Scheduling Algorithms for Multihop Radio Networks

Subramanian Ramanathan and Errol L. Lloyd

***Abstract*—New algorithms for transmission scheduling in multihop broadcast radio networks are presented. Both *link scheduling* and *broadcast scheduling* are considered. In each instance, scheduling algorithms are given that improve upon existing algorithms both theoretically and experimentally. Theoretically, it is shown that tree networks can be scheduled optimally, and that arbitrary networks can be scheduled so that the schedule is bounded by a length that is proportional to a function of the network *thickness* times the optimum. Previous algorithms could guarantee only that the schedules were bounded by a length no worse than the maximum node degree times optimum. Since the thickness is typically several orders of magnitude less than the maximum node degree, the algorithms presented here represent a considerable theoretical improvement. Experimentally, a realistic model of a radio network is given and the performance of the new algorithms is studied. These results show that, for both types of scheduling, the new algorithms (experimentally) perform consistently better than earlier methods.**

I. Introduction

A NETWORK of processors that communicate using broadcast radio is a *radio network*. Typical examples include packet radio networks, cellular phone networks, and satellite networks. The *stations* constituting a radio network share a common *radio channel* over which communication takes place. The multihop nature of most radio networks makes spatial reuse possible in the sharing or *assignment* of channels.The channel assignment considered here assigns transmission rights using time division multiplexing (TDM). In this method, transmissions that will not collide may overlap in time, thereby obtaining channel reuse in time.[1]

This is typically done by constructing a *schedule* [3]; that is, a sequence of fixed-length time slots, where each possible transmission is assigned a time slot in such a way that transmissions assigned to the same time slot do not collide. We address the problem of minimizing the number of time slots in such a schedule.

A. What is Scheduling?

In order to properly discuss the concept of scheduling, we first consider what is meant by a *collision*.[2] In particular, depending on the signaling mechanism, transmissions may collide in two ways—these are typically referred to as *primary* and *secondary* interference [12]. *Primary interference* occurs when the schedule is such that a station must do more than one thing in a single time slot—for instance, receive from two different transmitters. *Secondary interference* occurs when a receiver R tuned to a particular transmitter T is within range of another transmitter whose transmissions, though not intended for R, interfere with the transmissions of T.

We note that the scheduling protocol typically corresponds to the media access layer in the ISO–OSI model [29] and provides transparent channel access to the network layer. Depending on the service required by the network layer, there are two kinds of scheduling—*broadcast* and *link*.[3] In a *broadcast schedule*, the entities scheduled are the stations themselves. The transmission of a station is intended for, and must be received collision free by, *all* of its out-neighbors.[4] Here, primary interference is not tolerated, and it follows from the definitions that secondary interference does not arise. Thus, two stations may not be assigned to the same slot if they are either adjacent or have a common neighbor. In a *link schedule*, the *links*[5] between the stations are scheduled. The transmission of a station is intended for a particular out-neighbor, and we require that there be no collision at this receiver. Here, neither primary nor secondary interference is tolerated. Thus, two links may not be assigned to the same slot if either they are adjacent or there exists a third link from the transmitter of one link to the receiver of the other link.

B. An Overview—Our Approach and Results

In Sections III and IV, we present new algorithms for link and broadcast scheduling. Our interest in developing these algorithms has been to produce more nearly optimal schedules than existing algorithms. The schedule length is the most relevant measure of the performance of a scheduling algorithm, and it is important that the performance be good. In many applications, the transmission schedule is constructed just once (when the network is "brought up") and the actual data communication is done with this schedule for as long as the network remains up. In this context, note that a schedule that uses, for example, even one extra slot every five seconds thereby

Manuscript received July 15, 1992; revised January 1993; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Chih-Lin I. This work was supported in part by the National Science Foundation under Grant CCR-9120731. An earlier version of this paper was presented at SIGCOMM'92.

S. Ramanathan is with BBN Systems and Technologies, Cambridge, MA.(email: ramanath@udel.edu)

E. L. Lloyd is with the Department of Computer Science, University of Delaware, Newark, DE 19716. (email: elloyd@udel.edu)

IEEE Log Number 920828.

[1] An analogous technique whereby channel reuse in frequency is obtained in frequency division multiplexing (FDM). All results in this paper uniformly hold for FDM as well as TDM.

[2] The reader is referred to [3], [19] for a thorough treatment of the basic issues in channel assignment for radio networks. We give here only the basic concepts that are required in this paper.

[3] Link scheduling has also been referred to as *point-to-point* scheduling, *link activation* scheduling, and *receiver-directed* scheduling.

[4] If station x can transmit a message to station y, then y is an *out-neighbor* of x and x is an *in-neighbor* of y.

[5] If x can transmit a message to y, there is a *link* from x to y.

"wastes" 720 slots per hour of network operation. Clearly, it pays to invest in algorithms that reduce the schedule length.

Previous work on link scheduling includes [7], [12], [25], while work on broadcast scheduling includes [9], [11], [28]. The work in these papers covers aspects of scheduling such as distributed implementation and adaptation to toplogical changes. Little work has been done on developing algorithms that produce short schedules. Perhaps one reason for this lack of work is that finding short schedules is not easy. In particular, the problem of finding an *optimal* schedule, that is a minimum-length schedule, is NP-complete for both link and broadcast scheduling [10], [2], [28]. Further, both link and broadcast scheduling are closely related to the classic graph theoretic problem of vertex coloring [27]. This is a notoriously hard problem for which the best existing *approximation* algorithms are quite poor [30], and for which it is widely believed that there do not exist approximation algorithms with provably good worst-case performance bounds [14], [17].

Fortunately, the situation in practice may not be as bad as the above discussion indicates. It might be the case that radio networks are best modeled using various restricted classes of graphs (and that excellent scheduling algorithms can be formulated for those restricted classes). Indeed, it is claimed in [6] that most existing packet radio networks may be modeled by trees. While this may be true at present, the notion that packet radio networks may be modeled as trees seems to be too restrictive for long-term use. On the other hand, our experimental investigations in this regard have shown that even in the most general case, radio networks *can* be adequately modeled by *planar* or *close-to-planar* graphs. Intuitively, this is a consequence of the fact that the radio stations are "spread" over a geographical area and each station can only communicate with stations in its vicinity. In this paper, we study both trees and planar graphs as relevant restricted cases, and show that *tree networks* can be scheduled optimally and that *planar networks* can be scheduled nearly optimally.

Somewhat surprisingly, the study of these restricted cases yields considerable insight into the nature of the general problem and other possible solutions. In particular, we show that even if a network cannot be modeled by a planar graph, the worse-case performance of our algorithms may be expressed as a function of *how planar the graph is*. We use the notion of the *thickness* of a graph to measure is "nearness to planarity." Here, thickness is the minimum number of planar graphs into which a given graph can be partitioned. In this context, we show that for a graph having thickness θ and a maximum vertex degree of ρ, the worst-case number of slots used by our algorithms is proportional to $\theta^2\rho$ for link scheduling and proportional to $\theta\rho$ for broadcast scheduling. Previous algorithms could guarantee only that the schedules were bounded by a length proportional to ρ^2 for link scheduling [7]. For broadcast scheduling, a bound proportional to ρ^2 was shown in [28] and a bound proportional to $\rho \log N$ (N is the number of vertices) was shown in [1]. Since the thickness is typically several orders of magnitude less than the maximum station degree and $\log N$, the algorithms presented here represent a considerable improvement over existing methods in terms of worst-case performance.

In addition to the worst-case performance bounds, we provide an experimental analysis of our algorithms and their performance as it relates to existing methods. This analysis includes the formulation of a realistic experimental model of a radio network and shows that, for both types of scheduling, our new algorithms perform consistently better than previous methods. In particular, our experiments show that in comparison to previous methods, the new methods use about 8% fewer slots for link scheduling and 10% fewer slots for broadcast scheduling.

The remainder of the paper is organized as follows. Section II gives the definitions of terms and the explanation of concepts that we use in analyzing (theoretically and experimentally) our algorithms. Section III is devoted to link scheduling and Section IV to broadcast scheduling. Section V concludes with a summary of the results.

