Link Scheduling in Sensor Networks: Distributed Edge Coloring Revisited

Shashidhar Gandham
Department of Computer Science
University of Texas at Dallas
Email: gshashi@utdallas.edu

Milind Dawande
School of Management
University of Texas at Dallas
Email: milind@utdallas.edu

Ravi Prakash
Department of Computer Science
University of Texas at Dallas
Email: ravip@utdallas.edu

***Abstract*— We consider the problem of link scheduling in a sensor network employing a TDMA MAC protocol. Our link scheduling algorithm involves two phases. In the first phase, we assign a color to each edge in the network such that no two edges incident on the same node are assigned the same color. We propose a distributed edge coloring algorithm that needs at most $(\delta+1)$ colors, where δ is the maximum degree of the graph. To the best of our knowledge, this is the first distributed algorithm that can edge color a graph with at most $(\delta+1)$ colors. In the second phase, we map each color to a unique timeslot and attempt to identify a direction of transmission along each edge such that the hidden terminal and the exposed terminal problems are avoided. Next, considering topologies for which a feasible solution does not exist, we obtain a direction of transmission for each edge using additional timeslots, if necessary. Finally, we show that reversing the direction of transmission along every edge leads to another feasible direction of transmission. Using both the transmission assignments we obtain a TDMA MAC schedule which enables two-way communication between every pair of neighbors. For acyclic topologies, we show that at most $2(\delta+1)$ timeslots are required. Through simulations we show that for sparse graphs with cycles the number of timeslots assigned is close to $2(\delta+1)$.**

Keywords: Link Scheduling, Distributed Edge Coloring, Wireless Sensor Networks.

I. Introduction

We consider a sensor network consisting of a large number of static sensor nodes deployed on-the-fly for unattended operation [8], [10]. These networks are employed for critical applications like surveillance and remote monitoring [19]. Each sensor node is expected to independently monitor its surrounding environment and detect occurrences of events of interest. On detecting an event, a node attempts to quickly report it to a base station associated with the sensor network. As each node is powered by limited battery-supplied energy it may not be feasible to transmit the data directly to a base station. Thus, sensor nodes are required to form a multi-hop wireless network to communicate with a base station in a timely and energy-efficient fashion. A Medium Access Control (MAC) protocol is needed to forward data along each link of such a multi-hop wireless network.

In general, contention-based MAC protocols like IEEE 802.11 [16] may require multiple retransmissions of packets due to collisions. As each retransmission drains energy from the nodes, such protocols are not suitable for energy-constrained sensor nodes. Moreover, such schemes may result in non-deterministic per hop delay.

Time Division Multiple Access (TDMA) MAC protocols (see, e.g. [5]) eliminate collisions, guarantee fairness and provide bounds on per-hop latency. To conserve energy, a node employing a TDMA MAC protocol can switch off its transceiver when it is neither transmitting nor receiving. We therefore believe that TDMA is a suitable MAC protocol for sensor networks. In a TDMA MAC protocol, time is split into equal intervals referred to as *timeframes*. Each timeframe is further divided into slots of equal length referred to as *timeslots*. For facilitating two-way communication, at least two timeslots must be allocated for every pair of neighboring nodes.

The main challenge in adopting a TDMA MAC protocol for sensor networks is in allocating timeslots for each pair of neighboring nodes. This problem is referred to as *link scheduling* [6], [21]. To avoid collisions, a feasible timeslot allocation should be such that exactly one of the neighbors of a receiving node should transmit in a timeslot; the other neighbors might receive during the timeslot. A naive approach for a feasible timeslot allocation is to assign a unique timeslot to each pair of neighbors in the network. Although this eliminates the possibility of collisions, the large number of timeslots required increases the latency of communication. *We therefore require a feasible timeslot allocation that uses a minimal number of timeslots.*

A. Timeslot Assignment as an Edge Coloring Problem:

The timeslot assignment problem is closely related to the edge coloring problem for a graph. In a **valid edge coloring**, *no two edges incident on the same node are assigned the same color*. Vizing's theorem [7] states that a valid edge coloring for a graph can be obtained by using at most $(\delta+1)$ colors, where δ is the maximum degree of a node in the graph. Given a valid edge coloring of the graph that represents the sensor network, a timeslot assignment for the sensor nodes can be obtained by mapping each timeslot to a color. However, such a TDMA schedule suffers from the *hidden terminal problem* [24]. To illustrate this, consider the topology shown in Figure 1(a). Here, edges (1, 4) and (2, 3) are assigned color 1 and edges (1, 2) and (3, 4) are assigned color 2. Let the directions of

transmission in timeslot 1, corresponding to color 1, be as shown in Figure 1(a). The reception at nodes 2 and 4 will be garbled due to collision of transmissions from nodes 1 and 3, respectively. As a result, both the receivers (i.e., nodes 2 and 4) will be unable to receive properly in this timeslot.

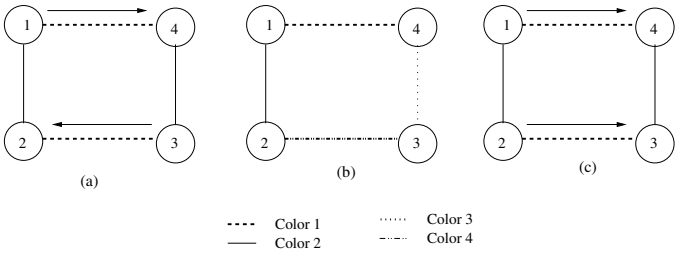


Fig. 1. TDMA Schedule Based on Edge Coloring.

To overcome the hidden terminal problem, a possible alternative is to assign colors to edges such that no two edges incident on a node or its neighbors have the same color. But, doing so results in the *exposed terminal problem* [24]. The result of such a coloring in our example graph is shown in Figure 1(b). In this case, the number of colors required is four and the number of timeslots required to support duplex communication is eight. Alternatively, if we can assign the direction of transmission along the edges as shown in Figure 1(c), we can facilitate transmission from node 1 to node 4, and from node 2 to node 3. Observe that by reversing the direction of transmission along each edge, we can facilitate transmission from node 4 to node 1, and from node 3 to node 2. Here, we need four timeslots to support duplex communication. The basic idea behind our timeslot assignment algorithm is to first obtain a valid edge coloring of the graph and then determine directions of transmission along each edge so that the hidden terminal problem is avoided. If no such assignment exists, then we identify the edges that need to be assigned a different timeslot (i.e., a color) to obtain a feasible assignment. Such an assignment enables one-way communication between every pair of nodes. We show that duplex communication can be facilitated between every pair of nodes by reversing the direction of transmission along each edge.

