Fair Scheduling in Wireless Packet Networks

Songwu Lu Vaduvur Bharghavan Rayadurgam Srikant
Coordinated Sciences Laboratory
University of Illinois at Urbana-Champaign
slu@crhc.uiuc.edu; bharghav@crhc.uiuc.edu; rsrikant@uiuc.edu

Abstract

Fair scheduling of delay and rate-sensitive packet flows over a wireless channel is not addressed effectively by most contemporary wireline fair scheduling algorithms because of two unique characteristics of wireless media: (a) bursty channel errors, and (b) location-dependent channel capacity and errors. Besides, in packet cellular networks, the base station typically performs the task of packet scheduling for both downlink and uplink flows in a cell; however a base station has only a limited knowledge of the arrival processes of uplink flows.

In this paper, we propose a new model for wireless fair scheduling based on an adaptation of fluid fair queueing to handle location-dependent error bursts. We describe an ideal wireless fair scheduling algorithm which provides a packetized implementation of the fluid model while assuming full knowledge of the current channel conditions. For this algorithm, we derive the worst-case throughput and delay bounds. Finally, we describe a practical wireless scheduling algorithm which approximates the ideal algorithm. Through simulations, we show that the algorithm achieves the desirable properties identified in the wireless fluid fair queueing model.

1 Introduction

Future indoor and outdoor packet cellular environments will seek to support communication-intensive applications such as multimedia teleconferencing, WWW browsing, etc. Supporting such applications requires the network to provide sustained quality of service to packet flows over scarce and shared wireless networking resources. In wireline networks, quality of service requirements are typically satisfied by a combination of resource reservation (at the flow level) and fair resource allocation/packet scheduling (at the packet level). However, in packet cellular environments, user mobility and wireless channel error make it very difficult to perform either resource reservation or fair packet scheduling. While there have been some recent efforts to provide resource reservation for mobile flows in packet cellular networks, the problem of fair packet scheduling in wireless networks has remained largely unaddressed. In fact, even the notion of fairness in shared channel wireless networks has not been precisely defined. In this work, we first propose a precisely quantifiable definition of fairness and then describe algorithms for packet scheduling in wireless networks to achieve such fairness.

In wireline networks, a popular model for packet scheduling over a link is the fluid fair queueing model [5, 11]. 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 Δt, a backlogged flow f is allocated a channel capacity of $C.\Delta t.(r_f / \sum_{i \in B(t)} r_i)$, where B(t) is the set of flows that are backlogged at time t. There are several packet-level algorithmic implementations of this model, such as WFQ [5], $\mathrm{WF}^2\mathrm{Q}$ [1], SCFQ [7], STFQ [8], etc. Essentially, the goal of each of these algorithms is to serve packets in an order that approximates fluid fair queueing as closely as possible. At first glance, it would seem that the fluid fair queueing model is applicable to scheduling over a wireless channel, and any of the above algorithms will work just as well for wireless channels. However, there are two key characteristics of shared wireless channels which render the fluid fair queueing model inapplicable: (a) *bursty channel errors*, and (b) *location-dependent channel capacity and errors*. Since wireless transmissions are locally broadcast, contention and effective channel capacity are location-dependent. Besides, due to interference, fades and multipath effects, channel errors are also location-dependent. This implies that at any time, only a subset of flows can be scheduled on the channel. All the algorithms cited above for wireline scheduling assume that the channel is error-free, or at least that either all flows can be scheduled or none of them can be scheduled. In fact, the fluid fair queueing model itself does not address the issue of fairness when a subset of backlogged flows are unable to transmit due to channel errors. This points to a clear need to develop a new fairness model for wireless channels.

Another unique concern in packet cellular networks is the following: within a cell, every host is only guaranteed to be within the range of the base station, and all transmissions are either uplink or downlink. Thus, the base station is the only logical choice for the scheduling entity in a cell. However, the base station has only limited knowledge about the arrival of packets in uplink flows. In particular, the base station may know if an uplink flow has any outstanding packets, but it does not know when packets arrive, or how many packets there are in the queue at any instant. Thus we cannot assume convenient tagging mechanisms at the source when packets arrive, as in most fair queueing algorithms. Besides, due to the shared nature of the wireless medium, we must take into account hidden/exposed stations [3]. This points to a close coordination of the scheduling algorithm with the medium access protocol for wireless channels.

In this paper, we seek to address the issues unique to wireless fair queueing, while drawing extensively from the basic fluid fair queueing model for wireline networks. The key contributions of this paper are the following: (a) a fairness model for wireless fair queueing, (b) the design of an ideal wireless fair queueing algorithm with analytically provable delay and throughput bounds, and (c) the design of a wireless scheduling algorithm which closely approximates the characteristics of the ideal algorithm, while addressing several practical issues in wireless medium access.