II. Preliminaries

In this section, we provide definitions of, and explanations for, some of the fundamental notions that we employ throughout later sections. These include certain graph theoretic concepts, the network model used in our experiments, and definitions related to bounds on the performance of approximation algorithms.

A. Graphical Representation

Our presentation of scheduling algorithms in Sections III and IV is based on a standard representation of a radio network by a directed graph $G = (V, A)$. Here, V is a set of *vertices* denoting the stations in the radio network, and A is a set of directed edges between vertices such that for any two distinct vertices $u, v \in V, (u, v) \in A$ if and only if v can receive u's transmission. Note that we do not make an *a priori* assumption about the edges of the corresponding graph being bidirectional.[6] That is, $(u, v) \in A$ does not necessarily imply $(v, u) \in A$.

A natural interpretation of scheduling in this context is as one of *coloring* the corresponding graph $G = (V, A)$. Thus, broadcast scheduling is one of *coloring* the vertices of the graph such that any pair of vertices a, b may be colored the same if and only if:

1) edge $(a, b) \notin A$ and edge $(b, a) \notin A$, and
2) there is no c such that $(b, c) \in A$ and $(a, c) \in A$.

When the first condition fails to hold, there is a *primary vertex conflict* between vertices a and b, and when the second condition fails to hold, there is a *secondary vertex conflict* between a and b.

Similarly, link scheduling corresponds to *coloring* the edges of the graph such that any pair of directed edges $(a, b), (c, d)$ may be colored the same if and only if:

1) a, b, c, d are all mutually distinct, and
2) $(a, d) \notin A$ and $(c, b) \notin A$ (recall that edges are not necessarily bidirectional).

[6] In some networks, extraneous noise or deliberate jamming at the site of one station [12] may cause one-way connections.

Here, when the first (second) condition fails to hold, there is a *primary (secondary) edge conflict* between (a, b) and (c, d).

In subsequent sections, we will interchangeably use the terminology of coloring and that of scheduling in referring to the problems we consider. The choice of terminology will depend on which is clearer in a given context.

B. The Experimental Model

In this subsection, we discuss the generation of radio networks that we use for experimentally studying the performance of our algorithms and existing methods.

As we noted in the previous subsection, the problem of scheduling a network is equivalent to one of coloring the corresponding graph. The experimental performance of some well-known coloring problems such as vertex coloring has traditionally been studied using a simple probabilistic model. In this model, a graph of a given number of vertices is generated by placing an edge between two vertices according to some probability distribution [5]. One such approach is to generate each edge with a fixed probability p. Unfortunately, the application that we are concerned with (radio networks) is not realistically modeled in such a fashion. Typically, radio network stations are *not* equally likely to be connected to all of the other stations. The connectivity depends on the geographical location of the stations, and a station has a link to the stations that are within a certain distance from itself. This unique property of link "locality" must be captured in any realistic model of radio networks.

Our experiments have been conducted under the assumption of a noiseless, immobile radio network in which all of the stations have the same transmission radius. In this context, the network may be represented by the three-tuple (N, R, P) where N is the number of stations, R is the transmission radius of each station, and $P = \{(x_i, y_i, 1 \leq i \leq N\}$ is the set of locations for each of the stations. The location of a station is generated randomly, using a uniform distribution for its X and Y coordinates, in a given area. We convert this network into a graph $G = (V, A)$ so that $|V| = N$, and $(u, v) \in A$ if and only if the Euclidean distance between (x_u, y_u) and (x_v, y_v) is less than or equal to R. Under this model, all edges in the graph are bidirectional.

We have studied the experimental performance of our algorithms by generating a number of random graphs for various "typical" values of N and R. Since real-life data is scarce in this area, we have chosen, as in [16], a range of values that one might *expect* for future applications. In the tables appearing in Sections III and IV, the results for each pair of (N, R) values are obtained by averaging over thirty random graphs generated with those values.

C. Approximation Algorithms

When dealing with optimization problems that are NP-complete, a common approach is to devise algorithms that produce approximate (i.e., nonoptimal) solutions. A standard measure of the performance of such an *approximation algorithm* is in terms of the ratio of the sizes of the approximate solution produced by the algorithm and an optimal solution. Formally, the performance of an algorithm A is given by its *performance guarantee* $\gamma = \max \{A(G)/\mathrm{OPT}(G)\}$, taken over all graphs G, where $A(G)$ denotes the size of a solution given by algorithm A, and $\mathrm{OPT}(G)$ denotes the size of an optimal solution. In conformance with existing notions, an approximation algorithm is *good* if its performance guarantee is O(1). In our context, this means that an algorithm is *good* if it produces a coloring using a constant times optimal number of colors. As we noted less formally in the Introduction, there do not exist any good approximation algorithms for the standard graph vertex coloring problem and, further, it is widely assumed that no such algorithms exist. On the other hand, there exist a wide range of NP-complete problems for which there are good approximation algorithms [8].

In our experimental analysis (Sections III-D and IV-C), we have compared the performance of our algorithms to *existing* algorithms and not *optimal* algorithms since, even for small networks, the time requirements of algorithms that obtain optimal solutions (using exhaustive search) are prohibitive.

III. Link Scheduling

In this section, we study link scheduling. As noted earlier, this is done in the context of coloring the edges of the corresponding directed graph. Recall that, in link scheduling, any pair of directed edges (a, b) and (c, d) may be colored the same if and only if a, b, c, d are all distinct, $(a, d) \notin A$, and $(c, b) \notin A$ [i.e., there is no primary or secondary edge confict between (a, b) and (c, d)].

Also recall that it is NP-complete to construct an optimum link schedule for an arbitrary graph. In fact, we show in [20], [21] that this holds even when we restrict our attention to planar graphs.

Fact 3.1: For a planar network, determining the existence of a link schedule using seven colors is NP-complete.

Given these NP-completeness results, this section examines what *can* be done in regard to link scheduling. In Section III-A, we describe a link scheduling algorithm for tree networks. This algorithm runs in polynomial time and does indeed produce an optimal link schedule (for tree networks). In Section III-B, we consider a more complicated situation involving oriented graphs (these are slighly generalized trees). Here, additional "interference" edges are present that, while not themselves needing to be colored, do invoke additional secondary conflicts that must be accomodated in the coloring of the "ordinary" edges of the oriented graph. Interestingly, the notion of interference(nonscheduled) and ordinary(scheduled) edges, though invented here as a conceptual step towards the actual coloring algorithm, models a realistic situation in radio networks. In some networks, the signal transmitted by a user i to a user j may be too weak to be decoded at j but may be strong enough to *interfere* with another signal arriving at j. In such a case, there would be an *interference edge* from i to j. Interference edges were considered in [26].

Since finding an optimal coloring in this situation is NP-complete, we provide an algorithm that produces an approximate solution. In Section III-C, we use the results of Section III-B to provide a new polynomial time algorithm for finding link schedules for arbitrary networks. The performance

guarantee for this algorithm is O(1) for planar graphs and $O(\theta^2)$ for arbitrary graphs of thickness θ. Finally, in Section III-D, we present some experimental results.

A. Tree Networks

In this section, we give an algorithm for optimum link scheduling of *tree networks*. These are connected[7] networks where, if the directions are removed from the edges, then the underlying graph is a tree. Without loss of generality, we assume that some vertex of the network is designated as the root, and hence the terminology of *parent* and *child* with respect to the tree follows naturally. Note that, between a parent x and a child y, there are three possibilities for the edge(s) present in the tree network: $(x, y), (y, x)$ or both. Also, a vertex located at distance k from the root is at *level* k. An edge is at level k if k is the greater of the levels of its endpoints, and a tree with a maximum level of k is said to have k levels.