In this paper, we present a distributed, edge coloring-based link scheduling algorithm for sensor networks. Our algorithm works in two phases. The first phase yields a valid coloring of the edges of the network in a distributed fashion. Our distributed edge coloring algorithm is based on a centralized constructive proof of Vizing's theorem due to Misra and Gries [14]. In the second phase, for every node and for every incident color on that node, a sign (+ or -) is associated such that collision-free transmission can take place along the edges. A node assigned a + (resp. -) sign can transmit (resp. receive) during the corresponding timeslot. The important contributions of our work are:

- A distributed edge-coloring algorithm using at most $\delta + 1$ colors. To our knowledge, this is the only known distributed algorithm to obtain a valid color assignment using at most $\delta + 1$ colors.
- A timeslot assignment for acyclic topologies using at most $2(\delta + 1)$ colors.

II. Related Work

Panconesi and Srinivasan [3] proposed a distributed edge-coloring algorithm that uses at most $(2\delta - 1)$ colors. Later, they presented a better distributed algorithm [4] that requires at most $1.6\delta + O(\log^{1+\delta} n)$ colors. The previous best known distributed edge coloring algorithm is a randomized algorithm due to Grable and Panconesi [9] and uses at most $\delta(\epsilon + 1)$ colors, where $\epsilon > 0$ is an input parameter. It is known that this algorithm might fail to assign a valid color to every edge in the graph. Higher the value of ϵ greater is the probability to color every edge in the graph. Marathe *et al.* [1] conducted an experimental study of this algorithm and listed the scenarios where it fails. We present a *deterministic, distributed* edge coloring algorithm that results in a valid edge coloring of the graph using at most $\delta + 1$ colors. Next, we look at timeslot assignment algorithms available in the literature.

Timeslot assignment to nodes is referred to as broadcast scheduling [2], [20], [23]. We focus on timeslot assignment to edges rather than to nodes due to the following reasons:

(1) *Concurrency of transmissions:* If a timeslot is assigned to a node then none of the two-hop neighbors of the node can be assigned the same slot [23] in a collision-free assignment. As shown in Figure 1(c), two-hop neighbors can transmit or receive concurrently during a timeslot if the slot assignment is made to the edges. Thus, assigning timeslots to edges increases the number of concurrent transmissions.

(2) *Bandwidth share proportional to potential requirements:* A slot assignment to nodes restricts each node to transmit in at most one timeslot in each timeframe, irrespective of the number of neighbors a node might have. When slots are assigned to edges, each node has one timeslot per neighbor in each timeframe. Hence, the bandwidth share available to a node is proportional to its number of neighbors.

(3) *Energy Conservation:* When timeslots are assigned to nodes, each neighbor of the transmitting node has to switch on its transceiver, irrespective of whether it is the intended receiver or not. Consequently, nodes waste energy in receiving frames not intended for them. On the other hand, if each edge is assigned a slot, only the intended receiver switches on its transceiver.

The problem of assigning timeslots to edges using a minimum number of timeslots is known to be NP-complete [21]. Tamura *et al.* [11] modeled the timeslot assignment as a two-hop edge coloring problem wherein two edges are assigned different colors (timeslots) if they are adjacent or if there are connected by an edge. Figure 1(b) illustrates that such an approach induces the exposed terminal problem and is not necessarily the best. In [15], Sohrabi *et al.* proposed a timeslot allocation algorithm for sensor networks. The TDMA schedule obtained from their method requires each link to operate on a different frequency in order to avoid the hidden terminal

problem. We consider sensor networks communicating on the same frequency band. Ramnathan [21] presented a unified framework for TDMA, FDMA and CDMA based multi-hop wireless networks. This study also proposed a timeslot assignment to edges; the number of timeslots required is at most $O(\theta)$ times the optimum number, where θ is the minimum number of planar graphs into which the network can be decomposed. We show that our approach requires at most $2(\delta+1)$ timeslots for acyclic graphs. For arbitrary sparse networks (i.e., those that may contain cycles) we show, through simulation, that the number of slots needed is close to $2(\delta+1)$. Note that 2δ is a lower bound on the number of timeslots required in any feasible assignment.

III. Distributed Edge Coloring Algorithm

Misra and Gries [14] provided a constructive proof for Vizing's theorem. Given an arbitrary graph with no self-loops and no multiple edges, their algorithm provides a valid edge coloring using at most $\delta+1$ colors. For a detailed description and a proof of correctness we refer the reader to [14]. Here, we briefly describe their centralized method and provide our distributed implementation of it.

A. Misra and Gries Centralized (MGC) Algorithm

The MGC algorithm uses two data structures: a fan and a cd-path. A fan $< f...l >$ (see Figure 2(a)) of a node i is a list of its neighbors with the following properties:

- $< f...l >$ is a non-empty set of distinct neighbors of node i.
- The edge (i, f) is uncolored and
- The color of edge (i, u^s) is free at node u, where u^s represents the successor of u in the fan.

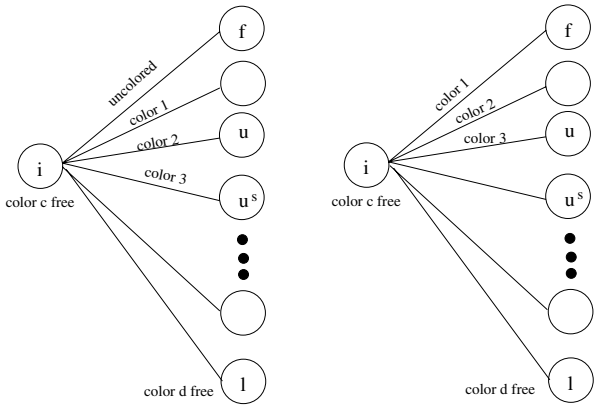
As shown in [14], a maximal fan can be constructed as follows:

1) Initialize the fan with a node f, where the edge (i, f) has to be assigned a color.
2) Find a neighbor u of i such that u is not already in the fan and color of (i, u) is free at the last node added to the fan. Add the node u to the fan.
3) Repeat step (2) until no node satisfies the condition stated in (2).

In the operation *rotate fan* (see Figure 2(b)) the color of every fan edge (i, u) is replaced by the color of the edge (i, u^s), where u^s is the successor of node u in the fan.

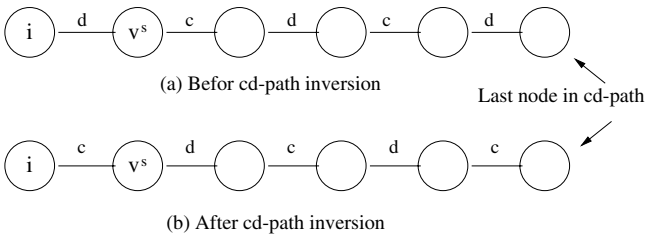
Let c and d be colors free at nodes i and l, respectively (see Figure 2), where l is the last node added to the fan. A cd-path is defined as a maximal path starting at node i with colors c and d alternating along the successive edges (see Figure 3(a)). It is shown in [14] that a cd-path is simple and unique. Switching the colors of the edges along cd-path (c to d and d to c) is termed as the operation *invert* (see Figure 3(b)). If the initial coloring of the graph is valid then the coloring of the graph after inverting a maximal cd-path remains valid [14].

