A Topology-Independent Fair Queueing Model in Ad Hoc Wireless Networks

Haiyun Luo, Songwu Lu
UCLA Computer Science Department
Los Angeles, CA 90095-1596
email: {hluo, slu}@cs.ucla.edu

Abstract

Fair queueing of rate and delay-sensitive packet flows in a shared-medium, multihop wireless network remains largely unaddressed because of the unique design issues such as location-dependent contention, spatial channel reuse, conflicts between ensuring fairness and maximizing channel utilization, and distributed fair scheduling. In this paper, we propose a new topology-independent fair queueing model for a shared-medium ad hoc network. Our model ensures coordinated fair channel access among spatially contending flows while seeking to maximize spatial channel reuse. We describe packetized algorithms that realize the fluid fairness model with analytically provable performance bounds. We further design distributed implementations that approximate the ideal centralized algorithm. We evaluate our design through both simulations and analysis.

1. Introduction

Fair queueing has been a popular paradigm for providing fairness, minimum throughput guarantees and bounded delay access for packet flows in wireline networks [1, 2] and packet cellular networks [3, 4]. However, the problem of fair packet scheduling in a shared-medium, multihop wireless network has remained largely unaddressed. Adapting fair queueing to such a network is challenging because of the unique issues such as location-dependent contention among transmitting flows, spatial channel reuse through concurrent flow transmissions in a partially connected network, location-dependent channel error, and the distributed nature of packet scheduling in ad hoc networks. In this work, we first propose a precisely quantifiable definition of fairness and then describe a packetized algorithm to achieve such fairness model.

In wireline networks, a popular model for packet scheduling over a communication link is the fluid fair queueing (FFQ) model [1, 2]. In this model, packet flows are modeled as fluid flows through a channel of capacity C, and every flow f is assigned a weight r_f; over any infinitesimally small window of time $[t, t + \Delta t]$, a backlogged flow f is allocated a channel capacity of $C \cdot \Delta t \cdot (r_f / \sum_{i \in \mathcal{B}(t)} r_i)$, where $\mathcal{B}(t)$ is the set of backlogged flows at time t. Packet scheduling algorithms such as WFQ and SCFQ seek to serve packets in an order that approximates FFQ as closely as possible while maintaining low implementation complexity. Wireless fair queueing in packet cellular networks [3, 4] further addresses the issue of location-dependent and bursty channel error; the idea is to allow for leading flows (that lead ahead of their error-free reference services) to gracefully give up their leads in order to let lagging flows (that lag behind their error-free services due to channel errors) catch up their lags, thus ensuring fair channel access for both types of flows over a larger time interval. At first glance, it would seem that the wireless fair queueing model and algorithms developed for packet cellular networks are equally applicable or can be readily extended to shared-medium ad hoc networks. However, there are at least three characteristics of ad hoc networks that render this fair queueing paradigm inapplicable:

- *Location-dependent contention*: In an ad hoc network, wireless transmissions are locally broadcast in the shared physical channel. The locality of wireless transmissions implies that channel contention is location-dependent. Nodes within the transmission range of an ongoing conversation are typically restrained not to transmit in order to avoid collisions. In wireline or packet cellular networks, the packet scheduler implemented at each network node/switch only needs to consider flows that share the link, and scheduling decisions over each link are performed independently. However, in ad hoc networks, packet scheduler at each local switch has to arbitrate spatial contentions between its flows and other flow transmissions in its spatial locality, in order to ensure *coordinated fair channel access* among contending flows (e.g. flows F_1, F_2, F_3 in Figure 1) that contend both in the time domain and in the spatial domain.

- *Spatial channel reuse*: The multihop wireless nature of ad hoc networks makes spatial channel reuse pos-

sible: for a given flow, other flows may transmit simultaneously if they are out of the transmission range of the current flow and are not interfering with each other (e.g. flows F_1 and F_4 in Figure 1). Note that, however, channel reuse is a spatial property and the amount of spatial reuse is dynamically changing depending upon which flows are transmitting at the moment. How to share the wireless medium fairly while maximizing spatial channel reuse poses another design challenge for fair queueing.

- *Conflicts between ensuring fairness and maximizing channel utilization*: Maximizing channel utilization in an ad hoc network may show preference for certain flows and starve other flows, thus violating the fair sharing principle. In a generic-topology ad hoc network, the two goals of ensuring fairness and maximizing channel utilization may be in conflict.

In this work, we seek to address the above issues. The three contributions of this paper are: (a) a topology-independent fairness model that ensures coordinated fair channel access among spatially contending flows while seeking to maximize spatial channel reuse, (b) design of a packetized fair queueing algorithm that realizes the fairness model with analytically provable performance bounds, and (c) distributed implementations of the ad hoc fair queueing algorithm. We show the effectiveness of our design through both simulations and analysis.

The rest of the paper is organized as follows. Section 2 describes the network model and identifies the key design issues. Section 3 proposes a model for fair queueing in an ad hoc network. Section 4 presents a packetized algorithm that realizes the fairness model and analyzes its properties. Section 5 describes distributed implementations of the packetized algorithm. Section 6 presents a simulation-based performance evaluation of the proposed design. Section 7 describes related works, and Section 8 concludes the paper.

2. Network Model and Design Issues

2.1. Network Model

We consider a packet-switched multihop wireless network in which a single physical channel with capacity C is available for wireless transmissions. Transmissions are locally broadcast and only receivers within the transmission range of a sender can receive its packets. Each link-layer packet flow is a stream of packets being transmitted from the source to the destination, where the source and destination are neighbors. We define two flows as *contending flows* if either the sender or the receiver of one flow is within the transmission range of the sender or the receiver of the other

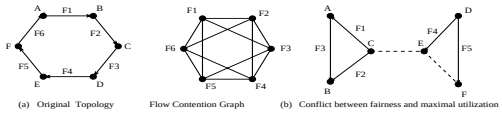

Figure 1. Spatial Contention and Reuse

flow[1] [5].

We make three assumptions [9, 8, 5, 7]: (a) neighborhood is a commutative property and hence flow contention is also commutative, (b) a node cannot transmit and receive packets simultaneously, and (c) a collision occurs when a receiver is in the reception range of two simultaneously transmitting nodes, thus unable to cleanly receive signal from either of them; we ignore capture effect in this paper. We do not explicitly consider mobility and non-collision-related channel errors in this paper.

2.2. Design Issues in Ad Hoc Fair Queueing

This section identifies two issues unique to fair queueing in ad hoc wireless networks.