The algorithm given colors the edges of a tree using a breadth first search starting from the root. Intuitively, the tree is colored *by level*. In this context, consider the coloring of (directed) edge (x, y) where x is the parent of y. Since the coloring is being done by levels, the only edges having either primary or secondary edge conflicts with (x, y) are edges that are either incident to x or are incoming to the parent of x. In particular, this means that the colors of the edges outgoing from the parent of x can be used to color edge (x, y), assuming that no other edge incident on x has already been so colored. Giving priority to such a *reuse* of colors is the primary observation needed to ensure that the coloring will be optimum. The details of the algorithm follow.

Algorithm TreeSched.
Input: A tree network $T = (V, A)$ with root r
Output: A coloring $c : A \rightarrow \{1, 2, \cdots\}$.
color the edges incident on r using
colors $1, 2, \cdots$, degree(r)
subtreecolor(r)
The procedure subtreecolor colors the remainder of the edges and is described next.
procedure subtreecolor(b)
for each child a of b in T do
$B_{\text{In}} \leftarrow \{c(u, b) : (u, b) \in A$ and colored$\}$
$B_{\text{Out}} \leftarrow \{c(b, u) : (b, u) \in A$ and colored$\}$
for each uncolored edge (x, a) do
if B_{In} is not empty
then let j be any color in B_{In}
$B_{\text{In}} \leftarrow B_{\text{In}} - \{j\}$
else let $j \leftarrow$ NonConflictingEdge $((x, a))$
$c(x, a) \leftarrow j$
for each uncolored edge (a, x) do
if B_{Out} is not empty
then let j be any color in B_{Out}
$B_{\text{Out}} \leftarrow B_{\text{Out}} - \{j\}$
else let $j \leftarrow$ NonConflictingEdge((a, x))
$c(a, x) \leftarrow j$
for each child a of b in T do
subtreecolor(a)

[7]Every station is reachable from every other station.

The function NonConflictingEdge simply collects the colors of all edges conflicting with the given edge and returns the first color in this set. It is described next.

function NonConflictingEdge(e)
Conflicting $\leftarrow \{c(h) : h$ is colored and e and h have a primary edge conflict $\} \cup$ $\{c(h) : h$ is colored and e and h have a secondary edge conflict$\}$.
return the least color $\notin$ Conflicting.

To see that the algorithm produces a conflict-free coloring, we first observe that a color is chosen either using the NonConflictingEdge function or from the nonadjacent edges at the previous level. Clearly, NonConflictingEdge returns a nonconflicting color. If the color is a reuse of a color in the previous level, it has to be in the *same* direction. That is, if we are coloring an edge outgoing (incoming) to "a," we choose a color from the outgoing (incoming) edges to a parent of "a." Since such edges do not conflict and we reuse a color at most once for edges incident on "a," the coloring is conflict free.

With a very careful implementation, the running time of the algorithm can be shown to be $O(v \log \rho)$, where ρ is the maximum degree of any vertex.

The remainder of this section is devoted to showing that:

Theorem 3.1: Algorithm TreeSched colors a tree network T with the optimum number of colors.

Proof: The proof is by induction on k, the number of levels in the tree network. When the tree network consists of only the root and its children, it follows immediately from the special handling afforded the root that exactly ρ colors are used (recall that ρ is the maximum (total) degree of any vertex—in this case, it is the degree of the root).

Thus, assume inductively that the theorem holds for all tree networks having fewer than k levels, and consider a tree network T with k levels. Thus, (inductively) the algorithm produces an optimum coloring of T, not including the edges of level k. Note that all of the calls to subtreecolor for the vertices of level k are completely independent (i.e., the edges colored in distinct calls are sufficiently far apart that neither primary nor secondary conflicts are possible). In this case, it is sufficient for us to concentrate on a single call to subtreecolor for a vertex a of level k. Thus, analogous to the body of subtreecolor, let $a_{\text{in}}, b_{\text{in}}, a_{\text{out}}, b_{\text{out}}$ be as shown in Fig. 1.

There are two situations, depending on whether there are edges in both or only one direction(s) between nodes a and b. In the proof given here, we assume that both edges are present. The other situation can be analyzed in a similar fashion, and is not presented. There are four cases.

Case 1: $a_{\text{out}} \leq b_{\text{out}}; a_{\text{in}} \leq b_{\text{in}}$. It follows from subtreecolor that each of the edges incident between vertex a and its children will be colored using colors of edges in b_{in} or b_{out}. Since no new colors are required, it follows that the overall coloring is optimum.

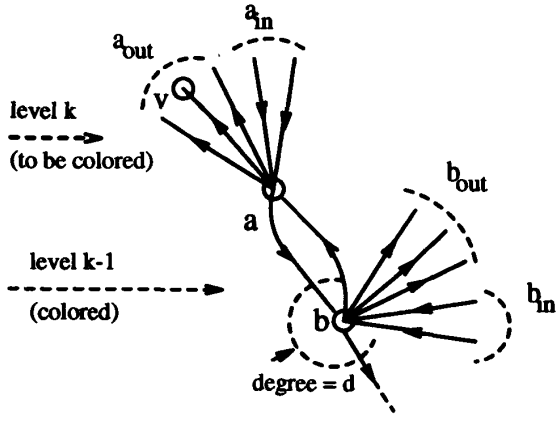

Fig. 1. Link scheduling a tree optimally.

Case 2: $a_{\text{out}} \leq b_{\text{out}}; a_{\text{in}} > b_{\text{in}}$. Let $x + d$ be the number of colors used to color T not including the edges of level k. Here, d is the degree of b and, hence, d is also the number of colors used to color edges incident on b. Then, x is the number of colors not used to color edges incident on b. Note: $d = b_{\text{in}} + b_{\text{out}} + 2$, and colors in b_{in}, along with the "x" colors, may be used to color the incoming edges to "a."

It follows that the total number of colors utilized, including the coloring of the edges incident to a, is $x + d +$ max(0, $(a_{\text{in}} - b_{\text{in}}) - x$). If the max term is 0, we are done since the number of colors used has not increased. Thus, suppose that term is nonzero and simplify the entire expression to $d+(a_{\text{in}}-b_{\text{in}}) = (b_{\text{in}}+b_{\text{out}}+2)+a_{\text{in}}-b_{\text{in}} = a_{\text{in}}+b_{\text{out}}+2$. Now note that the optimum number of colors is at least $a_{\text{in}}+b_{\text{out}}+2$, since each edge outgoing from b has a secondary edge conflict with each edge incoming to a. It follows that the overall coloring is optimum.

Case 3: $a_{\text{out}} > b_{\text{out}}; a_{\text{in}} \leq b_{\text{in}}$. This is completely analogous to Case 2.

Case 4: $a_{\text{out}} > b_{\text{out}}; a_{\text{in}} > b_{\text{in}}$. This is a modified version of Case 2. As before, let $x+d$ be the number of colors used to color T not including the edges of level k. Now, the x colors not used in edges incident on b may be used to color both the incoming and outgoing edges of vertex "a."

It follows that the total number of colors utilized, including the coloring of the edges incident to "a," is $x + d +$ max(0, $(a_{\text{in}} - b_{\text{in}}) + (a_{\text{out}} - b_{\text{out}}) - x$). If the max term is 0, we are done since the number of colors used has not increased. Thus, suppose that the term is nonzero and simplify the entire expression to $d+(a_{\text{in}}-b_{\text{in}})+(a_{\text{out}}-b_{\text{out}}) = (b_{\text{in}}+b_{\text{out}}+2)+(a_{\text{in}}-b_{\text{in}})+(a_{\text{out}}-b_{\text{out}}) = a_{\text{in}}+a_{\text{out}}+2$. Now note that the optimum number of colors is at least $a_{\text{in}} + a_{\text{out}} + 2$ since this is the degree of vertex "a." It follows that the overall coloring is optimum. □

B. Oriented Graphs with Interference Edges

In the next section, we provide a new algorithm for link scheduling of arbitrary networks. The motivation for that algorithm, and the key to its good performance, is the recognition of the fact that while the graph considered as a whole is difficult to color, it can be decomposed into several *pieces* each of which are fairly easy to color. These pieces can then be recombined to produce a good (though not necessarily optimal) coloring of the entire graph.