As color c is free at i, the edge in the cd-path that is incident on i has to be colored d. Let this edge be (i, v^s), where v^s is part of the fan[1]. Let node v be the predecessor of node v^s in the fan.



(a) Before Rotating the Fan (b) After Rotating the Fan

Fig. 2. Rotation of Fan.



(a) Befor cd-path inversion

(b) After cd-path inversion

Fig. 3. Inversion of cd-path.

MGC algorithm colors the edge (i, f) as follows:

- Construct a maximal fan $< f...l >$.
- Find a cd-path and *invert* it.
- Identify a node $w \in < f...l >$ as follows:
  - $w = l$ if no fan edge has color d [2] OR node v (defined above) is in the cd-path.
  - $w = v$ if node v (defined above) is not in the cd-path.
- The list $< f...w >$ will be a fan. Rotate this fan.
- Assign color d to the edge (i, w).

Note that the edge (i, f) is colored as a result of fan rotation. If node v is in the cd-path, then it will be the last node in the path because by construction of a fan, color d should be free at node v. Thus, the cd-path cannot be extended beyond node v. The above procedure is sequentially repeated at each node until all the edges in the graph are assigned a color.

The MGC algorithm described above is sequential and centralized. As sensor nodes are deployed in large numbers, it may not be possible to collect the entire topology at a single node and apply the algorithm. We therefore describe a distributed

[1] Assume v^s was not part of the fan. We know that color d is free at node l, the last node in the fan. By the algorithm provided for construction of a maximal fan, v^s should be added into the fan after node l, thus contradicting the fact that l is the last node in the fan.

[2] *i.e.* cd-path is of length zero.

implementation which can be invoked concurrently by sensor nodes and requires minimal amount of local information.

B. Distributed Edge Coloring

The design goals of our distributed implementation of the MGC algorithm are: (1) to reduce the number of messages exchanged by the nodes (conserves energy at sensor nodes), and (2) allow as many nodes as possible to concurrently color the edges incident on them. We first identify the required conditions to concurrently execute the algorithm and a few strategies to reduce message complexity of our distributed implementation. We then present our distributed implementation.

Consider a sensor node i, which is independently applying the MGC algorithm to color the edges incident on it. For coloring an edge, i constructs a fan. To ensure correctness, the fan has to remain valid until the coloring of edge (i, w) (see Section III-A) is completed. The only required condition for the fan to remain valid is that the color assignment of edges incident on neighbors of i remains unchanged. Thus, when a node is coloring the edges incident on it, none of its two-hop neighbors should change the colors of edges incident on them.

Any distributed implementation of the $invert$ (Section III-A) operation would require propagation of a message along the cd-path to: (1) explore the cd-path and (2) invert the colors of edges along the cd-path. Note that while a cd-path is explored, the colors of edges along the cd-path should not change. Hence, the cd-path invert operation requires all the nodes along the path to refrain from changing colors, including nodes beyond the two-hop neighborhood of i. Thus, the cd-path exploration can potentially reduce the concurrency of the distributed edge coloring algorithm.

To minimize loss of concurrency and reduce the message complexity we propose to split the edge coloring process into two rounds. In the first round, colors are assigned to as many edges as possible without invoking the $invert$ operation. If color d is selected such that it is free at node i also (see Figure 2), then the length of the cd-path has to be zero (explained in Section III-A). This will allow us to color the edge (i, f) (see Figure 2) without invoking the $invert$ operation. In the second round, the remaining edges are colored using the cd-path invert operation.

C. Coloring in First Round

The set of available colors, with cardinality $\delta + 1$, is represented by $Available$. During the first round of coloring, a node i maintains the following sets: (1) $Allocate_i$ and (2) $Free_i$. $Allocate_i$ represents the set of colors that are assigned to edges incident on i. $Free_i$ represents the set of colors that are not assigned to any edge incident on i. Note that $Free_i = Available - Allocate_i$.

Node i attempts to color its edges only after all its two-hop neighbors, with IDs greater than i, have finished their first round of coloring. As a result, when i is coloring its edges, none of its two-hop neighbors are coloring their edges. Nodes with the highest ID in their two-hop neighborhood will be the first to perform coloring. As no edges have been colored in their two-hop neighborhood, they can directly assign unique colors from the set $Available$ to all edges incident on them. This does not require construction of a fan.

Before coloring in the first round, node i obtains the set $Allocate_k$ from each neighboring node k and performs the coloring locally. After constructing a fan, the following rule is used in selecting colors c and d:

- If ($Free_i \cap Free_l \neq$ NULL) then $d \in Free_i \cap Free_l$ and $c \in Free_i - d$.

The above rule ensures that color d is free at both i and l. As a result the cd-path will be of length zero. For each uncolored edge a fan is constructed once. If colors c and d satisfying the above rule can be found, the fan rotation operation is performed to color the uncolored edge. As a result of the fan rotation operation, the colors of the previously colored edges will change. Once node i is done with the first round of coloring, it announces the colors of its edges to all its neighbors. In addition, i informs all its two-hop neighbors that it has finished coloring in round one.

By the end of the first round of coloring, there may be some uncolored edges. In the second round, we allow invocation of the $invert$ operation on the cd-path.

D. Problems with Concurrent cd-path Inversion

A simple approach to implement the cd-path invert operation would be to forward a message along the path starting from node i with alternating edges colored d and c. Any node receiving this message would check if the cd-path can be further extended. If so, the message will be forwarded to the next node on the cd-path. If the cd-path cannot be extended, the node will include its ID in the message and send it back towards node i along the cd-path. Once node i gets this message it initiates the cd-path invert operation, followed by fan rotation (see Section III-A).

With different nodes concurrently coloring edges in different parts of the network, it is possible that cd-paths originating at two different nodes might overlap or intersect. If a c_1d_1-path and a c_2d_2-path are being concurrently explored, where c_1, c_2, d_1 and d_2 are distinct colors, they can only intersect at nodes and will have no edges in common. Therefore concurrent inversion of these two paths will not interfere with each other. However, if two such paths have at least one color in common, they may share some common edges. We refer to such paths as *overlapping paths*. If the invert operations corresponding to the overlapping paths are carried out independently there is a possibility that some edges might receive conflicting color assignments. For example, consider a c_1d-path and a c_2d-path overlapping with one common edge (k, l) which is colored d as shown in Figure 4(a). Edge (k, l) will undergo two concurrent color inversions, c_1 and c_2, in some order. Let the inversion of the color of edge (k, l) happen first with respect to the c_1d-path (see Figure 4(b)) followed by that for the c_2d-path (see Figure 4(c)). As can be seen in Figure 4(c), both nodes k and l have two edges with color d incident on them. Thus, the resulting color assignment is invalid. Hence, we need to

ensure that the invert operations on overlapping cd-paths are properly serialized.