2.2.1 Fairness for spatially contending flows

In a wireline or packet cellular network, packet scheduler implemented at each link needs to consider flows that are contending for the link only. Fluid fairness defined for such networks is, in essence, a *local* property for transmitting flows over each link and fair queueing algorithms ensure *local fairness* in the time domain among contending flows that share a single link.

Location-dependent contention in a shared-medium ad hoc network implies that fairness model cannot be defined with respect to "local" flows in a node only, and has to embrace spatial transmission constraints. Ideally, we would still like to preserve the local fairness property inherent in wireline fair queueing model as much as possible, while addressing the issue of spatial contention among flows.

An additional issue is to define a fairness model that takes into account spatial channel reuse. This is nontrivial since spatial reuse is *flow dependent*: for certain flows, no spatial reuse is possible; for other flows, multiple concurrent flow transmissions can take place.

2.2.2 Conflicts between fairness and maximal channel utilization

In fair queueing over ad hoc networks, we seek to accomplish two goals: (a) ensure fairness among contending flows; this is the design tenet of every fair queueing algorithm; and (b) maximize spatial channel reuse thus increase the overall effective capacity. Ideally, we would like to achieve both goals at the same time. However, this is not always possible in a generic-topology ad hoc network.

[1] Following the CSMA/CA medium access paradigm, we assume that data transmission will be preceded by a control handshake. Thus the nodes in the neighborhood of both the sender and the receiver must defer transmission to ensure a successful handshake.

Consider the five-flow example shown in Figure 1.b. The system capacity will be $2C$ if we let F_3 and F_5 transmit all the time. However, it is easy to verify that the total effective capacity will be less than $2C$ if all five flows have to transmit and get a fair share. This example illustrates the fundamental conflict between achieving flow fairness and maximizing overall system throughput.

3. A New Fair Queueing Model

In this section, we will develop a topology-independent fairness model for ad hoc networks. In the model to be developed, the granularity of each packet is a bit, and each flow f is assigned a weight r_f. The goal is to define a bit-by-bit fair queueing model such that each packet flow receives a fair share in proportional to its weight r_f while addressing the design issues identified in Section 2.2. We assume a perfect knowledge on the network topology and flow information at the moment.

Our proposed model seeks to provide the *maximal fair share to each flow according to each flow's flow weight*. That is, given a flow's weight, the flow will receive a maximum fair share of bandwidth in proportional to the flow's weight. In the following, we describe the proposed fairness model through a step-by-step procedure. Our model ensures fair channel access while seeking to minimize the total expected transmission times via spatial reuse, thus maximizing overall effective throughput of the network.

3.1. Generating a flow contention graph

We first convert packet flows in a network graph into a *flow contention graph*, which characterizes the contention relationship among flows (see Figure 1 for an example). In a flow graph, each vertex represents a backlogged flow, and an edge between two vertex denotes two contending flows. If two vertices are not connected, these two flows can transmit simultaneously, thus making spatial reuse possible.

3.2. Choosing appropriate flow sets to define a fair queueing model

Fair queueing is typically defined with respect to a set of contending flows that compete for limited shared resources. In wireline networks, the definition of a contending flow set is straightforward: it consists of flows contending for an output link. In an ad hoc network, the spatial correlation among transmitting flows complicates the choice of contending flow set for fair queueing definition.

In this work, we seek to preserve the "local" fairness property of fair queueing while addressing the issue of spatial contention between flows: we partition the flows in the network into multiple *partially contending flow sets*, and a fair queueing model is defined with respect to each flow set. *Each partially contending set is defined as a closed partially connected flow graph.*

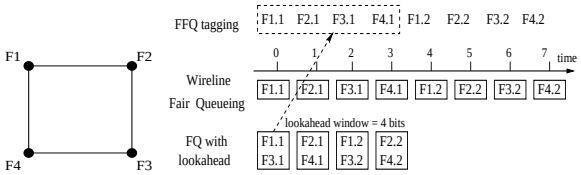

Figure 2. Lookahead FQ

Two properties of a partially contending flow set follow immediately: (a) partially contending flow sets are disconnected/decoupled from each other and flow scheduling can be performed in each set independently; (b) flows in a partially contending flow set are not necessarily mutually contending, thus spatial reuse is possible in each set.

3.3. Resolving the conflicts between fairness and maximal spatial reuse

Since flows in a partially contending flow set may not be mutually contending, spatial reuse is possible within the flow set. The issue to address is how to resolve the conflicts between ensuring fairness and maximizing spatial reuse in a generic-topology ad hoc network. Two extreme points in the solution space (see [13] for a detailed discussion on the solution space) are: (a) maximize channel reuse as much as possible even at the cost of violating fairness; (b) ensure fairness, and maximize spatial reuse subject to the fairness constraint. In this work, according to fair sharing principle, we choose option (b), and place ensuring fairness as our primary design tenet whereas maximizing spatial reuse as secondary.

3.4. Seeking to maximize spatial channel reuse

Therefore, we seek to maximize spatial reuse subject to the fairness criterion. To this end, an intuitive model is to impose a *maximality* property upon any fair allocation policy, i.e., a fair channel allocation in which no further spatial reuse assignment can be made without violating the fairness constraint. However, realizing this model is nontrivial as illustrated by the example in Figure 2.

In the example, we have four flows with equal weights. The scheduling precedence based on a standard wireline FFQ algorithm is shown in Figure 2. Hence, in wireline fair queueing, this round of bits are transmitted in the strict order of $F1.1, F2.1, F3.1, F4.1$, where $Ff.p$ denotes the p-th bit of flow f. Now let us look at ad hoc networks. We start by transmitting $F1.1$. Since flows F_1 and F_3 can transmit simultaneously according to the flow graph, we concurrently transmit both bits $F1.1$ and $F3.1$ in order to increase spatial reuse[2]. However, in wireline fair queueing, we should transmit $F2.1$ before $F3.1$ since it has a smaller finish tag. Transmitting $F3.1$ before $F2.1$ violates the transmission precedence, as well as instantaneous

[2]If we do so, a simple calculation reveals that we can achieve total throughput of $2C$. Otherwise, if we transmit according to the strict order $F1.1, F2.1, F3.1, F4.1$, no spatial reuse is possible and we only have C.

fairness constraint, imposed by the standard wireline fair queueing.