Given that we can color trees optimally, an obvious candidate for a "piece" is a tree. Unfortunately, we are unable to utilize the algorithm of the previous section since the coloring of each piece (a tree) *cannot* be done in isolation from the other pieces (i.e., the edges of the other trees in the decomposition). The remainder of this section is devoted to handling coloring in this situation. Because the coloring of each decomposed piece must take into account the influence of the other pieces, the algorithm that we present lies somewhere between the approaches of coloring in one "global" swoop and coloring purely "locally."

In the context of the algorithm given in the following section, there are two additional considerations. First, it is a bit more convenient to work with a slight generalization of trees rather than trees themselves. Second, in order to obtain a good performance bound, certain values must be bounded. Specifically, when coloring an edge e, the number of potential edge conflicts with e must be bounded in some way. Thus, the algorithm of the next section does not decompose the graph into trees but rather into *oriented graphs.* Here, an *in-oriented* graph is one in which every vertex has at most one outgoing edge (that is, a local view of a vertex shows several incoming edges but only one outgoing edge). Similarly, an *out-oriented* graph is one in which every vertex has at most one incoming edge.

Thus, in this section, we assume that we are given an oriented graph T and a set of *interference edges* between the vertices of T. The problem is to provide a proper coloring of T's edges, even taking into consideration the additional secondary conflicts induced by the interference edges. Note that the interference edges do *not* have to be colored–they simply induce conflicts between edges that must be colored. This notion is best understood in the context of decomposing a graph G into oriented graphs. If T is an oriented graph in that decomposition, then all edges of G that are not part of T itself will be taken as *interference edges* with respect to T. For instance, consider Fig. 2. There, the solid edges are the edges of T, and it is those edges that must be colored. The remainder of the graph is shown using dashed edges. The dashed edges are the interference edges since they cause interference between the edges of T. The interference edge (a, d), for example, causes a secondary conflict between edges (a, b) and (c, d) forcing them to receive different colors.

A natural question is whether an optimal coloring of an oriented graph with interference edges may be found easily. Unfortunately, in [27] we show:

Fact 3.2: It is NP-complete to find an optimal coloring of an oriented graph with interference edges.

Thus, we are left to consider methods that obtain only near-optimal solutions. One such method is the subject of this section. The method presented here will be used in the following section as a subroutine in our algorithm for the coloring of arbitrary networks.

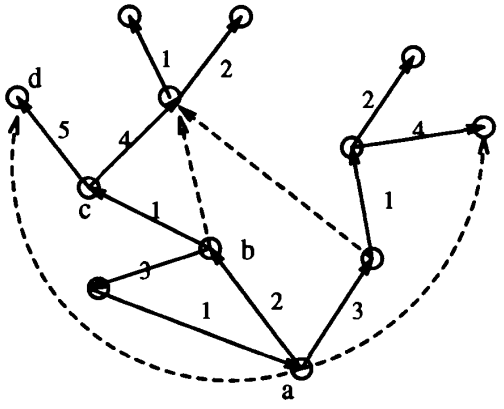

Fig. 2. Oriented graph coloring with interference edges.

The algorithm for coloring an oriented graph with interference edges has two phases. In the first phase, a unique *label* is assigned to each of the vertices of G. In the second phase, the edges of the oriented graph are colored. This is accomplished by considering the vertices in increasing order by label. For each vertex x, we take the sole incoming/outgoing edge incident on x and assign a color to that edge. That assigned color is the lowest numbered color that can be assigned without causing a conflict with previously colored edges. We note that the ordering of the vertices is crucial to bounding the worst-case performance of the algorithm.

As a precursor to the actual coloring algorithm, we first provide the details of the labeling mentioned previously. In particular, we give a function *labeler* which takes as input an oriented graph with interference edges, and assigns a unique label to each vertex of that graph. The function returns the highest assigned label.

integer **labeler**(T)
  if T is not empty
    let u be a vertex of T of minimum degree
      (considering edges in both T and I)
    $L(u) \leftarrow 1+$ labeler$(T-u)$
    return $L(u)$
  else
    return 0

Here, in an abuse of notation we let $T-u$ denote the oriented graph with interference edges that result when the vertex u and all of its incident (graph and interference) edges are deleted from T.

The algorithm for scheduling the edges of an oriented graph with interference edges is given. The algorithm uses the function NonConflictingEdge, which was described in Section III-A.

Algorithm OrientedGraphSchedule
Input: $T = (V, A, I)$, an oriented graph with interference edges, where V and A are the vertices and edges of the graph and I is the set of interference edges
Output: An assignment of colors(slots) $c : A \rightarrow \{1, 2, \cdots\}$
Phase 1:
  $n \leftarrow$ labeler (T)
Phase 2:
  for $j \leftarrow 1$ to n do
    if T is out-oriented
      then let $e = (u, v)$ be such that $L(u) = j$
      else that $e = (u, v)$ be such that $L(v) = j$
    $c(e) \leftarrow$ NonConflictingEdge(e)

To see that this algorithm yields a legal coloring of an oriented graph, we assume inductively that the coloring constructed through the $(i-1)$st iteration is legal. Then, let $e = (u, v)$ be the edge to be colored in the ith iteration. The colors of the edges that interfere with (hence, influence the color of) e are collected in the set Conflicting (of function NonConflictingEdge). These edges are shown in Fig. 3. Note that interference edges incoming to u and outgoing from v do not introduce conflicts between e and other edges of the oriented graph being considered. Since we assign to e a color that is different from the color assigned to each of the edges interfering with e, the coloring inclusive of edge e (ith iteration) is legal. Thus, the algorithm produces a legal coloring of each oriented graph.

In regard to the number of colors used by the above algorithm, we have:

Lemma 3.1: Suppose that each vertex of T has at most β neighbors with lower labels. Then, T may be colored using no more than $O(\beta\rho)$ colors.

Proof: We prove the lemma for an out-oriented graph. The proof for an in-oriented graph is analogous. Thus, we begin by considering the coloring of an out-oriented graph using OrientedGraphSchedule, and let n_1, n_2, n_3, n_4 be the number of colors added to the set Conflicting from the sources as shown in Fig. 3. It suffices to show that $n_1 + n_2 + n_3 + n_4$ is $O(\beta\rho)$.

It is easy to see from Fig. 3 that $n_1 \leq \rho - 1$. Also, since there can be at most one edge of the oriented graph incoming to x_i for any interference edge (u, x_i), it follows that $n_2 + n_3$ is $O(\beta\rho)$.

We show that n_4 is $O(\beta\rho)$ by carefully examining interference edges contributing to n_4. Let (x_i, y_i) be an edge in n_4 and let (x_i, v) be the interference edge that induces a conflict between (x_i, y_i) and (u, v) (see Fig. 3). Then, either $L(x_i) > L(v)$ or $L(x_i) < L(v)$. Suppose that there are k edges (x_i^1, v) such that $L(x_i^1) < L(v)$ and $\rho - 1 - k$ edges (x_i^2, v) such that $L(x_i^2) > L(v)$ (note that $k \leq \beta$).