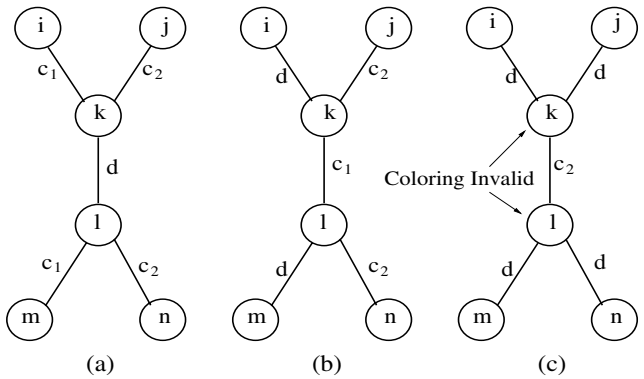


Fig. 4. Non-serialized cd-path Inversion.

A well-known mechanism for serialization between two concurrent computations that share resources is locking. In the case of cd-path inversion we need to lock the colors of edges on the cd-path. Once colors of all edges along the cd-path are successfully locked, invert operation can be performed and locks can be released. However, this locking mechanism is prone to deadlocks.

Locking the colors of an edge entails acquiring corresponding locks at the two nodes on which the edge is incident. Let (n_1, n_2) be the common edge of two overlapping cd-paths P_1 and P_2. If concurrently, P_1 acquires the lock at n_1 and P_2 acquires the lock at n_2 then neither P_1 nor P_2 can successfully lock the edge. A deadlock occurs because P_1 is waiting for P_2 to release the lock at n_2 and vice versa.

Another possible deadlock scenario is illustrated in Figure 5. Here, the paths to be inverted are P_1 and P_2 with colors c, d_2 and c, d_1, respectively. The nodes along the paths P_1 and P_2 are $x-p-q-r-s$ and $y-r-s-q-p$, respectively. At the instant shown in Figure 5, P_1 has succeeded in locking colors c and d_2 at nodes p and q and is waiting at node r for the locks. Similarly, P_2 has succeeded in locking colors c and d_1 at nodes r and s and is waiting at node q for the locks. As can be seen in Figure 5, we have a circular wait resulting in a deadlock.

E. Basic Idea Behind Second Round Coloring

In order to avoid deadlocks we introduce priorities to lock requests, and pre-emption of locks based on priorities. Locking requests to extend a cd-path are prioritized based on the ID of the cd-path initiator: *Higher the initiators ID, higher the priority of locking request*. We first describe how colors of edges along the cd-path are locked and subsequently inverted. Second, we describe how deadlocks are avoided in this locking mechanism.

We use two types of locks: pre-emptive and non-preemptive. Their meaning will become clear in the following discussion. Node i, shown in Figure 6(a) initiates cd-path discovery by acquiring a pre-emptive lock on color d corresponding to edge (i, j). Then it forwards the lock request (along with the

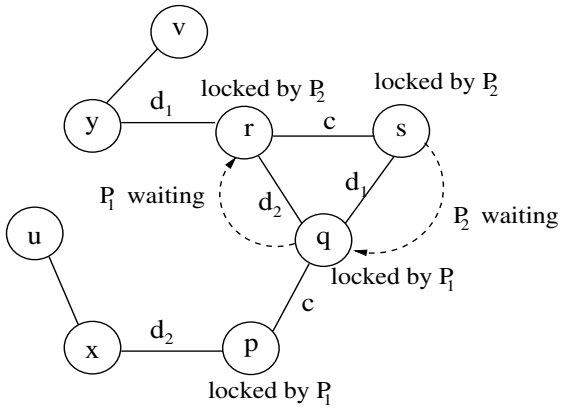


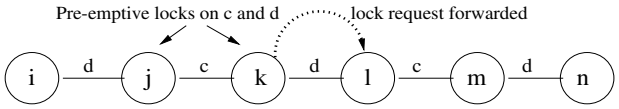
Fig. 5. Deadlock Due to Locking.

discovery message) to node j. On receiving this message each node on the cd-path, except the last one, acquires pre-emptive locks on colors c and d and forwards the lock request message. For example, node k shown in Figure 6(a), on receiving the lock request message acquires pre-emptive locks on colors c and d for the edges (j, k) and (k, l), respectively. The last node n (see Figure 6(b)) on receiving the message acquires a non-preemptive lock for color d corresponding to edge (m, n) and sends a lock success message back towards i along the same cd-path. Every node along the cd-path on receiving the lock success message converts its locks from pre-emptive to non-preemptive, and forwards the message towards i. Eventually when node i receives the lock success message, it concludes that locking of colors c and d was successful at all the nodes on the cd-path. Then, node i sends a message along the cd-path requesting intermediate nodes to invert the colors of edges on the cd-path. On receiving this message, a node will invert the colors, release the locks and forward the message to next node on the cd-path (see Figure 6(d)). Note that the color of an edge is changed only when the nodes at both end points of the edge have made a note of the new color assigned to the edge.

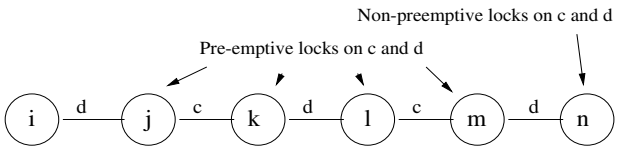
Now, let us observe how pre-emption can be used to avoid deadlocks in the above-described locking mechanism. If a node receives a lock request message corresponding to a cd_1-path and color c is locked (say by a cd_2-path), then the lock request message has to wait until the lock on color c is released under the following conditions:

- Lock on color c is non-preemptive.
- Lock on color c is pre-emptive but the priority of lock request message associated with cd_1 path is lower than that of the cd_2 path.

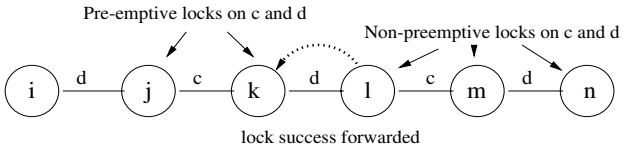
If neither of the conditions are satisfied, *i.e.* the lock on color c is pre-emptive and the incoming lock request has a higher priority, then a pre-emptive lock on color c is obtained for cd_1 path and a lock failure message is sent in both directions of the cd_2 path. A node on receiving the lock failure message would release the locks of corresponding colors and forward the message to the next node on the path. When the initiator of



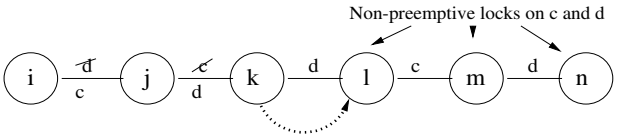
(a) cd-path Lock Request



(b) Lock Request at Last Node



(c) cd-path Lock Success



(d) cd-path Invert

Fig. 6. Deadlock Free Locking.

a cd-path receives failures message, it would retry to explore and invert the cd-path.