Lookahead window to enable spatial reuse Therefore, in order to enable spatial reuse, we have to *locally swap the transmission orders* so that flow F_3 can transmit its $F3.1$ bit before flow F_2 sends $F2.1$. This motivates a "lookahead window" for ad hoc fair queueing, in order to make spatial reuse possible in a generic topology ad hoc network. At any time instant, the scheduler schedule bits within a lookahead window ρ (defined in virtual time), and may locally switch the transmission order for the bits within the window. Basically, a lookahead window of ρ enables the scheduler to look ahead in virtual time by ρ rounds, and may swap the service order of packets in the window in order to increase spatial reuse.

In general, the choice of a large ρ tends to increase spatial reuse to certain degree. However, this is achieved at the cost of violating instantaneous fairness. Therefore, we bound the value of ρ to ensure long-term fairness.

Maximizing spatial reuse within a static lookahead window is a minimum graph coloring problem If the scheduler allows for local swapping of transmission orders within a lookahead window, then maximizing spatial reuse is equivalent to transmitting the full window of packets in minimum time. This is equivalent to a minimum graph coloring problem: let each vertex denote a flow bit and each edge in the flow graph denote a spatial contention between the two nodes, assign a color to each vertex such that the colors on the pair of nodes incident to any edge be different (thus avoid spatial collisions), the goal is to find the minimum number of colors for the graph.

Sliding window for ad hoc fair queueing However, it should be noted that maximizing spatial reuse in ad hoc fair queueing is not equivalent to a *static* minimum coloring problem. Fair queueing is a dynamic scheduling paradigm. Whenever the scheduler schedules/transmits a bit (or several bits due to spatial reuse) within the current lookahead window, it will move its window ahead. As a consequence, we arrive at a dynamic sliding window problem. This brings several new design issues. First, we now have a dynamic minimum coloring problem since the number of bits within the lookahead window at any instant may be dynamically changing (in fact, some bits within the window range may not be present at the moment since they have been transmitted in advance via spatial reuse). Second, in general, minimizing transmission times of packets in the current window tends to maximize overall expected throughput but they are not equivalent. Third, we have to balance two design goals: (a) transmit the current window of bits as soon as possible, and (b) move the window as fast as possible so that more new bits can move into the lookahead window.

3.5. A fair queueing model for ad hoc networks

A brief summary of our design decisions so far is the following: (a) We partition flows in the network into multiple independent "partially contending flow sets." We define a fluid fair queueing model for flows within each set. (b) The fluid queueing model is specified as follows: Each flow receives its weighted fair services, which come from both the basic physical channel and spatial reuse. We seek to maximize spatial reuse subject to the fairness constraint. (c) In order to enable and increase spatial reuse, at any time instant, the scheduler schedules bits (from different packet flows) within a lookahead window ρ, and may locally swap the transmission orders for bits within the window. Bounding the window ρ ensures fairness over a larger time interval. (d) In order to maximize spatial reuse, we seek to minimize the transmission times of packets within the current lookahead window. This is equivalent to a dynamic minimum graph coloring problem as the window moves forward after each transmission.

4. A Packetized Fair Queueing Algorithm

We now describe an idealized ad hoc fair queueing algorithm that realizes the fluid fairness model of Section 3. The algorithm is idealized because we assume perfect knowledge of the per-flow information. For simplicity of presentation, we assume that all packets are of the same size L_p, and that each packet is transmitted in one slot. Comments on variable packet size are given in [14].

4.1. Algorithm description

Our packetized algorithm is composed of two key components: a basic scheduling loop and an adaptive dynamic coloring algorithm.

4.1.1 The basic scheduling algorithm

An overview of the basic scheduling is the following:

1. We simulate weighted fair queueing (WFQ) to assign two tags for each arriving packet: a start tag and a finish tag. Specifically, a packet with sequence number n of flow i arriving at time $A(t_{i,n})$ is assigned two tags: a start tag $s_{i,n}$ and a finish tag $f_{i,n}$, defined as follows:

   $$s_{i,n} = \max\{v(A(t_{i,n})), f_{i,n-1}\}; \quad f_{i,n} = s_{i,n} + L_p/r_i$$

   where $v(A(t))$ is derived from the FFQ: $dv/dt = C/\sum_{i\in\mathcal{B}(t)} r_i$, C is the channel capacity and $B(t)$ is the set of backlogged flows at time t.

2. The scheduler maintains a lookahead window ρ of packets to make scheduling decisions for each slot, with ρ being set to be $\rho = \max_{f\in F}\lceil\frac{L_p}{r_f}\rceil$, where $\lceil\cdot\rceil$ is the ceiling function that rounds a float-point value to its closest integer upper bound.

3. The virtual time V(t) of the ad hoc scheduler at t is set to be the start tag of the packet that has the *smallest* finish tag among packets which are being served at t.

4. The actions in the scheduling loop are the following:

   (a) for all the packets with start tag being in the range of $[V(t), V(t)+\rho]$, apply an adaptive coloring algorithm to partition flow packets into $m(t)$ disjoint sets, denoted by $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{m(t)}$, such that within each set concurrent transmissions are possible. The dynamic coloring algorithm, to be described in next section, seeks to form a number of disjoint sets as small as possible.

   (b) among flows with start tag within $[V(t), V(t)+\rho]$, pick the flow f with the least finish tag, transmit the head-of-line packets of f and all flows in the disjoint set $\mathcal{C}_f$ that flow f belongs to.

   (c) update $V(t)$ according to Step (3) and move the sliding window forward to the next packet that has the smallest finish tag.

Several remarks on the algorithm are available: (a) Setting $\rho = \max_{f\in F}\lceil\frac{L_p}{r_f}\rceil$ allows each flow to transmit at least one full packet by the end of the window. (b) Since all flows have identical packet size L_p, as long as the physical channel capacity C is fixed, all concurrent flows can complete their packet transmissions in the shared wireless medium simultaneously. (c) The virtual time $V(t)$ typically moves much faster than t due to spatial reuse, and the sliding window $[V(t), V(t)+\rho]$ may leap forward for multiple slots after each slot transmission.

In the packetized algorithm, we have two goals to achieve: (a) minimize the total number of disjoint sets $m(t)$, thus minimize the total expected transmission times for the current window; (b) move the window forward as fast as possible. Our algorithm seeks to strike a balance: upon every slot transmission, the window moves forward at least one slot; meanwhile, the number of disjoint sets will not increase when window moves.