Clearly, there can be at most $\rho - 1$ edges from T incident on x_i^1 and at most $\rho - 1$ edges from T incident on x_i^2. However, of the latter $\rho - 1$ edges, at most $(\beta - 1)$ edges (x_i^2, y) incident on x_i^2 are *colored* at this point. This is because, for (x_i^2, y) to be colored, we require that $L(y) > L(v)$. However, since $L(x_i^2) > L(v)$, it follows that $L(y) < L(x_i^2)$ and, by our

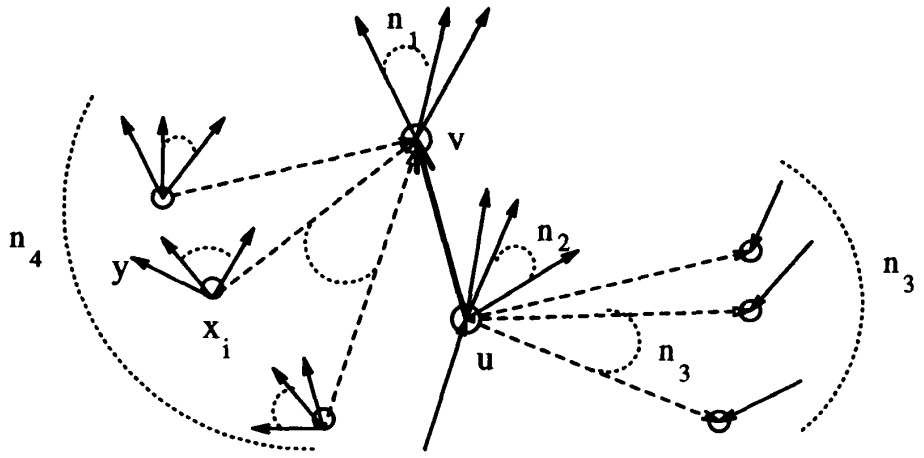

Fig. 3. Neighborhood of edge to be colored in Link Schedule.

assumption, there can be only $\beta - 1$ such y's, excluding v itself.

Thus, $n_4 \leq k(\rho - 1) + (\rho - 1 - k)(\beta - 1)$. Since $k \leq \beta, n_4$ is $O(\beta\rho)$.

Regarding the time complexity, we have

Lemma 3.2: The running time of Phase 2 of OrientedGraphSchedule is $O(v\beta\rho)$[8].

Proof: From Lemma 3.1, we have that the size of the set Conflicting (of function NonConflictingEdge) is $O(\beta\rho)$. Thus, finding a new color for an edge in Phase 2 takes $O(\beta\rho)$ steps. Since this is done once for each label (and hence for each vertex), it follows that the overall running time in Phase 2 of OrientedGraphSchedule is $O(v\beta\rho)$. □

C. Arbitrary and Planar Networks

The main result of this section is a new algorithm for link scheduling. The performance guarantee for this algorithm is O(1) for planar graphs and $O(\theta^2)$ for arbitrary graphs of thickness θ. In what follows, we begin with a description of the algorithm, followed by an analysis of its performance.

1) The Algorithm: As noted in the previous section, our algorithm begins by decomposing the graph G into several *in-oriented* and *out-oriented* graphs $T_1, T_2 \cdots T_k$. The decomposition is such that while the T_i's are not necessarily vertex disjoint, it *is* the case that every edge of G is in exactly one of the T_i's. For any particular T_i, the edges of G that are not in A_i are interference edges for T_i.

The decomposition of G into oriented graphs is done by first partitioning the *undirected* equivalent of the graph into undirected forests. It is possible to do this in an optimal fashion using the techniques given in [15], [18] or it may be done nonoptimally, but somewhat more speedily, by using successive breadth first searches (the latter approach is the one that is implemented and experimentally studied). Following this initial partitioning, each of the undirected forests is split further into two forests—one containing the edges pointing away from the root and the other containing edges pointing towards the root. The former produces out-oriented graphs and the latter produces in-oriented graphs. This splitting of the undirected forests into two oriented graphs is straightforward, and is not considered further here.

Recall that the algorithm for link scheduling of oriented graphs had two phases: the first for labeling the vertices, and the second for actually assigning colors. It should be apparent that the same labeling will be done for each of graph T_i, since that labeling takes into account both graph and interference edges in determining vertex degrees. Thus, for efficiency, the labeling is performed only once. This labeling may be done either before or after the decomposition into oriented graphs. For convenience in explanation, we assume it occurs before the decomposition is performed. With the preliminaries concluded, we now present the entire link scheduling algorighm.

Algorithm ArboricalLinkSchedule
Input: A directed graph $G = (V, A)$
Output: A coloring $c : A \rightarrow \{1, 2, \cdots\}$
$n \leftarrow$ labeler(T)
Decompose G into oriented graphs $T_i, 1 \leq i \leq k$
for $i \leftarrow 1$ to k do
  apply Phase 2 of OrientedGraphSchedule to graph T_i, letting the interference edges $I = A - T_i$}

Fig. 4 illustrates the execution of the algorithm. The various "steps" are labeled on the arrows. In Step 1, the vertices are labeled (the labels are shown within parantheses), directed edges are coalesced, and directions removed to form the undirected equivalent. Step 2 decomposes this graph into two trees. Step 3 divides the first of these into out-oriented and in-oriented graphs, and labels them. Step 4(a) and 4(b) show the "greedy" coloring of the edges of each of the oriented graphs taken in order of the labeling and taking a fresh set of colors for each. The dashed edges are the interference edges from the remainder of the graph. The processing of the other undirected tree is similar and is not shown here.

[8] In stating the running times, we use v and e to denote the number of vertices and the number of edges, respectively.

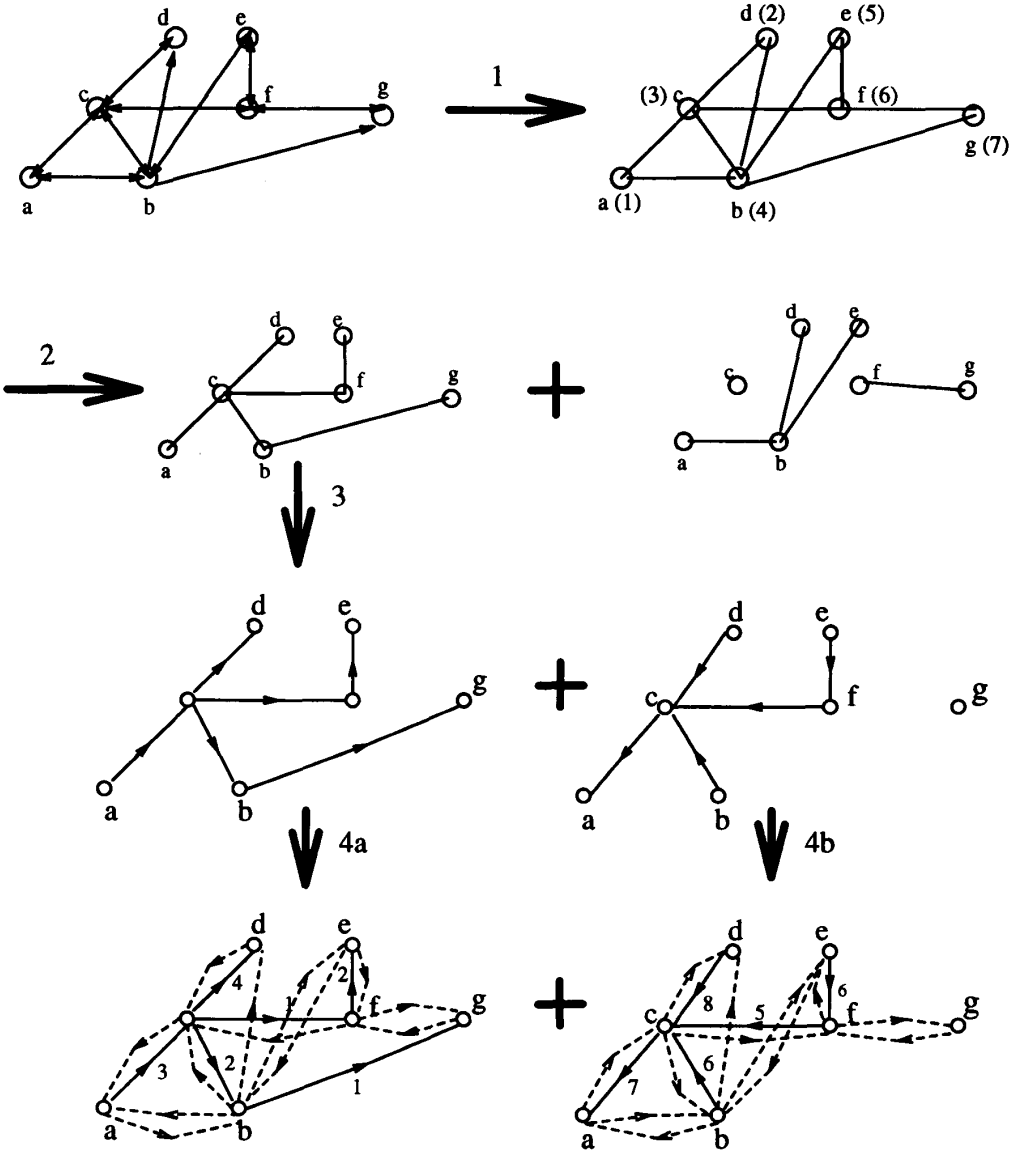

Fig. 4. Steps in the aborical partition and coloring.

