: Networks with Varying Number of Nodes

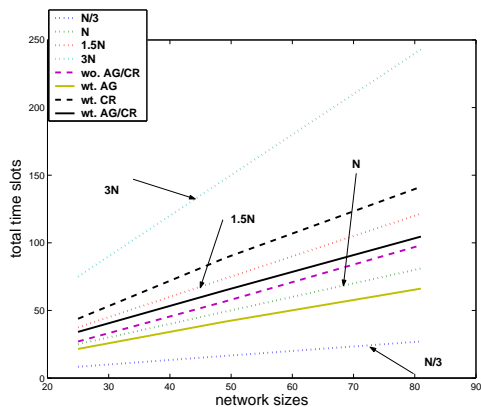


Fig. 6. Number of Time Slots Used by scheduled Convergecast Variants

When conflict resolution was employed, almost 100% of the packets were received by the sink (see Fig. 7(a)). Note that when an aggregated packet was received it was counted as multiple constituent packets. An important observation to be noticed is that *even without conflict resolution the success rate is significantly higher* (about 85%). We offer the following explanation: The nodes that interfere and belong two different one-hop-subtrees might not be active simultaneously (see Section III-D). In addition, not all nodes of interfering one-hop-subtrees interfere with each other. As a result, not all overlapping transmissions along the interfering subtrees result in collisions.

From Fig. 7(b), it can be observed that scheduling with the packet aggregation but without conflict resolution have the highest throughput and lowest latency. Based on the application requirements, one can trade-off success rate for higher throughput by not implementing the conflict resolution mechanism. The convergecast scheduling algorithm results in low latencies in comparison with directed flooding and radial coordination (see Fig. 7(c)).

C. Sparse Data

In this experiment, we compare the new scheduling algorithm using packet counts (PC) for large but sparse data

sources with the algorithm designed for each node has one data packet, with or without CR. We tested two cases: (1) percent of nodes having one data packet is varied from 20 to 80, and (2) given the percent of nodes having data to be 20, the number of packets in sources varies from 1 to 7. Fig. 8 shows throughput and latency of case (1). Algorithms using packet counts (PC) have high throughput and lower latency, the difference is larger when the percentage is smaller. Fig. 9 shows throughput and latency of case (2). Again algorithms using packet counts (PC) have high throughput and lower latency. Note that the throughput is almost constant with any number of packets, but the latency increases with the number of packets.

VII. CONCLUSIONS

We proposed two variations of minimal time distributed convergecast scheduling algorithm for sensor networks, one for convergecast of small data where packets can be aggregated on their way to the sink and the other for situations where there is large amount of data but only sparsely distributed in space. Simulation results show that the new variations improve performance significantly.

The advantages of this type of scheduling are that it produces optimal or near-optimal time for data collection and guarantees to be collision-free. The disadvantages are that it requires knowledge of the demand distribution in initialization in order to produce optimal-time scheduling, it has the high cost of initialization and is not adaptive to permanent node failures during convergecast. How to repair a distributed schedule efficiently in permanent node failures is an interesting future research. Simulations on more realistic radio models and/or more general interference models will also be conducted.

REFERENCES

- [1] Tanenbaum A. S. Computer Networks, 3rd Edition. *Prentice-Hall Inc.*, 1996.
- [2] B. Hohlt and E. Brewer. Network Power Scheduling for TinyOS Application. In *Second IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2006.
- [3] B. Hohlt and L. Doherty and E. Brewer. Flexible Power Scheduling for Sensor Networks. In *Second International Conference Information Processing in Sensor Networks (IPSN04)*, 2004.

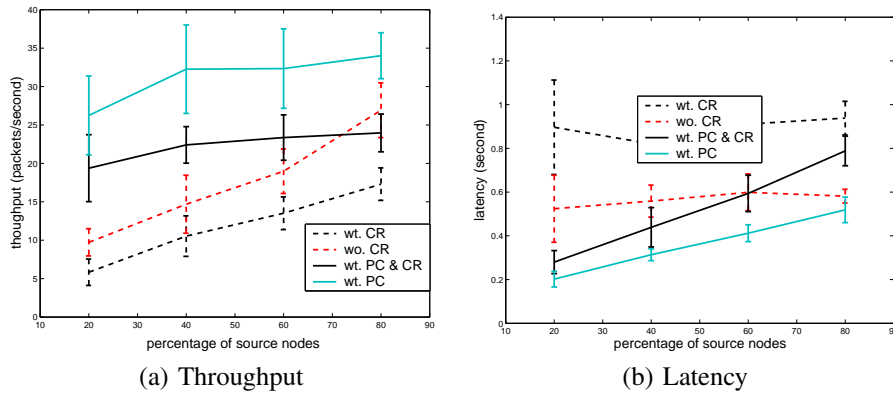


Fig. 8. Performance Comparisons: Networks with Varying Percent of Nodes Having one Data Packet.