4.1.2 An adaptive algorithm to the dynamic graph coloring problem

We now present an adaptive solution to the following dynamic graph coloring problem: at any time t, consider n flows and each flow f has $k_f(t)$ packets in the window $[V(t), V(t)+\rho]$, how to transmit $\sum_f k_f(t)$ in minimal time subject to the spatial contention topology specified by the flow graph.

Our algorithm makes use of two facts: (a) Consider the problem setting of Section 4.1.1, within any window $[V(t), V(t)+\rho]$, each flow f has at most $\lceil\frac{\rho}{r_f L_p}\rceil$ packets. (b) Dynamic graph coloring for two consecutive transmissions is correlated in general. Some flows are present in both windows; hence, these two windows have overlaps. This implies that a recursive solution to the dynamic coloring problem is possible.

The detailed algorithm works as follows:

1. At $t = 0$, use an approximation algorithm to solve the following static minimum coloring problem: Let each vertex denote a flow packet and each edge denote a contention between the two vertices in the graph, consider $k_f(0) = \lceil\frac{\rho}{r_f L_p}\rceil$ vertices for flow f, assign a color i to each vertex such that the colors on the pair of nodes incident to any edge be different (thus avoiding contentions), the goal is to find the minimum number of colors m for the graph.

   The solution partitions the graph into $m(0)$ disjoint sets, denoted by $\mathcal{C}_1$, $\mathcal{C}_2$, ..., $\mathcal{C}_{m(0)}$, and each flow f is associated with $k_f(0)$ sets. A straightforward approximation is the so-called greedy algorithm [10].

2. At time $(t-1)$, denote the $m(t-1)$ disjoint sets as $\mathcal{C}_1$, $\mathcal{C}_2$, ..., $\mathcal{C}_{m(t-1)}$. Each set $\mathcal{C}_k$ has w_k flows, i.e., $\mathcal{C}_k =: \{f_1^k, \ldots, f_{w_k}^k\}$.

3. At time t, the sliding window moves forward to $[V(t), V(t)+\rho]$ and some new packets move into the window. We take the following actions for packets in $[V(t), V(t)+\rho]$:

   (a) For any unserved packets that come from the previous window $[V(t-1), V(t-1)+\rho]$, retain the disjoint sets that these unserved packets belong to, say, $\mathcal{C}_1$, $\mathcal{C}_2$, ..., $\mathcal{C}_{m'(t)}$. Obviously, $m'(t) \le m(t-1)$.

   (b) For each of the newly joined packets, denoted by p_f from flow f,

      i. merge p_f with one of the existing $m'(t)$ disjoint sets. If p_f is not contending with any flow in the set $\mathcal{C}_k$, then it declares a "join success," p_f joins set $\mathcal{C}_k$ and records the set ID of $\mathcal{C}_k$. In this step, p_f performs contention checks with every flow in the set $\mathcal{C}_k =: \{f_1^k, \ldots, f_{w_k}^k\}$, by assuming that *all* these w_k flows had packets, waiting to be served, in the set.

      ii. If no merge if possible, create a new set $\mathcal{C}_{m'(t)+1}$, and p_f joins the new set. Now, the existing sets are the following: $\mathcal{C}_1$, $\mathcal{C}_2$, ..., $\mathcal{C}_{m'(t)}$, $\mathcal{C}_{m'(t)+1}$. Then update the new number of sets as $m(t) = m'(t) + 1$.

   (c) retain all the disjoint sets $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{m(t)}$ until $t+1$.

There are several features regarding the above algorithm: (a) As time evolves, the total number of disjoint sets (i.e.,

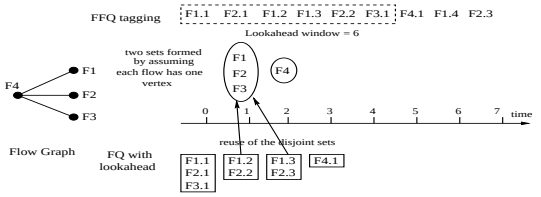

Figure 3. Reusing the basic disjoint sets

the number of colors) will never increase. We will show this property in Section 4.2. (b) Our algorithm is self adaptive in the sense that at each time instant, it seeks to reduce the total number of disjoint sets from the last time instant. This is important when we may only have a poor approximation algorithm in the initial step. (c) In Step 3.(b).i, the assumption of all flows are present in the set C_k is crucial in the join test for p_f. (d) In Step 3.(b).i, the join test for p_f should start with the set C_k which has the smallest finish tag for packets in the current window $[V(t), V(t)+\rho]$. This way, it helps to move the window forward faster.

Several optimization techniques are available that further improve the algorithms of this section; the details are provided in [14]. We omit them here due to lack of space.

4.2. Analytical properties

In this section, we characterize the analytical performance of the above algorithm. The proofs are given in [14].

Fairness guarantee

The following theorem states that short-term inter-flow fairness is mainly governed by the lookahead parameter ρ. This result is quite intuitive since the service differences between any two flows in the ideal fluid fair queueing model are upper bounded by ρ.

Theorem 4.1 *(Short-term Fairness Property) Consider a fixed flow set $\mathcal{F}$ in which n flows remain backlogged during $[t_1, t_2]$. Let $W_f(t_1, t_2)$ denotes the service (in bits) that flow f receives in the lookahead packet fair queueing (LPFQ) algorithm of Section 4.1 during $[t_1, t_2]$. Then the difference in the service received by two flows f and m at a LPFQ server is given as:*

$$\left| \frac{W_f(t_1,t_2)}{r_f} - \frac{W_m(t_1,t_2)}{r_m} \right| \leq \rho + \frac{L_p}{r_m} + \frac{L_p}{r_f}\,, \qquad (1)$$

where ρ is the lookahead parameter in bits.

The following theorem shows the long-term fairness property: as t becomes large, the effect of short-term fair service deviation ρ diminishes.

Theorem 4.2 *In the problem setting of Theorem 4.1, let $\Delta t = t_2 - t_1$, then the following holds:*

$$\lim_{\Delta t \to \infty} \frac{W_f(t_1, t_1+\Delta t)}{W_m(t_1, t_1+\Delta t)} = \frac{r_f}{r_m}\,. \qquad (2)$$

Properties of the adaptive coloring algorithm

In the following theorem, we will show that the total number of disjoint sets will never increase as time t evolves. This guarantees that the transmission times for any sliding window of packets will never increase. Therefore, it tends to decrease the total number of transmissions thus increasing spatial reuse.