2) The Analysis: In this subsection we consider both the performance bounds and time complexity of LinkSchedule. The fact that the algorithm provides a legal coloring is easy to see and is omitted. Thus, we begin by considering the performance of ArboricalLinkSchedule in terms of the quality of the coloring that is produced.

We begin by letting, for each $i, 1 \leq i \leq k$, c_i be the number of *fresh* colors used in the coloring of T_i. That is, c_i denotes the number of colors that are used for the first time in the coloring of T_i.

It follows from Lemma 3.1 that:

Lemma 3.3: For each $i, 1 \leq i \leq k$, c_i is $O(\beta\rho)$. Thus, since the total number of colors used by the algorithm is equal to the sum of the c_i's, it follows that:

Lemma 3.4: **ArboricalLinkSchedule** uses no more than $O(k\beta\rho)$ colors, where k is the number of oriented graphs, β is the maximum number of neighbors with lower labels, and ρ is the maximum vertex degree.

3) Planar networks: Consider a call of the function labeler when the input graph G is planar. Since every subgraph of a planar graph is also planar, it follows that in each recursive call to labeler, the input graph is planar. Now, in any such call, consider the vertex u of minimum degree. By a fundamental property of planar graphs [4], that minimum degree vertex has at most five neighbors. These five neighbors of u will all receive labels lower than $L(u)$. All other neighbors of u in the original graph G will receive higher labels than does u. Thus,

Lemma 3.5: If G is planar, then for each vertex of G there are at most five neighbors with lower labels.

In terms of Lemma 3.4, this means that $\beta \leq 5$. Further, it is known [24] that a planar graph can be partitioned into at most three undirected forests. Since the algorithm of [15] gives an optimal partitioning, it follows that a planar G is partitioned into no more than six oriented graphs. Hence, $k \leq 6$, and it follows from Lemma 3.4 that if G is planar, then ArboricalLinkSchedule uses no more than $O(\rho)$ colors. Since for any G the optimal number of colors required is trivially at least ρ, the performance guarantee $\gamma = O(1)$.

As far as the running time is concerned, let e and v denote the number of edges and the number of vertices, respectively, in the input graph $G = (V, E)$. The function labeler can be done in time $O(e + v \log v)$ by using a Fibonacci Heap [13] to select a vertex of minimum degree and to update vertex degrees as the graph is reduced via the recursive calls. The *decomposition* of the input graph into oriented subgraphs requires time $O(ev \log v)$ [15]. Since $k \leq 6$, it follows from Lemma 3.2 that the for-loop of ArboricalLinkSchedule runs in time $O(v\rho)$. Thus, the overall running time of ArboricalLinkSchedule is $O(ev\log v + (e + v\log v) + v\rho)$. Thus, we have the following.

Theorem 3.2: For a planar graph G, ArboricalLinkSchedule has a performance guarantee of O(1) and a worst-case time complexity of $O(ev\log v)$.

4) Arbitrary Networks: We use the algorithm Arboricol LinkSchedule given in Section III-C-1). ArboricalLinkSchedule may be applied to any graph, not only ones that are planar. Note that neither in the algorithm nor in the proof of its correctness did we use the "planar" aspect of the graph. However, while the algorithm produces a schedule for any graph, the performance analysis of the previous section (for planar graphs) obviously does not apply. In this section, we consider the performance of that algorithm on arbitrary graphs. Because the analysis is primarily a generalization of that given in the previous section for planar graphs, we provide here only a brief sketch of the necessary changes.

Theorem 3.3: For an arbitrary graph of thickness θ and maximum degree ρ, ArboricalLinkSchedule has a performance guarantee of $O(\theta^2)$ and a running time of $O(ev\log v + v\theta^2\rho)$.

Proof: The primary change in the analysis is that the bound of five on β, the number of neighbors of a vertex with lower labels, no longer applies. Since for planar graphs the bound of five neighbors was essential in proving the O(1) performance bound, this presents something of a problem. What we can, however, show is that for a graph of thickness θ, there exists at least one vertex of degree at most $6\theta - 1$, hence $\beta \leq 6\theta - 1$. Further, the partitioning method of [15] will result in a decomposition of a thickness θ graph into at most 6θ oriented graphs. It follows from these two facts that the entire graph can be colored with $O(\theta^2\rho)$ colors. An examination of the running time in a manner similar to that in Section III-C-3) shows that the running time is $O(ev\log v + v\theta^2\rho)$. For future reference, we note that when θ is small relative to v, the running time is dominated by the decomposition and is $O(ev\log v)$.□

Previous algorithms for link scheduling all have a bound of $O(\rho^2)$ (a formal analysis is done in [7]). Intuitively, it seems that the thickness of a graph is a small quantity compared to the maximum vertex degree—especially for graphs representing radio networks. This fact is confirmed by our experiments. Here, we generated a number of random graphs (as described in Section II-B) over a large number of values for the "size" (number of vertices) and the transmission range of each of the vertices. A portion of these results is shown in Table I. It can be seen that although both the maximum degree and the thickness[9] increase as the graphs become denser, the thickness increases much more slowly. For all of the graphs generated (including a great number not shown), θ^2 was considerably less than ρ. It follows that our algorithm provides a considerable performance improvement over existing methods, even for arbitrary networks.

TABLE I
COMPARISON OF MAXIMUM DEGREE AND THICKNESS

Nodes	Range	Maxdeg	Thick	% plnr
200	20	9	1	99
200	30	17	2	95
200	40	24	2	84
200	50	34	3	64
400	20	16	2	96
400	30	28	3	79
400	40	43	4	53
400	50	61	5	33

We observe that while the performance is proportional to the square of the thickness, we never actually need to compute the thickness itself. This facet of the algorithm is especially significant since, as mentioned earlier, it is NP-complete to compute the thickness of an arbitrary graph.

A final interesting observation is illustrated in the last column in Table I, which shows the percentage of edges that are present in a maximally planar component of the input graph. As we can see, many graphs are "almost" planar. Further investigation (not documented here) shows the existence of a few isolated edge "cliques" (contributing to nonplanarity) and a very large planar component.

D. Experimental Results

In this section, we provide an experimental analysis of the performance of LinkSchedule in comparison with existing heuristics for the problem of link scheduling. We begin with brief descriptions of the heuristics that were studied.

1) Pure Greedy (PG): This is a straightforward algorithm in which an edge is chosen at random and colored with the first available, nonconflicting color. Most of the algorithms described in previous works are essentially this method.

2) Extended Maximum Degree First Ordering (EMDF) Algorithm: In this method, we take a *maximal* mutually conflicting clique of edges around the maximum degree vertex first, color it, and then progressively do the same for the remainder of the graph. This method is based on the intuitive notion that it is better to color the more "crowded" areas first.

[9] Since determining the thickness of a graph is itself an NP-complete problem [22], this is an estimate of thickness based on the number of trees into which the graph may be decomposed.

TABLE II
COMPARISON OF LINK SCHEDULING HEURISTICS

Nodes	Range	PG	EMDF	ARBORIC
200	20	22	22	22
200	30	46	46	46
200	40	91	90	87
200	50	154	151	143
400	20	50	49	49
400	30	114	111	107
400	40	233	225	215
400	50	436	426	391

Heuristics with this philosophy were first examined in [23] for vertex coloring and were found to do quite well.