The lock pre-emption mechanism described above will ensure that there are no circular waits, thus avoiding deadlocks. Before performing coloring in the second round, a node will acquire non-preemptive locks on all the colors at itself and its neighbors.

IV. Sign Assignment to Nodes

The algorithm described in Section III produces a valid edge coloring of the sensor network. We now describe how this coloring can be used to assign timeslots to each edge. The idea is to map each color to a unique timeslot and assign a feasible direction of transmission for each edge such that there are no collisions. For assigning a direction of transmission to an edge in a timeslot, we decide which node amongst the two endpoints of that edge will transmit in the timeslot. The transmitting node is assigned a $+$ sign and the receiving node is assigned a $-$ sign. In this section, we describe the signing problem and our solution. To simplify the discussion, we consider a centralized scenario wherein the entire sensor network and the colors assigned to each edge in the network are known. The DFS-based sign assignment algorithm presented in Section IV-B can be easily implemented in a distributed fashion [12], [17].

A. Notation and Problem Formulation

Let *G(V, E)* denote the sensor network, where *V* is the set of sensor nodes and $E \subseteq V \times V$ is the set of wireless links. Given a valid coloring of G, define the subgraph $G^g(V^g, E^g)$ corresponding to a color, say g, as follows:

1) V^g = Set of all vertices which have color g edge incident on them.
2) E^g = Set of all edges in E which have both endpoints in V^g.

In the timeslot corresponding to color g, only the nodes in V^g either transmit or receive. For each color g, a *valid signing* of G^g is an assignment of a sign, + (transmit) or - (receive), to each node such that:

a. When a node transmits (i.e., assigned a + sign), the only neighbor that receives (i.e., assigned a - sign) is the neighbor connected by the edge of color g. The other neighbors are assigned a + sign, and can transmit. This constraint prevents the *exposed terminal problem* [24].
b. When a node receives (i.e., assigned a - sign), the only neighbor that transmits (i.e., assigned a + sign) is the neighbor connected by the edge of color g. The other neighbors are assigned a - sign, and can receive. This constraint prevents the *hidden terminal problem* [24].

The above constraints state that in a valid signing of G^g, nodes connected by an edge with color g will always have an opposite sign. Similarly, nodes connected by an edge with color other than g will have same sign. These constraints ensure that there are no collisions at the receiving nodes, and result in a set of feasible directions for transmission (in a timeslot) along each edge of color g in G^g. If there does not exist a valid signing, then we identify a set of edges with color g, which when removed from G^g results in a valid signing. The removed edges are then assigned colors other than g.

B. The DFS-based Sign Assignment Algorithm

The following algorithm, based on Depth First Search (DFS), assigns a valid signing to G^g provided such an assignment exists. In a DFS, when an edge (i, j) is traversed from a visited node i to an unvisited node j then i is referred to as parent of j and j is referred to as child of i. Note that the DFS is performed only on the derived graph G^g.

1) Start by visiting any node *r* in V^g and initiate a DFS procedure. Assign a + sign to node *r*. Follow an edge in G^g to visit a neighbor of r.
2) Let *ParentSign* be the sign assigned to the parent node (say i) of the node currently visited.
3) Set the sign of current node (say j) = *ParentSign* if the edge (i, j) does not have the color g. Otherwise, sign of current node = $flip$(*ParentSign*) if edge (i, j) has color g.

The function $flip$ returns + if the input is - and vice versa. As G^g may consist of different connected components, every

node independently initiates the DFS procedure; once visited by a DFS initiated by a higher ID node, a node will act as an unvisited node and abort the DFS procedure initiated by it. *The DFS initiated by highest ID node in each connected component will sign all the nodes in corresponding connected components*. Note that signing of nodes has to be done with respect to each color used in the edge coloring. Figure 7 shows an example of a valid signing.

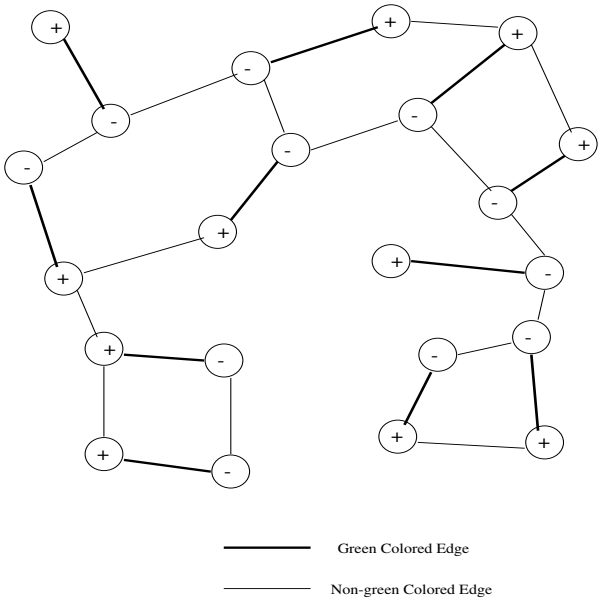


Fig. 7. Valid Signing.

C. Valid Sign Assignments in Cycles

It is easy to see that the algorithm described above obtains a valid signing for acyclic networks. However, if G^g contains cycles, a valid signing may not exist. Below, in Theorem 4.3, we prove a characterization of cycles that cannot have a valid signing. We first need the following two results.

Theorem 4.1: If a path P in G^g has an even number of edges with color g, then the first node and the last node in P have the same sign in a valid signing of P.

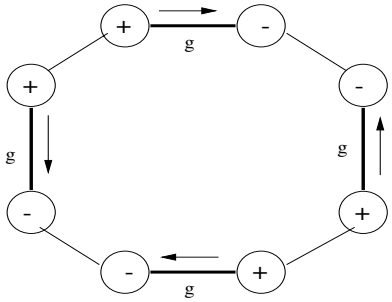
Proof: Let $P = v_1 - v_2 - v_3 - ... - v_{k-1} - v_k$. Without loss of generality, assume that in a valid signing of nodes in *P*, v_1 is assigned +. Now consider the working of the DFS-based sign assignment algorithm on path *P*. Starting at node v_1, it flips the sign to be assigned to a node if and only if it crosses an edge with color g. Thus, a node that is visited immediately after crossing two edges with color g has the same sign as v_1. Moreover, all nodes that are visited before the crossing of the third edge with color g also have the same sign as v_1. In general, for a non-negative integer q, all nodes in P that are visited after crossing $2q$ edges of color g and before crossing the $(2q+1)^{th}$ edge, have the same sign as v_1. ■

A similar argument can be used to prove the following theorem.