Theorem 4.3 *(Non-increasing total number of disjoint sets) Consider n flows, and each of which contributes exactly one vertex in the graph coloring problem. Define the total number of sets at t, denoted as $\mathcal{N}(t)$, as the number of disjoint sets assuming all n flows were present at t. Then $\mathcal{N}(t)$ is non-increasing as t evolves. That is, for any given t, the following holds: $\mathcal{N}(t) \leq \mathcal{N}(t-1)$.*

Spatial reuse

The following corollary states that the total expected transmission times for current window are not increasing with t. This result follows readily from Theorem 4.3.

Corollary 4.1 *(Non-increasing expected transmission times for any sliding window) In the setting of Theorem 4.3, since flows in each disjoint set can be concurrently transmitted in one slot, the worst-case total expected transmission times for the current window, by assuming that the window is full, are never increasing.*

Throughput and delay bounds

First note that each backlogged flow will always receive a basic fair service by assuming that no spatial reuse were available. That is, each flow receives at least a fair share from the basic physical channel capacity C. Then both the long-term throughput and packet delay bounds, developed for a standard WFQ scheduler [2] hold in our case. We have not included them here due to lack of space. Short-term throughput can differ by ρ because a flow may have received services in advance, which is upper bounded by ρ, due to concurrent transmissions in the previous time interval. In addition to the basic services, each flow may receive additional fair services due to spatial reuse.

5. Distributed Implementation

5.1. Two design issues

Distributed fair queueing In an ad hoc network, contending flows originate from different sending nodes, and no single logical entity for scheduling of these flows is available. Besides, the flow information is "distributed" among these sending nodes, and each sender does not have direct access to other flows' information at other senders.

Information propagation in a broadcast medium In order to achieve distributed fair queueing, we have to propagate certain flow information (e.g., flow weights), *in minimum time*, to the entire network graph. In a network that

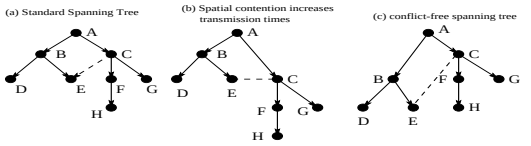

Figure 4. Information propagation along a spanning tree with spatial contention

has point-to-point links, the optimal solution to propagate information from a given node to *all* the rest nodes of the network in minimum time is to build up a shared, minimum-height spanning tree. However, in a multihop wireless network, the wireless medium is a local broadcast channel, and there are potential collisions for packet transmissions in a spatial locality. As a result, propagating information along a minimum-height spanning tree may not be optimal any more! Unlike in a point-to-point link medium, sibling nodes (located at the same level) in the tree may not be able to concurrently transmit in a broadcast medium due to spatial contention. This effectively increases the total propagation time needed to propagate information to all the nodes along the tree.

Consider the example shown in Figure 4. Figure 4.(a) shows the standard spanning tree, and in a network with point-to-point links, the transmission times to propagate information from root A to all the rest nodes will be 3 units. However, since both B and C are within range of E (shown by the dotted line in the figure), in order to propagate to all the nodes, sibling nodes B and C cannot transmit concurrently to their children (otherwise, E perceives collisions). Hence, A has to transmit to B and C sequentially, and it takes 4 units to reach to all nodes, as shown in Figure 4.(b). However, if we construct the tree as in Figure 4.(c), we only need 3 units to propagate information from A to all nodes.

5.2. Propagating information along a conflict-free shared tree

In order to address the above issue, we seek to build up a core-based shared tree that provides minimum time transmissions from the core node to all other nodes in the tree and ensures conflict-free concurrent delivery for sibling nodes at the same height of the tree.

We start with a standard core-based shared spanning tree; this can be achieved by constructing the spanning tree for each node using the breadth-first search algorithm, and selecting the minimum-height spanning tree from these trees. Given the spanning tree, we resolve collisions among sibling transmitting nodes through delaying packet transmissions along some branches of the tree (see Figure 4(c) for an example). For this purpose, each transmitting node maintains a delay counter C_d, which records the delay time for the packet transmissions in its branch.

We use a backoff-based mechanism to construct a conflict-free shared tree. We start from the nodes closest to the core node. Every transmitting node senses the channel and waits for a backoff number of minislots before initiating its RTS-DATA multicast message (no CTS or ACK is used due to its multicast nature). We set the backoff value to to be the difference between the height of the tree and the height of a node's current branch. This way, we give priorities to the branches with larger height, and may delay the transmissions of other shorter branches in the presence of potential collisions. When a node hears either RTS or collisions, it increments its delay counter C_d by 1, thus delaying transmissions along its branch. At the receiver side, a single receiver may be within the transmission range of multiple transmitters. Whenever it hears a collision, it broadcasts a NACK message to the senders. Upon receiving the NACK message, the transmitting nodes will randomly decide whether to increase their delay counter C_d by one or not. More details are given in [14].

5.3. Implementation I: A link-state alike approach

In our first implementation, each node runs a local copy of the packet scheduler described in Section 4.1. To this end, the packet scheduler needs to have access to the following information: (a) the flow graph that a flow belongs to, (b) a flow's weight r_f, and (c) which flow is backlogged at time t and packet arrival times. Among these three types of information, (a) and (b) can be propagated just once when the flow was established, but the last type of information may be dynamically changing and has to be updated online. However, we do not expect that flows change backlogged status very often in a typical scenario; even if a flow's arrival pattern may be quite bursty, the bandwidth-constrained wireless channel will further amortize packet arrivals and make flows remain backlogged most of the time.

In order to propagate the flow information to each node, we adopt a two-logical channel approach by dividing the physical channel into two logical channels, one for control and the other for data transmissions. We use the control channel to propagate the above information along a conflict-free multicast tree to be described next, and the data channel to forward data packets. The control and data channels are logically distinct, but share the same physical channel.

5.4. Implementation II: A backoff-based approach

In our second implementation, we present a backoff-based distributed algorithm within the framework of CSMA/CA MAC paradigm.

In our design, we use weighted round robin to approximate the fair queueing algorithm. To this end, we normalize the flow weight r_f for each flow f so that the smallest flow weight is normalized as one. We seek to schedule r_f packets for each flow f within a minimum cycle period as time evolves. This is equivalent to a dynamic graph coloring problem, that is, consider a flow graph that can be colored with α colors at t. If we define a cycle of α slots, then each

flow will be able to transmit at least r_f slots in the cycle. We use the largest-degree first (LF) algorithm [10] in the coloring approximation.