3) The Algorithm ArboricalLinkSchedule (ARBORIC): This is an implementation of the algorithm of Section III-C-1). In this implementation, we do not use the method of [15] to partition the graph into oriented graphs but rather we use breadth first search, progressively, to produce an oriented graph partitioning. Using this partitioning method, the running time of the entire algorithm becomes $O(v\theta\rho)$ (recall that the partitioning method of [15] dominated the running time cited previously). This is typically considerably less than the $O(ev\log v + v\theta\rho)$ time required by [15] and is considerably more suitable for practical use.

The performance of these heuristics (number of slots used for scheduling) is shown in Table II. Each entry is averaged over thirty different random graphs with the same parameters. As seen in that table, ArboricalLinkSchedule generally performs the best, followed by the maximal clique first algorithm, and lastly by the commonly used Pure Greedy algorithm. On average, the ArboricalLinkSchedule algorithm uses roughly 8% fewer slots than the Pure Greedy algorithm. The difference in performance widens as we tend toward higher ranges and a higher population (higher densities). Note that ArboricalLinkSchedule does significantly better (about 10%) for a population of 400 vertices, each with a range of 50. In the context of the fact that schedules are set up once and used repeatedly many times over, even a small reduction in the number of slots used is worthwhile since it is amplified by the amount of time the schedule is in operation.

Finally, we note that we have conducted a great many more experiments than those described here. These additional experiments considered both a wider variety of algorithms and additional values of S and R. It is notable that in no circumstance did any algorithm have an average performance better than that of ArboricalLinkSchedule. The *high consistency* of these results certainly gives enhanced credibility to the value of ArboricalLinkSchedule.

IV. BROADCAST SCHEDULING

In this section, we study broadcast scheduling. Recall that we construct a broadcast schedule by coloring the *vertices* of the graph such that two vertices must be given different colors if they either are adjacent or have a common out-neighbor. In this section, we shall consider only graphs with bidirectional edges. Given this and that we color *vertices* and *not* edges, we can model the network as an undirected graph. Thus, the problem of broadcast scheduling is to color an input undirected graph $G = (V, E)$ so that u, v are colored the same only if

1) $(u, v) \notin E$.
2) There is no w such that $(u, w) \in$ and $(v, w) \in E$.

As mentioned in Section I, finding an optimal broadcast schedule is NP-complete. Actually, the following stronger result is shown in [20], [27].

Fact 4.1: For a planar graph $G = (V, E)$, finding an optimal broadcast schedule using seven colors is NP-complete.

However, similar to the previous section, we can find an optimal algorithm for broadcast scheduling of tree networks,[10] and a suitable approximation algorithm for planar and arbitrary networks. In particular, we give here an algorithm for finding a coloring in a arbitrary graph, and show that the algorithm has a performance guarantee of O(1) for planar graphs and a performance guarantee of $O(\theta)$ for an arbitrary graph of thickness θ.

A. The Algorithm

The algorithm BroadcastSchedule given below is very similar to the algorithm given in Section III-C-1) for link scheduling of oriented graphs. The main difference is that here we are concerned with vertex conflicts (primary and secondary) rather than edge conflicts. Note that in broadcast scheduling, the colors that cannot be used to color v are those of its one-hop neighbors and of its two-hop neighbors (see Fig. 5).

Algorithm BroadcastSchedule
input: A graph $G = (V, E)$
output: A coloring $c : V \rightarrow \{1, 2, 3, \cdots\}$.
Phase 1:
  $n \leftarrow$ labeler(G)
Phase 2:
  for $j \leftarrow 1$ to n do
    let u be such that $L(u) = j$
  $c(u) \leftarrow$ NonConflictingVertex(u)

The function NonConflictingVertex simply collects all of the vertices conflicting with the input vertex and returns the first color not in this set. It is described next.

function NonConflictingVertex(u)
  Conflicting $\leftarrow$ $\{c(x) : x$ is colored and is a neighbor of $u\} \cup$ $\{c(x) : x$ is colored and is two-hops away from $u\}$
  return the least color $\notin$ Conflicting

In the above, the call to labeler is actually a call to a modified version of labeler in which an arbitrary undirected graph G is passed rather than an oriented tree with interference edges. The necessary changes to labeler are straightforward and are omitted here. The function NonConflictingVertex may be de-

[10] This algorithm is reasonably straighforward and is omitted here.

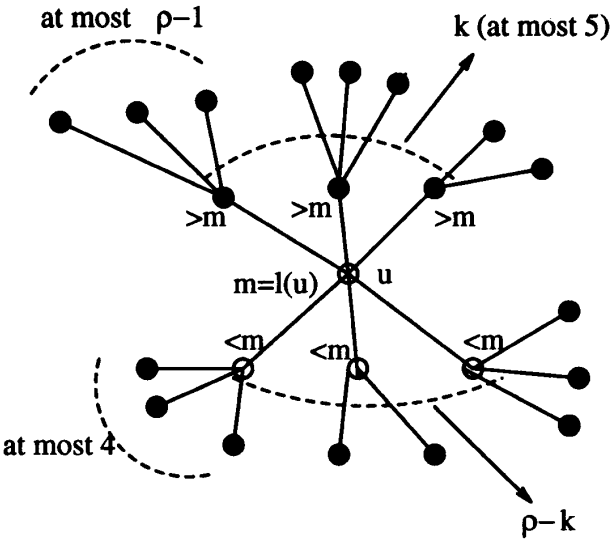

Fig. 5. Neighborhood of the vertex to be colored in Broadcast Schedule.

fined in a manner similar to the function NonConflictingEdge defined for edges in Section III-A—instead of considering primary/secondary *edge* conflicts, we consider *vertex* conflicts. Note also that to broadcast schedule an arbitrary graph, only a single pass is required rather than the series of passes (one for each oriented subgraph) that was needed in the case of link scheduling.

B. Analysis

It is straightforward to show that the algorithm produces a legal coloring. As far as the performance guarantees and the running times are concerned, we have the following.

Theorem 4.1: For a planar graph G, BroadcastSchedule has a performance guarantee of O(1) and a worst-case time complexity of $O(v\rho)$.

Proof: First we discuss the performance and then consider the time complexity of algorithm BroadcastSchedule.

Performance: We begin by noting that the number of colors n used by the algorithm is no more than the maximum color returned by the function NonConflictingVertex, over all vertices.

Without loss of generality, let n be the largest color used and u be a vertex colored n. We show that n is $O(\rho)$ by carefully examining the vertices that affect the color of u. Clearly, such vertices are one-hop or two-hop away from u (see Fig. 5).

Consider a neighbor x_i of u. Then, either $L(x_i) > L(u)$ or $L(x_i) < L(u)$. Suppose that there are k vertices x_i^1 such that $L(x_i^1) < L(u)$ and $\rho - k$ vertices x_i^2 such that $L(x_i^2) > L(u)$. Note that since the labeling algorithm is the same as the one used for Link Scheduling, Lemma 3.5 applies and therefore $k \leq 5$.

Each of the x_i^1 may have at most $\rho - 1$ neighbors (not including u itself) and, hence, the x_i^1 and their neighbors may utilize at most $k(\rho - 1) + k$ or $k\rho$ colors that may not be assigned to u.

We now turn our attention to x_i^2. There are $\rho - k$ such vertices. However, none of them are colored since $L(x_i^2) > L(u)$ (recall that we do the coloring in the increasing order of vertex labels). Nevertheless, we still have to consider the neighbors of x_i^2, some of which may have lower labels than u and hence may be colored. Such vertices would, however, then also have smaller labels than x_i^2 and, by Lemma 3.5, there can be at most $4(\rho - k)$ of them (four for each vertex; note that u itself is a neighbor of x_i^2 having a smaller label).

Thus, u can be colored using no more than $k\rho + 4(\rho - k) + 1$ colors, $(k \leq 5)$, which is at most $9\rho - 19$. Since the optimum is at least $\rho + 1$, the performance guarantee of **BroadcastSchedule** is O(1).