The rest of the paper is organized as follows. Section 2 describes the wireless network model. Section 3 proposes a model for wireless fluid fair queueing. Section 4 describes a wireless scheduling algorithm which realizes the fairness model for an idealized scenario, while Section 5 derives its analytical throughput and delay bounds. Section 6 discusses some implementation issues, while Section 7 describes the practical wireless packet scheduling algorithm. Section 8 provides a simulation based performance evaluation of this algorithm. Section 9 compares our work with related work, and Section 10 concludes the paper.

2 Model

For the purposes of this paper, we consider a packet cellular network with a high-speed wired backbone and small, partially overlapping, shared channel wireless cells. Each cell is served by a base station, which performs the scheduling of packet transmissions for the cell. Neighboring cells are assumed to transmit on different logical channels. All transmissions are either uplink (from a mobile host to a base station) or downlink (from a base station to a mobile host). Every mobile host in a cell can communicate with the base station, though it is not required for any two mobile hosts to be within range of each other. Each flow of packets is identified by a <host, uplink/downlink flag, id> triple. We say that a flow 'perceives a channel error' (i.e. sees a 'bad' channel state) if either the source or the destination of the flow experiences an error burst in its locality. Error patterns are assumed to vary depending on location; we do not make any explicit assumption about the error model, though our simulations typically use a two-state Markov chain model. By varying the transition probabilities, we generate a range of error patterns in the wireless channels, ranging from highly bursty to Bernoulli distributed errors.

3 Wireless Fluid Fair Queueing

As described in Section 1, the Fluid Fair Queueing model [5] treats each packet flow as a fluid flow. Each flow i is given a weight r_i, and for any time interval $[t_1, t_2]$ during which there is no change in the set of backlogged flows $B(t_1, t_2)$, the channel capacity granted to each flow i, $W_i(t_1, t_2)$, satisfies the following property:

$$\forall i, j \in B(t_1, t_2), \left|\frac{W_i(t_1, t_2)}{r_i} - \frac{W_j(t_1, t_2)}{r_j}\right| = 0. \qquad (1)$$

The above definition of fair queueing is applicable for both channels with constant capacity and channels with time varying capacity. However, it does not address the issue of location-dependent channel error, as shown below:

Consider three backlogged flows with $r_1 = r_2 = r_3 = 1/3$. Flow 1 and flow 2 have error free channels while flow 3 perceives a channel error during the time interval [0,1]. By applying equation (1) over the time periods [0,1] and [1,2], we arrive at the following channel capacity allocation:

$$W_1[0,1] = W_2[0,1] = 1/2, W_1[1,2] = W_2[1,2] = W_3[1,2] = 1/3,$$

Now, over the time window [0,2], the allocation is

$$W_1[0,2] = W_2[0,2] = 5/6, W_3[0,2] = 1/3.$$

which does not satisfy the fairness property of equation (1). This simple example illustrates the difficulty in defining fairness in a wireless network, even in an idealized model. In general, server allocations designed to be fair over one time interval may be inconsistent with fairness over a different time interval.

In the fluid fair queueing model, when a flow has nothing to transmit during a time window $[t, t+\Delta]$, it is not allowed reclaim the channel capacity that would have been allocated to it during $[t, t+\Delta]$ if it were backlogged at t. However, in a wireless channel, it may happen that the flow is backlogged, but unable to transmit due to channel error. In such circumstances, should the flow be compensated at a later time? In other words, should channel error and empty queues be treated the same or differently? In particular, consider the scenario when flows f_1 and f_2 are both backlogged, but f_1 perceives a channel error while f_2 perceives a good channel. In this case, f_2 will additionally receive the share of the channel which would have been granted to f_1 in the error-free case. The question is whether the fairness model should readjust the service granted to f_1 and f_2 in a future time window in order to compensate f_1. The traditional fluid fair queueing model does not need to address this issue since in a wireline model, either all flows are permitted to transmit or none of them is.

In the Virtual Clock model [13], when a flow has nothing to transmit during a time window, it can reclaim its missed share of the channel capacity at a later time. Arguments have been made against allowing for such compensation, including the fact that were the packet flows really fluids flowing through a common pipe, such compensation would not be allowed. However, we argue that the case of empty queues and the case of channel error should be treated differently. From a user-centric perspective, we would like the well-behaved flows to remain oblivious of short error bursts, and react only to prolonged error bursts. In particular, we would like to make use of the fact that some channels may be error free when other channels experience error, and implicitly *swap* the channel allocation over short time windows in order to accommodate short error bursts. However, we would still like to provide separation between flows by bounding the amount of compensation that can be provided to a flow due to its channel error. Essentially, we seek to make a trade-off between the full compensation model and the full separation model in order to provide bounded compensation. Note that our compensation model is fundamentally different from the Virtual Clock approach, since we only compensate if the flow has packets to transmit but is unable to do so because of channel error. In particular, we do not penalize a flow for using the entire channel capacity if no other flow had anything to send.