Decide cycle length α We use the shared tree of Section 5.2 to propagate the cycle size. In the bootstrapping period, the core node will estimate the total number of colors needed to color the flow contending graph. For example, set $\alpha = (1+\varepsilon)\Delta$ where Δ is an estimation of the maximum flow degree and ε is a parameter in 0.03–0.20 [11]. The core multicasts this initial α to each flow. Each flow then contends for r_f slots based on the *LF* approximation, and reports to the core the total number of slots it uses to successfully contend for r_f slots. The core adjusts the cycle size accordingly based on the feedback and multicasts a new cycle size again to every sender. If every flow can be settled in the previous cycle, the core shrinks the cycle size. If some flows could not get r_f slots, the core increases the cycle size. After several rounds, the network will converge to a cycle size. Note that the core of the shared tree does not maintain any per-flow information during the process.

Individual flow scheduling in a cycle In a CSMA/CA MAC protocol, there is an RTS-CTS handshake before each packet transmission. Each transmitting node senses the carrier before sending out its RTS, and this is preceded by a backoff of certain number of minislots. In our design, we set the backoff value to approximate the LF algorithm. To this end, we give higher priorities to flows with larger flow degrees by setting their backoff periods shorter. In the bootstrapping period, each flow will receive two parameters from the shared tree: cycle size α and the core's estimation of the maximum flow degree Δ. Starting from the first slot in a cycle, each flow contends with backoff $\Delta - d$ minislots (where d is its degree). When the backoff timer expires, if the sender senses the carrier idle, it sends out its RTS and occupies the slot in the cycle. Once a flow grabs a certain slot in a cycle, it will contend for that slot with backoff zero in future. Each flow repeats contending for each slot in the cycle until it grabs r_f slots.

When neighboring flows have the same flow degree, collisions may happen. Our collision resolution works as follows: once a flow experiences collision in a cycle, it set its backoff value to be a random number in $[1, \Delta - d]$ in the next coming cycle. Note that in the next cycle, flows that have already successfully grabbed slots in the previous cycles will contend for their "reserved" slots with zero backoff value, flows with non-zero backoff values will not induce any collision in the occupied slots.

Idle flows In the above contending process, for each individual flow, all the slots before its occupied last slot in a cycle will be grabbed by either its neighbors or itself. This means a flow will always be able to hear transmissions in all these slots before it finishes its transmissions in a cycle. Once a flow becomes idle, its neighboring flows will be able to detect an idle slot in the current cycle. Therefore, they may contend for that slot using backoffs in the next cycle. If some of them grab that slot successfully, it will report to the root to initiate possible shrinking of the cycle size. If the idle flow does not remain idle for more than a cycle period, it can reclaim its old slots by contending with backoff zero. If it remains idle for more than one cycle and is backlogged again, it will be treated as a new flow.

New flows joining and old flows leaving When a new flow joins, it will try to fit itself into the cycle by contending with backoffs. If it could not fit into the current cycle size, it reports to the core node of the shared tree. The core will then increase the cycle size and let new flows contend. When an existing flow terminates, other flows treat it as an idle flow and use the procedure described above to contend for its idle slots.

6 Simulations

In this section, we evaluate our design of Sections 4 and 5 by simulations. We provide three simulation examples. Example 1 illustrates the features of ensuring fairness and increasing spatial reuse, and how idle flows may affect the optimal spatial reuse. Example 2 shows that our algorithm achieves both short-term and long-term fairness, throughput and packet delay bounds. Example 3 evaluates 17 flows in a more complicated scenario, shows the convergence property of the adaptive coloring algorithm, and demonstrates the fair sharing of spatial reuse among these flows.

Several performance measures are used to evaluate the algorithm. W_l : number of transmitted packets of the flow during the simulation lifetime; W_s : number of transmitted packets of the flow during a small time interval; D_{avg} : average delay of transmitted packets; $D_{\max}$: maximum delay of transmitted packets. σ_D : standard deviation of the delay.

Each of our simulations has a typical run of $100,000$ time units. To obtain measurements over short time windows, we measured the performance over 10 different time windows, each of which has 200 time units, and averaged the results for the value shown here. In all cases, we assume that the physical channel capacity C is one slot per time unit.

We consider three types of source traffic in our simulations: constant rate, Poisson and Markov-modulated Poisson process (MMPP). For the MMPP sources, the modulated process is a two-state Markov chain. The transition rate from ON to OFF is 0.9 and OFF to ON is 0.1.

Example 1 In this example, we evaluate the fairness property of our idealized algorithm of Section 4, and its efficiency in achieving spatial reuse of bandwidth. We also include the simulation results using the distributed implementation of Section 5 for Scenario (c). We consider seven

flows with equal weights $r_f = 1$, and the flow graph for the simulation is shown in Figure 5. We evaluate the following three scenarios:

(a) In this scenario, we set flows to be infinite sources, i.e., each flow remains continually backlogged all the time. The throughput result for each flow is shown in Table 1. As expected, the algorithm achieves both long-term and short-term fairness and throughput guarantees. In this example, it achieves 233% of total effective throughput (the throughput based on a physical channel capacity C is defined to be 100%). It can be easily verified that 233% is the optimal throughput under fairness constraint. However, we do point out that, the maximal throughput can be 300%, and is achieved by concurrent transmissions of flows F_0, F_3, F_5. In this case, other flows have to be starved in order to sustain this amount of maximal utilization. This illustrates the fundamental conflicts between fairness and maximal throughput in a generic network topology.

Figure 6 shows the performance of the adaptive coloring algorithm. In this case, the algorithm quickly (within several time units) reaches a total number of three disjoint sets. The figure also shows that the number of active sets, each of which has at least an unserved flow, vary from one to three within each window.

(b) Now we show the throughput and fairness performances when Flow F_3 becomes idle after $t = 50,000$ in the setting of Scenario (a). The results are given in Table 2. We can see from the table, our algorithm achieves a total throughput of 200% during the time period of $[50000,\ 100000]$, which reaches the optimal fair spatial reuse. In this case, we can easily verify that maximal system throughput (even without the fairness constraint) is also 200%. The criteria of fairness and maximal throughput are not in conflict for this scenario.

(c) Now, we change packet arrivals: Flows F_0, F_1, F_4 are Poisson, Flows F_2, F_5, F_6 are MMPP, and Flow F_3 is constant rate. We set the source parameters such that all have an average arrival rate of 0.23. From the results shown in Table 3, we can see that these seven flows achieve long-term and short-term fairness and throughput guarantees. Since MMPP traffic is more bursty, the maximum packet delay of MMPP sources, as well as average delay, is higher than other sources.