Running Time: The running time is clearly dominated by the running time of NonConflictingVertex. Since at most 5ρ one-hop or two-hop neighbors are colored, coloring a vertex requires processing (updating and searching) a list of at most 5ρ vertices. This is done for each vertex and, therefore, the running time of BroadcastSchedule is $O(v\rho)$.□

Theorem 4.2: For an arbitrary graph G, BroadcastSchedule has a performance guarantee of $O(\theta)$ and a worst-case time complexity of $O(v\theta\rho)$.

Proof: As in the case of link scheduling, the primary change in the analysis involves the observation of the fact that while the bound of 5 on the vertex degree is no longer true (Lemma 3.5), the bound of $6\theta - 1$ holds. Proceeding in a manner analogous to Theorem 4.1, the performance guarantee of BroadcastSchedule for a graph of thickness θ is $O(\theta)$ and the running time is $O(v\theta\rho)$.

C. Experimental Results

We first describe briefly the various heuristics for the problem of BroadcastScheduling. The performance of these heuristics (number of slots used for scheduling) is shown in Table III. Each entry is averaged over thirty different random graphs with the same parameters.

1) *Pure Greedy (PG):* This is a straightforward algorithm in which a vertex is chosen at random and colored in a greedy fashion with the first available, nonconflicting color. Despite its simplicity, it performs quite well. Most of the algorithms described in previous works [28], [11] are essentially this method.
2) *The Maximum Degree First (MDF) Algorithm:* This is an extension of the technique used in [23] for vertex coloring and selects the vertices in decreasing order of their degrees for coloring. The coloring is done "greedily" that is, by using the least color not used by one-hop or two-hop neighbors.
3) *Our algorithm BroadcastSchedule (Bsch):* This is a direct implementation of the algorithm given in Section IV-A.

From the results tabulated in Table III, we see that BroadcastSchedule performs the best of all (more than 10% better than the commonly used Pure Greedy algorithm), followed by the Maximum Degree First algorithm, and then the Pure Greedy algorithm.

TABLE III
COMPARISON OF BROADCAST SCHEDULING HEURISTICS

Nodes	Range	PG	MDF	BSch
200	20	14	14	13
200	30	27	26	24
200	40	43	42	39
200	50	57	56	52
400	20	25	25	23
400	30	53	52	46
400	40	83	83	73
400	50	112	111	101

We observe here, as in the case of link scheduling, that the gap widens as we tend toward higher densities. From more detailed studies that cannot be presented here for lack of space, we observed that for *none* of the over 30 different (S, R) values did the Pure Greedy algorithm do better on average than BroadcastSchedule.

V. CONCLUSIONS

The problems of link and broadcast scheduling for multihop broadcast networks were studied for both arbitrary and restricted networks. New algorithms were given for each case. The performance of our algorithms is superior to existing ones, both theoretically and experimentally. Specifically, the notion of the *thickness* (θ) of a graph was used to analyze the performance. It was shown that, in the worst case, our algorithms have performance guarantees of $O(\theta^2)$ for link scheduling and $O(\theta)$ for broadcast scheduling. These represent significant theoretical improvements over existing algorithms [which have performance quarantees of $O(\rho)$], since θ is typically much smaller than ρ. In each case, the explicit calculation of the thickness itself was not a requirement. A realistic experimental modeling showed that the algorithms described in this paper used, on the average, roughly 8% (10%) fewer slots than did existing link scheduling (broadcast scheduling) algorithms. Since schedules are typically constructed only once and then used for as long as the network is "up," these improvements in performance translate into the savings of precious bandwidth, especially under heavily loaded conditions.

REFERENCES

[1] N. Alon, A. Bar-Noy, N. Linial, and D. Peleg, "On the complexity of radio communication," in *Proc. Twenty First Ann. ACM Symp. Theory of Comput.*, 1989, pp. 274–285.

[2] E. Arikan, "Some complexity results about packet radio networks," *IEEE Trans. Inform. Theory*, vol. IT-30, pp. 910–918, July 1984.

[3] D. Bertsekas and R. Gallager, *Data Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[4] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*. New York: American Elsevier, 1976.

[5] B. Bollobas, *Random Graphs*. London: Academic, 1985.

[6] R. Bar-Yehuda, A. Israeli, and A. Itai, "Multiple communication in multi-hop radio networks," in *Proc. Eighth Ann. ACM Symp. Princ. Distrib. Comput.*, 1989, pp. 329–338.

[7] I. Chlamtac and S. Lerner, "A link allocation protocl for mobile multihop radio networks," in *Proc. GLOBECOM*, Dec. 1985.

[8] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. New York: McGraw-Hill, 1990.

[9] I. Chlamtac and S. Kutten, "A spatial reuse tdma/fdma for mobile multi-hop radio networks," in *INFOCOM Conf. Proc.*, Mar. 1985.

[10] S. Even, O. Goldreich, S. Moran, and P. Tong, "On the np-completeness of certain network testing problems," *Networks*, vol. 14, pp. 1–24, 1984.

[11] A. Ephremedis and T. Truong, "A distributed algorithm for efficient and interference free broadcasting in radio networks," in *Proc. INFOCOM*, 1988.

[12] A. Ephremedis, J. E. Wieselthier, and D. J. Baker, "A design concept for reliable mobile radio networks with frequency hopping signalling," *Proc. IEEE*, vol. 75, no. 1, pp. 56–73, Jan. 1987.

[13] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *J. ACM*, vol. 34, no. 3, pp. 596–615, 1987.

[14] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.

[15] H. N. Gabow and H. H. Westermann, "Forests, frames and games: Algorithms for matroid sums and applications," in *Proc. 20th Ann. ACM Symp. Theory of Comp.*, 1988, pp. 407–421.

[16] L. Hu, "Reliability analysis of sparse topologies for packet radio net works," in *Proc. INFOCOM*, 1992.

[17] D. S. Johnson, "The NP-completeness column," *J. Algorithms* (appears periodically from 1981).

[18] E. L. Lawler, *Combinatorial Optimization*. New York: Holt, Reinhardt and Winston, 1976.

[19] B. M. Leiner, D. L. Nielson, and F. A. Tobagi, "Issues in packet radio network design," *Proc. IEEE*, vol. 75, no. 1, Jan. 1987.

[20] E. L. Lloyd and S. Ramanathan, "On the complexity of distance-2 coloring," in *Proc. 4th Int. Conf. Comput. and Inform.*, May 1992.

[21] E. L. Lloyd and S. Ramanathan, "On the complexity of link scheduling in multi-hop radio networks," in *Proc. 26th Conf. Inform. Sci. and Syst.*, Mar 1992.

[22] A. Mansfield, "Determining the thickness of graphs is np-hard," *Math. Proc. Cambridge Philos. Soc.*, vol. 93, pp. 9–23, 1983.

[23] D. W. Matula, G. Marble, and J. F. Issacson, "Graph coloring algorithms," in *Graph Theory and Computing*. New York: Academic, 1972.

[24] C. Nash-Williams, "Edge-disjoint spanning trees of finite graphs," *J. London Math. Soc.*, vol. 36, pp. 213–228, 1961.

[25] R. Ogier, "A decomposition method for optimal scheduling," in *Proc. 24th Allerton Conf.*, Oct 1986.

[26] C. G. Prohazka, "Decoupling link scheduling constraints in multi-hop packet radio networks," *IEEE Trans.Comput.*,vol.38,no.3, pp. 455–458, Mar. 1989.

[27] S. Ramanathan and E. L. Lloyd, "Complexity of certain graph coloring problems with applications to radio networks," Tech. Rep. 92-18, Dept. Comput. Sci., Univ. Delaware, 1992.

[28] R. Ramaswami and K. K. Parhi, "Distributed scheduling of broadcasts in a radio network," in *Proc. INFOCOM*, 1989.

[29] A. S. Tanenbaum, *Computer Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[30] A. Wigderson, "Improving the performance guarantee for approximate graph coloring," *J. ACM*, vol. 30, no. 4, pp. 729–735, Oct. 1983.

Subramanian Ramanathan, photograph and biography not available at the time of publication.

Errol L. Lloyd, photograph and biography not available at the time of publication.