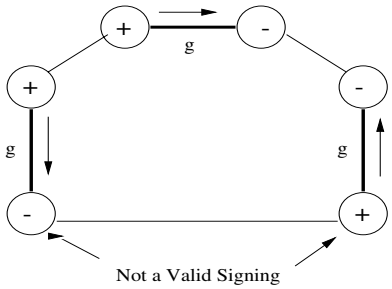
Theorem 4.2: If a path P in G^g has an odd number of edges with color g, then the first node and the last node in P have opposite signs in a valid signing of P.

Theorem 4.3: All the nodes in a cycle C in G^g can be given a valid sign if and only if there are an even number of edges with color g in C.

Proof: Let $C = v_1 - v_2 - ... - v_{k-1} - v_k - v_1$ and assume that C has an even number of edges with color g. Consider the path $P = v_1 - v_2 - ... - v_{k-1} - v_k$. If edge (v_k, v_1) has color g, then P has an odd number of edges with color g. A valid signing of P has opposite signs for nodes v_1 and v_k (Theorem 4.2) and is therefore a valid signing of C (see constraints listed in Section IV-A). Otherwise, if edge (v_k, v_1) does not have color g, then P has an even number of edges with color g. Then, a valid signing of P has the same signs for nodes v_1 and v_k (Theorem 4.1) and is therefore a valid signing of C. The converse follows from a similar argument. ■



(a) Cycle with Even Number of Edges with Color g



(b) Cycle with Odd Number of Edges with Color g

Edge with Color g
Edge with Other Color

Fig. 8. Signing of an Even Cycle and an Odd Cycle.

Figure 8(a) shows an example of a cycle with even number of edges with color g and a feasible sign assignment. Figure 8(b) shows an example of a cycle with odd number of edges with color g where a feasible signing does not exists. Note that a cycle in G^g indicates that the same cycle is present in G and every node in the cycle has an edge with color g incident on them. Next, we show that if the DFS-based signing algorithm cannot assign a feasible sign to a node then no

feasible signing exists for that node.

Theorem 4.4: If the signing algorithm cannot assign a valid sign to some node i in G^g, then i belongs to a cycle C which has odd number of edges with color g.

Proof: Consider only the DFS initiated by the highest ID node in a connected component of G^g [3]. Note that the signing algorithm cannot assign a valid sign to a node, say i, in G^g if and only if

1) *i* has already been assigned a sign in a previous visit during DFS, and
2) the sign of *i* is different from the sign that would have been assigned to *i* in the current visit.

Since G^g is undirected, the algorithm traverses a back edge to revisit node *i*. Node i is therefore part of a cycle, say C. The result now follows from Theorem 4.3. ∎

Node i, on noticing that it cannot have a valid sign with respect to color g, will assign a different color to the edge with color g incident on it. For assigning a color to this edge, i may use more than $(\delta + 1)$ colors in a greedy fashion.

A valid sign assignment for the nodes in G^g facilitates a *one directional* communication between pairs of nodes in V^g that are connected by an edge with color g. The following result stated without proof follows directly from the definition of a valid signing and yields a signing for communication between the same pair of nodes in the reverse direction.

Theorem 4.5: Given a valid sign assignment of a graph $G^g(V^g, E^g)$, we can obtain another valid sign assignment by reversing the sign of each node.

We know that two timeslots are to be assigned to each edge to support two-way communication between neighboring nodes. A conclusion that follows easily from Theorem 4.5 is that the nodes that are transmitters (resp. receivers) in the first timeslot assigned to an edge will be receivers (resp. transmitters) in the second timeslot assigned to the same edge.

From Theorem 4.4 we can conclude that if the graph $G(V, E)$ does not have any cycles, then it is always possible to give a valid sign to all the nodes in the graph G^g. From Section III we know that at most $(\delta + 1)$ colors are required for edge coloring any graph. Hence, from Theorems 4.4 and 4.5 we can conclude that:

Theorem 4.6: Timeslot allocation in all acyclic topologies can be accomplished using at most $2(\delta + 1)$ timeslots.

D. Set-up Phase

To color edges and assign signs, neighboring nodes need to communicate with each other. Hence, the question is how do nodes communicate even before the TDMA schedule is ready? We propose to employ an ad hoc TDMA schedule before the coloring and sign assignment is done. Once the timeslots are assigned to edges, nodes can switch to the computed schedule. As most of the messages exchanged by sensor nodes while coloring and signing are intended for all the neighboring nodes, we propose to employ a TDMA MAC wherein each node is assigned a timeslot (*broadcast scheduling*). We assume that a reasonably long TDMA time-frame is used during the setup phase of the sensor network. Each node picks a timeslot randomly and propagates a tuple of $< nodeID, slotnumber >$ in its two-hop neighborhood. Some well known mechanisms like CSMA/CA [5] can be used to propagate this information. If some nodes in a two-hop neighborhood have selected the same timeslot, the node with the highest node ID will retain the timeslot. All nodes which were not successful in selecting a timeslot will wait for sufficient time to learn about timeslot selection of all their two-hop neighbors. Then, these nodes will randomly select a slot from the remaining timeslots and repeat the above procedure until they grab a timeslot. Note that this slot assignment will be used by the nodes only during the coloring and sign assignment.

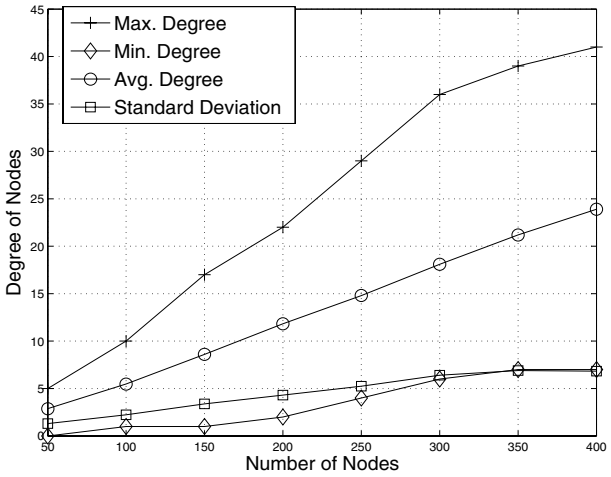
[3]As described in Section IV-B, DFS initiated by all other nodes in the connected component are aborted and do not impact the signing of nodes in G^g