(d) In the setting of Scenario (c), we further simulate the distributed implementation algorithm, and the results are given in Table 4. We can see that the throughput decreases slightly to 155% (from 161%), and the average delay of flows has increased in the distributed implementation.

Example 2 In this example, we further test the fairness property as well as spatial reuse efficiency of our algorithm in the scenario shown in Figures 7 and 8. In Figure 7, flows F_1, F_3, F_8 are MMPP sources, and other six flows are Poisson sources; all of them have an average arrival rate of 0.3.

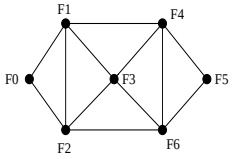

Figure 5. Example 1

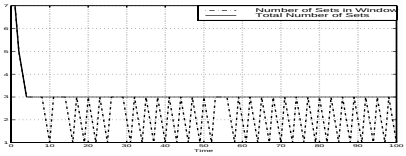

Figure 6. Performance of adaptive coloring algorithm

Flow	W_l	W_s
0	33333	67
1	33333	66
2	33334	67
3	33333	67
4	33334	66
5	33333	66
6	33333	66

Table 1. Ex. 1: Scenario (a)

Flow	$W_l(1)$	$W_l(2)$	W_s
0	16667	16667	67
1	16667	16667	66
2	16666	16666	67
3	16667	-	-
4	16666	16666	66
5	16666	16667	66
6	16667	16667	66

Table 2. Ex. 1: Scenario (b)

All flows are backlogged until $t = 50000$, then flows F_3 and F_6 become idle until $t = 100000$. The throughput, fairness and delay results are given in Table 5. In this scenario, if all flows are backlogged (see Figure 7), the maximal throughput under fairness constraint is 300%; after flows F_3 and F_6 become idle, the maximal throughput under fairness constraint actually increases to 350%. However, since the total arrival rates for the rest seven flows are only 210%, these flows cannot use up the extra capacity. This illustrates the importance of *adaptive flows* in the ad hoc scenario. Ideally, flows should be adaptive such that they can adjust their transmission rates depending upon network bandwidth availability.

Example 3 In this example, we test a more involved scenario, in which we have 17 flows in the graph (shown in Figure 9). The 17 flows have different weights and Poisson arrival rates (shown in Table 6). The long-term and short-term throughput and fairness, average delay and its standard deviation, and maximum delay using the idealized algorithm of Section 4 are shown in Table 6. We can see that all flows receive their services close to their weighted fair shares (the randomness in the packet arrivals contributes to the deviation) in both short term and long term. In this case, the actual channel utilization is 313% of the physical channel capacity.

Flow	W_l	W_s	D_{avg}	D_{max}	σ_D
0	23029	44	0.2	16	0.95
1	23021	46	3.4	26	3.26
2	23041	43	38.1	220	37.6
3	22994	47	0.02	8	0.21
4	23025	46	2.3	38	3.89
5	23022	45	33.5	218	37.8
6	23018	48	38.5	221	37.4

Table 3. Example 1: Scenario (c)

Flow	W_l	W_s	D_{avg}	D_{max}	σ_D
0	21743	41	1.2	23	2.5
1	21732	43	5.8	42	34
2	22568	40	55	230	42
3	21641	42	1.3	19	7
4	21845	41	5.4	41	12
5	22548	42	50	229	38
6	22566	40	48	233	41

Table 4. Ex.1: Distributed implementation

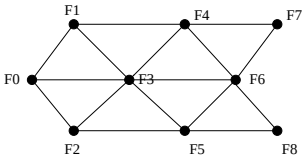

Figure 7. Flow graph of Example 2: all backlogged.

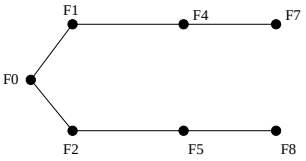

Figure 8. Flow graph of Example 2: F_3, F_6 are idle.

In the same setting, we also give the results using the distributed implementation in Table 7. Again the system throughput decreases slightly to 300% (compared to 313%), each flow receives service approximately in proportional to its flow weight, and the average packet delay has increased compared to the ideal centralized algorithm.

In Figures 10, 11 and 12, we also show the convergence feature of our adaptive coloring algorithm. In the case of infinite packet arrivals for all flows, it takes less than 10 time units to reach a stable sets of 4 as shown in Figure 10; if the flows have Poisson packet arrivals, it takes about 75 time units to finally reach the total sets of 4 using the idealized algorithm of Section 4. This shows that more aggressive packet arrivals provide more opportunities for the adaptive algorithm to converge. In Figure 12, we also show that the distributed implementation of the adaptive coloring algorithm performs similar to the infinite arrivals even for Poisson sources. The reason is that in the distributed implementation, the flows will not update their status immediately when a backlogged flow becomes idle; this helps the adaptive coloring algorithm to converge.

Flow	W_l	W_s	D_{avg}	W'_l	W_s	D_{avg}
0	15014	62	4.8	15017	59	1.4
1	14926	57	13.5	14931	60	4
2	15013	60	3.7	15025	62	0.44
3	14922	55	11.5	-	-	-
4	15009	59	7.9	15027	59	1.44
5	15011	58	5.0	15019	59	1.44
6	15019	62	5.2	-	-	-
7	15014	64	3.6	15022	63	0.44
8	14927	59	12.5	14963	61	4

Table 5. Performance of Example 2: Left part: all backlogged; Right part: F_3, F_6 idle

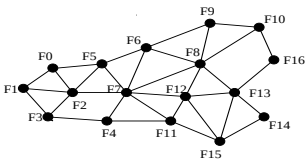

Figure 9. Flow graph of Ex. 3

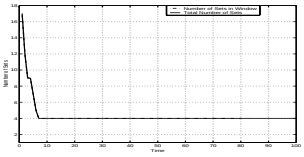

Figure 10. Ex. 3: Infinite arrivals

Flow	r_f	arrival rates	W_l	W_s	D_{avg}	$D_{\max}$	σ_D
0	3	0.39	39174	78	0.8	22	1.5
1	1	0.13	13051	25	1.7	24	1.8
2	1	0.13	13046	26	2.7	32	2.9
3	1	0.13	13053	29	0.7	16	0.9
4	2	0.26	26123	56	1.0	29	1.4
5	1	0.13	13049	24	2.9	38	3.4
6	1	0.13	13051	26	1.6	28	1.7
7	2	0.26	26126	55	0.8	18	1.3
8	1	0.13	13044	28	3.9	37	3.7
9	1	0.13	13045	29	0.8	16	1.0
10	1	0.13	13048	23	1.3	21	1.4
11	1	0.13	13039	25	2.2	25	2.6
12	2	0.26	26129	53	1.1	31	2.6
13	1	0.13	13042	26	3.2	36	3.6
14	1	0.13	13040	24	0.8	29	0.9
15	1	0.13	13044	24	1.2	30	1.7
16	3	0.39	39173	76	0.5	12	0.8

Table 6. Performance of the Idealized Algorithm in Example 3: 17 Poisson flows

7. Related Work

Fair packet scheduling has been the subject of intensive study in the networking literature and numerous algorithms have been proposed since the weighted fair queueing algorithm was first presented in [1],[2]. In recent years, there are several research efforts on adapting fair queueing to wireless cellular networks in order to handle location-dependent channel errors [3], [4].