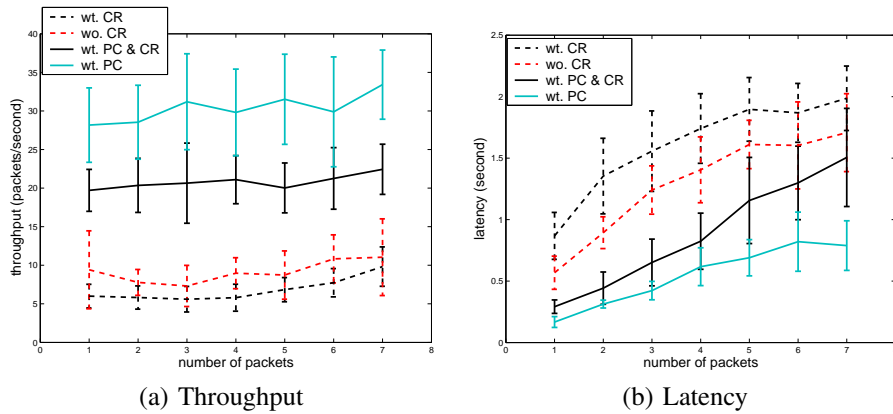


Fig. 9. Performance Comparisons: Networks with Varying Number of Packets at Each Source.

- [4] Florens C. and McEliece R. Packet Distribution Algorithms for Sensor Networks. In *IEEE INFOCOM*, 2003.
- [5] C. Florens and M. Franceschetti and R. McEliece. Lower Bounds on Data Collection Time in Sensory Networks. *IEEE Journal on Selected Areas in Communications*, 22(6), August 2004.
- [6] Bianchi G. Performance Analysis of the IEEE 802.11 Distributed Coordination Function. *IEEE JSAC*, 18(3), March 2000.
- [7] Gandham S., Dawande M. and Prakash R. Link Scheduling in Sensor Networks: Distributed Edge Coloring Revisited. *Proceedings of IEEE INFOCOM*, March 2005.
- [8] H. Zhang and A. Arora and Y. Choi and M. Gouda. Reliable Bursty Convergecast in Wireless Sensor Networks. In *ACM MobiHoc 2005*, 2005.
- [9] Hajek B. and Sasaki G. Link Scheduling in Polynomial Time. *IEEE Transactions on Information Theory*, 34:910–917, Sept. 1988.
- [10] T. He, J.A. Stankovic, C. Lu, and T.F. Abdelzaher. Speed: A stateless protocol for real-time communication in sensor networks. In *Proceedings of the International Conference on Distributed Computing Systems*, May 2003.
- [11] Ju Wang Hongsik Choi and Esther A. Hughes. Scheduling on Sensor Hybrid Network. In *IEEE ICCCN*, 2005.
- [12] Q. Huang and Y. Zhang. Radial Coordination for Convergecast in Wireless Sensor Networks. In *First IEEE Workshop on Embedded Networked Sensors*, 2004.
- [13] Crossbow Technologies Inc. www.xbow.com.
- [14] M. Maroti. Directed flood-routing framework. In *5th International Middleware Conference*, 2004.
- [15] Goodrich M.T. and Tamassia R. *Data Structures and Algorithms in Java*. John Wiley and Sons, Inc., second edition, 2001.
- [16] R. Kannan and S. Wei. Approximation Algorithms for Power-Aware Scheduling of Wireless Sensor Networks. In *Second IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2006.
- [17] Ramanathan S. A Unified Framework and Algorithm for Channel Assignment in Wireless Networks. *Wireless Networks*, 5:81–94, 1999.
- [18] Ramaswami R. and Parhi K.K. Distributed Scheduling of Broadcasts in a Radio Network. *IEEE INFOCOM*, 1989.
- [19] S. Gandham and M. Dawande and R. Prakash and S. Venkatesan. Energy Efficient Schemes for Wireless Sensor Networks with Multiple Mobile Base Stations. *IEEE Globecom*, December, 2003.
- [20] S. Gandham and Y. Zhang and Q. Huang. Distributed Minimal Time Convergecast Scheduling in Wireless Sensor Networks. In *IEEE ICDCS06*, 2006.
- [21] G. Simon. Probabilistic Wireless Network Simulator. <http://www.isis.vanderbilt.edu/projects/nest/prowler/>.
- [22] Balogh G. Kusy B. Ndas A. Pap G. Sallai J. Simon G., Ldeczi A. and Frampton K. Sensor Network-Based Countersniper System. In *Sensys*, 2004.
- [23] Jack Stankovic, T. Abdelzaher, Chenyang Lu, Lui Sha, and J. Hou. Real-time communication and coordination in embedded sensor networks. *Proceedings of the IEEE*, 91(7):1002–1022, July 2003.
- [24] T. He and S. Krishnamurthy and L. Luo and T. Yan and R. Stoleru and G. Zhou and Q. Cao and P. Vicaire and J.A. Stankovic and T.F. Abdelzaher and J. Hui and B. Krogh. VigilNet: An Integrated Sensor Network System for Energy-Efficient Surveillance. *ACM Transactions on Sensor Networks*, Feburay 2006.
- [25] W. Song. Time-Optimum Packet Scheduling for Many-to-One Routing in Wireless Sensor Networks. In *Third IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS06)*, 2006.
- [26] Y. Zhang and G. Simon and G. Balogh. High-Level Sensor Network Simulations for Routing Performance Evaluations. In *Third International Conference on Networked Sensing Systems (INSS06)*, 2006.