V. Simulation Results

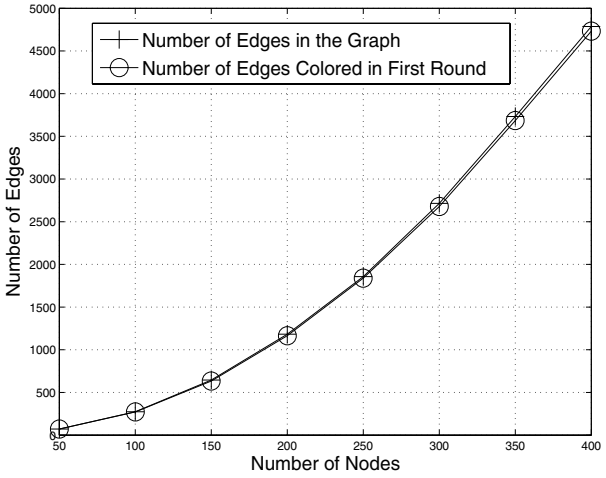
To evaluate the performance of the proposed link scheduling algorithm, we simulated a sensor network with nodes randomly distributed in a 200 × 200 meter-square area. The number of nodes in the network was varied from 50 to 400 in steps of 50. The transmission range of each sensor node was set to 30 meters. The maximum, minimum and average degree of nodes in the simulated networks are shown in figure 9(a). The energy spent in transmitting a bit over one meter distance was taken as 0.1 nJ/bit-m^2 [25] and the energy spent in receiving a bit was set to 50 nJ/bit [25]. Packets of variable length were used. The length of packet header was set to 40 bits. Node ID and colors were represented using 32 bits and 8 bits, respectively. The length of packets used to explore and invert cd-paths was set to 128 bits.

As explained in Section III-B, we propose to split the edge coloring of a graph into two rounds. In the first round, the goal is to color as many edges as possible without invoking the cd-path invert operation. Figure 9(b) shows the number of edges colored in the first round. It can be noticed that for sparse graphs almost all the edges are colored during the first round. For dense graphs, with average degree up to 24, around 98.85% of the edges are colored during this round. Thus, for coloring only a small fraction of nodes we need to invoke the cd-path invert operation. As the cd-path invert operation reduces the concurrency of distributed coloring, we conclude that the proposed algorithm colors almost all edges with a high level of concurrency.

Figures 10(a) and 10(b) show the maximum, minimum and average energy spent by sensor nodes while employing our link scheduling algorithm. It can be noticed from Figure 10(a) that increasing the number of nodes in the same area results in a polynomial increase in the average energy spent by a node. This is because adding more nodes results in polynomial increase in the number edges (see Figure 9(b)) that need to be assigned a timeslot. This inference is supported by Figure 10(b) which shows that the average energy spent by a node increases linearly with increasing number of edges.

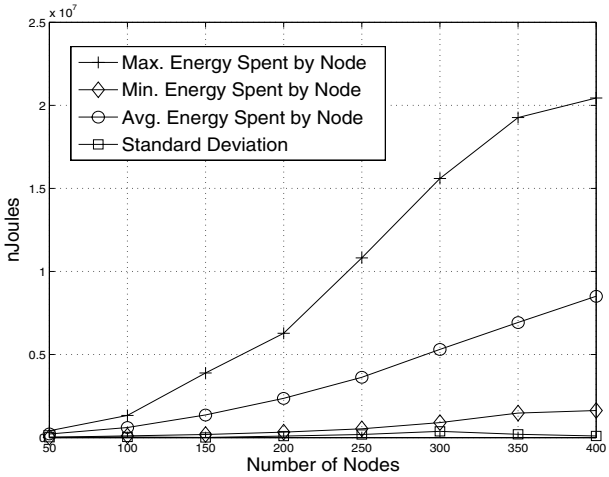


(a) Degree of Nodes.

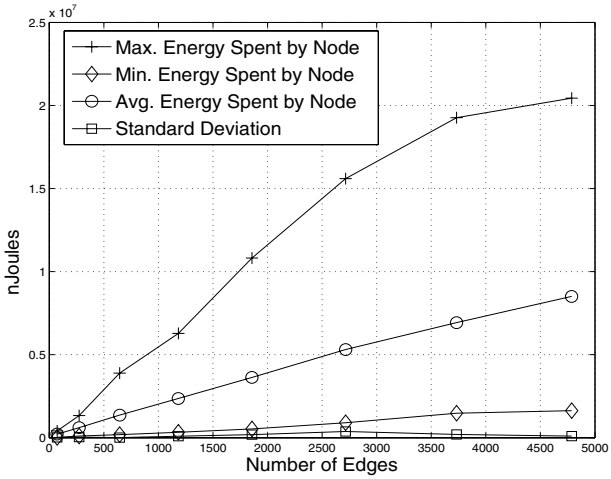


(b) Edges Colored in First Round.

Fig. 9.



(a) Varying the Number of Nodes



(b) Varying the Number of Edges

Fig. 10. Energy Spent in Link Scheduling.

The maximum energy spent by any node in a network of 400 nodes is around 0.02 Joules (Figure 10(a)). Typically, sensor nodes have couple joules of energy [22], [25]. Thus, the amount of energy spent by sensor nodes in assigning timeslots is acceptable.

Figure 11, compares the number of timeslots used against the value 2δ, the lower bound on required slots. For sparse topologies the actual number of slots used is close to the lower bound. There is a noticeable difference between the two values for dense topologies. This can be attributed to the fact that dense topologies have more number of cycles. More the number of cycles higher is the chance that we have cycles, whose edges cannot be assigned a feasible direction of transmission (refer to Theorem 4.4). Thus, requiring more number of timeslots. For dense topologies the optimum number of

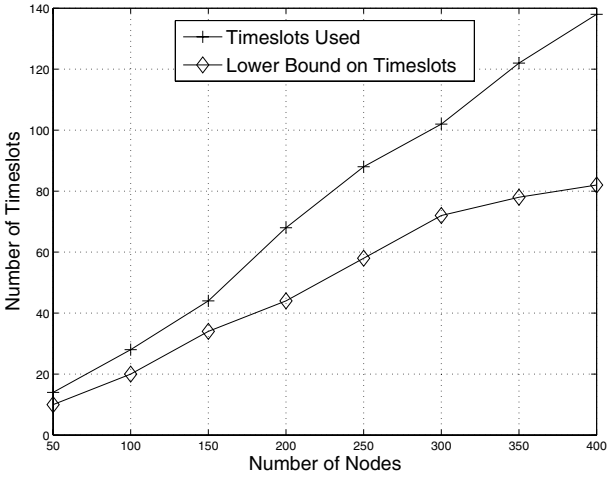


Fig. 11. Number of Timeslots Allocated.

timeslots would be much more than 2δ. For example, in a fully connected network, where $\delta = N-1$, to support duplex communication we need $N(N-1)$ ($O(\delta^2)$) timeslots. Here, N is the number of nodes in the network.