While wireline fair queueing and wireless fair queueing in packet cellular networks have been actively researched, fair queueing in a shared-medium ad hoc network is a relatively unchartered territory. In ad hoc networks, providing minimum throughput guarantees and bounded delay access has been studied at the MAC layer [7, 9]. A popular approach has been to establish transmission schedules and allocate stations to different time slots of a TDMA cycle in a

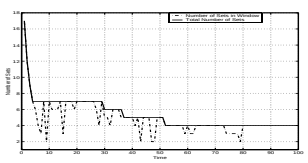

Figure 11. Example 3: idealized algorithm

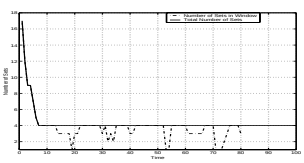

Figure 12. Example 3: distributed implementation

Flow	r_f	arrival rates	W_l	W_s	D_{avg}	$D_{\max}$	σ_D
0	3	0.39	38708	75	8.2	51	7.5
1	1	0.13	12264	23	8.9	41	7.3
2	1	0.13	12326	23	8.8	44	7.2
3	1	0.13	12010	22	7.5	41	7.4
4	2	0.26	25403	50	7.6	45	7.0
5	1	0.13	12324	24	9.3	45	7.8
6	1	0.13	12205	21	7.8	42	6.4
7	2	0.26	25630	49	9.0	51	7.7
8	1	0.13	12461	23	10.2	41	7.9
9	1	0.13	12002	22	7.4	42	7.2
10	1	0.13	12105	24	8.3	46	7.8
11	1	0.13	12382	22	9.5	41	7.5
12	2	0.26	25625	52	8.8	47	7.5
13	1	0.13	12269	21	8.7	42	7.1
14	1	0.13	11901	23	6.8	39	6.9
15	1	0.13	12186	24	8.3	44	7.3
16	3	0.39	38509	74	6.9	51	7.2

Table 7. Performance of Example 3: 17 Poisson flows, Distributed implementation

way that no collisions occur. The design goal is to design conflict-free link scheduling schemes that seek to maximize the spatial reuse of the bandwidth and remain immune to topological changes in a mobile ad hoc networking environment. Another study [8] also investigates the fair link activation problem in such a network. However, all these previous studies seek to provide throughput guarantees or weighted fairness for wireless *links*, not for packet *flows*; hence, they do not address the problem of fair scheduling of packet flows. Besides, the focus of these MAC-layer studies has been on the mechanisms of channel access by assuming that the packet scheduling algorithm has been worked out, rather than the other way around.

Another recent work [13] also addresses the problem of packet scheduling in multihip wireless networks. However, the focus there is to resolve the conflicts between fairness and maximal channel utilization, and the solution proposed does not provide fair service to each flow. When we prepare the camera-ready version of this paper, we are aware of another independent work of [12], which also seeks to address the issue of fair queueing in multihop wireless networks. However, the authors have not addressed the issue of maximizing spatial reuse subject to fairness constraint. Besides, both the design and the implementation are significantly different from ours.

8. Conclusion

Fair queueing in a shared-medium ad hoc network is an important emerging area of wireless networking research, because simple best-effort scheduling of flows is inadequate in scarce and heavily loaded wireless medium. In this work, we formulate the problem and propose an initial solution. While our proposed solution addresses the problem to certain extent and the results obtained so far are quite encouraging, we readily admit that many issues such as user mobility and interaction with the underlying MAC protocol are not fully or explicitly addressed in the current work. We expect to explore these issues in the future work.

References

[1] A. Demers, S. Keshav and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *ACM SIGCOMM'89*, August 1989.

[2] A. Parekh, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks," *PhD Thesis*, MIT Laboratory for Information and Decision Systems, Technical Report LIDS-TR-2089, 1992.

[3] S. Lu, V. Bharghavan and R. Srikant, "Fair scheduling in wireless packet networks," *IEEE Trans. Networking*, August 1999.

[4] T.S. Ng, I. Stoica and H. Zhang, "Packet fair queueing algorithms for wireless networks with location-dependent errors," *IEEE INFOCOM'98*, March 1998.

[5] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, "MACAW: A medium access protocol for wireless LANs," *ACM SIGCOMM'94*, 1994.

[6] "IEEE 802.11 Standard Specification," 1997.

[7] Z. Tang and J.J. Garacia-Luna-Aceves, "A protocol for topology-dependent transmission scheduling in wireless networks," *WCNC'99*, September 1999.

[8] I. Chlamtac and A. Lerner, "Fair algorithms for maximal link activation in multihop radio networks," *IEEE Trans. Communications*, 35(7), July 1987.

[9] J. Ju and V.O.K. Li, "An optimal topology-transparent scheduling method in multihop packet radio networks," *IEEE Trans. Networking*, 6(3), June 1998.

[10] D. Welsh and M. Powell, "An upper bound to the chromatic number of a graph and its application to timetalbling problems," *Computer Journal*, Vol. 10, No. 1.

[11] M.V. Marathe, A. Panconesi, and L. Risinger Jr., "An experimental study of a simple distributed edge coloring algorithm," 1999.

[12] N. H. Vaidya, P. Bahl and S. Gupta, "Distributed fair scheduling in a wireless LAN," *MOBICOM'2000*, 2000.

[13] H. Luo, S. Lu and V. Bharghavan, "A new model for packet scheduling in multihop wireless networks," *MOBICOM'2000*, 2000.

[14] H. Luo and S. Lu, "Fair queueing in ad hoc wireless networks," *Technical Report*, UCLA, Januarary 2000.