Though sensor nodes are deployed in large numbers, various topology control algorithms [13], [18] are proposed to increase the lifetime of the network. As a result of employing topology control algorithms, the degree of active nodes at any given instance of time is considerably less. For example, the topology control scheme proposed in [18] preserves the network connectivity and the node degree of any node is bounded by 6 in the derived topology. Thus employing our link scheduling algorithm in conjunction with known topology control schemes would ensure that number of timeslots used is close to the lower bound. This is a topic of future research.

In all the simulation experiments discussed above, the locations of the nodes were generated randomly. We repeated all the above experiments with nodes distributed uniformly in an area of 200 $\times$ 200 meter-square. Similar trends were observed.

VI. Conclusions and Future Work

We proposed a link scheduling algorithm for sensor networks employing a TDMA MAC protocol. The algorithm consists of two phases. In the first phase, each edge in the sensor network is assigned a valid color in a distributed fashion. The distributed edge coloring algorithm presented in this paper is the only known algorithm to edge color a graph using at most $(\delta+1)$ colors. In the second phase, each color is mapped to a unique timeslot and a direction of transmission is assigned to each edge. The direction of transmissions are such that both the hidden terminal problem and the exposed terminal problem are avoided. The topologies for which a feasible direction of transmission cannot be assigned are identified. For such topologies additional colors (timeslots) are added in a greedy fashion. Given a feasible direction assignment, it is shown that reversing the direction of transmission along every edge will result in another feasible direction assignment. The TDMA schedule to provide duplex communication between every pair of neighbors can be constructed using both feasible directions of transmission. We have shown that for all acyclic topologies a TDMA schedule can be constructed using at most $2(\delta+1)$ timeslots. Through simulations we demonstrated that for sparse graphs with cycles the number of timeslots used is close to $2(\delta+1)$. This result is of significance in sensor networks where nodes employ topology control schemes to reduce the degree of active nodes and to extend the lifetime of the network.

Our link scheduling algorithm expects the topology of the network to remain unchanged while timeslot assignment is done. However, topology might change in sensor networks due to displacement or failure of nodes. Fault tolerant edge coloring and signing algorithms are interesting research directions that we are currently exploring. We are also interested in extending the current link scheduling algorithm such that new sensor nodes when added to an existing network can efficiently assign timeslots to their edges.

References

[1] Madhav V. Marathe, A. Panconesi and Larry D. Risinger. An experimental study of a simple, distributed edge coloring algorithm. *Proc. of the twelfth annual ACM symposium on Parallel algorithms and architectures* , 2000.

[2] S. Ramanathan and E.L. Lloyd. Scheduling Algorithms for Multi-hop Radio Networks. *IEEE/ACM Transactions on Networking* , 1:166–177, 1993.

[3] A. Panconesi and A. Srinivasan. Improved Distributed Algorithms for Coloring and Network Decomposition Problems. *Proc. Of ACM Symposium on Theory of Computing*, 1992.

[4] A. Panconesi and A. Srinivasan. Randomized Distributed Edge Coloring. *SIAM Journal on Computing*, 26(2):350–368, 1997.

[5] Andrew S. Tanenbaum. Computer Networks, 3rd Edition. *Prentice-Hall Inc.*, 1996.

[6] B. Hajek and G. Sasaki. Link Scheduling in Polynomial Time. *IEEE Transactions on Information Theory*, 34:910–917, Sept. 1988.

[7] C. Berge. Graphs and Hyper Graphs. *North-Holland, Amsterdam*, 1973.

[8] D. Estrin, L. Girod, G. Pottie and M. Srivastava. Instrumenting the world with wireless sensor networks. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 2033–2036, 2001.

[9] David A. Grable and A. Panconesi. Nearly optimal distributed edge colouring in O(log log n) rounds. *Proc. of the eighth annual ACM-SIAM symposium on Discrete algorithms*, 1997.

[10] G.J. Pottie. Wireless sensor networks. *Information Theory Workshop*, pages 139–140, 1998.

[11] H. Tamura, K. Watanabe, M. Sengoku and S. Shinoda. On a New Edge Coloring Related To Multi-hop Wireless Networks. *APCCAS '02. 2002 Asia-Pacific Conference on Circuits and Systems*, Oct. 2002.

[12] Isreal Cidon. Yet Another Distributed Depth-First-Search Algorithm. *Information Processing Letters*, 26(6):301–305, 1998.

[13] J. Liu and B. Li. Distributed Topology Control in Wireless Sensor Networks with Asymmetric Links. *In Proceedings of IEEE GLOBECOM*, Dec. 2003.

[14] J. Misra and D. Gries. A Constructive Proof of Vizing's Theorem. *Inf. Proc. Lett.*, 41:131 – 133, 1992.

[15] K. Sohrabi, J. Gao, V. Ailawadhi and G.J. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications*, 7(5):16–27, 2000.

[16] LAN MAN Standards Committee of the IEEE Computer Society. Wireless LAN medium access control (MAC) and physical layer (PHY) specification. *IEEE, New York, NY, USA, IEEE Std. 802.11*, 1997.

[17] M. B. Sharma, N. K. Mandyam and S. S. Iyangar. An Optimal Distributed Depth-First-Search Algorithm. *Proceedings of the seventeenth annual ACM conference on Computer science : Computing trends in the 1990's*, 1989.

[18] N. Li, J. C. Hou, and L. Sha. Design and Analysis of an MST-Based Topology Control Algorithm. *In Proceedings of IEEE INFOCOM*, 2003.

[19] Praveen Rentala, Ravi Musunuri, Shashidhar Gandham and Udit Saxena. Survey on Sensor Networks. *UTD Technical Reports, UTDCS-10-03* , 2003.

[20] R. Ramaswami and K.K. Parhi. Distributed Scheduling of Broadcasts in a Radio Network. *IEEE INFOCOM*, 1989.

[21] S. Ramanathan. A Unified Framework and Algorithm for Channel Assignment in Wireless Networks. *Wireless Networks*, 5:81–94, 1999.

[22] Shashidhar Rao Gandham, Milind Dawande, Ravi Prakash and S. Venkatesan. Energy efficient schemes for wireless sensor networks with multiple mobile stations. *IEEE Globecom*, December, 2003.

[23] Sven O. Krumke, Madhav V. Marathe and S.S. Ravi. Models and Approximation Algorithms for Channel Assignment in Radio Networks. *Wireless Networks* , 7:575–584, 2001.

[24] Vaduvur Bharghavan, Alan Demers, Scott Shenker and Lixia Zhang. MACAW: A Media Access Protocol for Wireless LAN's. *ACM SIGCOMM*, 1994.

[25] W.R. Heinzelman, A. Chandrakasan and H. Balakrishnan. Energy-efficient communication protocol for wireless micro sensor networks. *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, pages 3005–3014, 2000.