Given the arrival processes for each of the flows and the error patterns perceived by each of the flows, we define an *error-free service* as the fluid fair queueing service for the flows with identical arrival processes and completely error-free channels. We define a flow to be *lagging* at any time instant if its queue length is greater than the queue length of its error-free service at the same time instant. We define a flow to be *leading* at any time instant if its queue length is less than the queue length of its error-free service at the same time instant. *The key feature of our wireless fluid fairness model is to allow lagging flows to make up their lag by causing leading flows to give up their lead.* By artificially bounding the amount of the lag and lead, we can trade-off between long-term fairness and separation between flows. We now define the bounds on lag and lead for each flow.

1. The aggregate lag of all flows which will be compensated is bounded by a constant B bits. A lagging flow i with weight r_i is allowed to compensate a maximum

of $b_i = B * \frac{r_i}{\sum_{j \in F} r_j}$ bits, where F is the set of all flows.

2. A leading flow is allowed to lead by a maximum of l_i bits. Specifically, even when a flow is ahead of its error-free service by more than l_i bits, it only gives up a channel capacity worth l_i bits to other contending flows.

We now define the wireless fair queueing model. The granularity of transmission of a flow is a bit. Each bit has a *service tag*, which is the virtual time of its error-free service. The service tag of a backlogged flow is the service tag of the first bit in its queue; the service tag of a non-backlogged flow is ∞. The virtual time at any instant is the virtual time of the error-free service at the same instant. Based on the above definitions, the wireless fluid fair queueing server works as follows:

1. The next bit to be transmitted is chosen from the head of the queue of the flow with the minimum service tag among the backlogged flows which perceive a good channel.

2. Each lagging flow i is allowed to retain (at most) the earliest b_i bits with a service tag less than the current virtual time.

3. If a flow i leads its error-free service by more than l_i bits, its service tag is adjusted to reflect a lead of l_i bits.

There are three important points to note regarding the wireless fluid fair queueing model: (a) the flow which has lagged the longest has the lowest service tag, and hence has highest precedence to access the channel, (b) a flow which always perceives an error-free channel may still lag its error-free service by up to B bits because it has to defer for lagging flows with lower service tags, and (c) a flow which leads by more than l_i bits does not have to 'pay' for more than l_i bits; likewise, a flow which lags by more than b_i bits cannot reclaim more than b_i bits.

4 The Idealized Wireless Fair Queueing Algorithm

In this section, we describe an idealized wireless fair queueing (IWFQ) algorithm that realizes the wireless fluid fair queueing model. This algorithm is idealized because it makes two key assumptions: (a) each flow knows whether it can transmit correctly in the current slot (i.e. transmitted packets are never lost in transit), and (b) packets can be tagged as soon as they arrive. For simplicity, we assume that all packets are of the same size L_P, and that each packet is transmitted in one slot.

4.1 Algorithm Description

The overview of the algorithm is the following:

1. We simulate the error-free fluid service for the flows. At any instant, the virtual time for the idealized wireless fair queueing algorithm is the virtual time of the error-free service at the same instant.

2. Each arriving packet is tagged as in the Weighted Fair Queueing algorithm: 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:

$$s_{i,n} = max\{v(A(t_{i,n})), f_{i,n-1}\} \tag{2}$$

$$f_{i,n} = s_{i,n} + L_P/r_i \tag{3}$$

$v(A(t))$ is derived from the error-free service as follows:

$$dv(t)/dt = C/\sum_{i \in B(t)} r_i,$$

where C is the channel capacity in bits/sec and $B(t)$ is the set of backlogged flows at time t in the error-free service.

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

   (a) readjust tags for each flow.

   (b) for each flow, set its *service tag* equal to the finish tag of the head of line packet of the flow. If there is no packet in the queue, set the service tag to ∞.

   (c) among the flows which can transmit (i.e. channel is good), pick the flow with least service tag and transmit its head of line packet. This algorithm adapts the selection process of WFQ. By restricting the packets eligible for selection to only those which would have started their error-free service in the fluid model, we could adapt the selection process of WF^{2}Q.

4. Readjusting tags for a flow involves the following:

   (a) for each lagging flow i, if the number of packets with finish tags less than the virtual time is greater than $B_i = \frac{B.r_i}{\sum_{i \in F} r_i.L_P}$, then retain only the first (lowest tagged) B_i packets in the queue and delete the remaining packets[1].

   (b) for each leading flow i, if the start tag of the head of line packet ($s_{i,hol}$) is greater than the virtual time ($v(t)$) by more than l_i/r_i, then

$$s_{i,hol} = v(t) + l_i/r_i, \quad f_{i,hol} = s_{i,hol} + L_P/r_i \tag{4}$$

The service that a flow receives in the IWFQ algorithm is never behind the service that it would receive in the wireless fluid fair queueing model by more than L_P (for the same reason that the service in WFQ is never behind the service in FFQ by more than L_P [5]). For error-free service, IWFQ and WFQ are identical. When some flows perceive short error-bursts (i.e. neither lags nor leads exceed their bounds), IWFQ performs local adjustments in channel allocation in order to compensate the flows for their channel errors. The goal of IWFQ is thus to approximate WFQ while still accommodating short-term errors. However, IWFQ and WFQ differ in several ways.

A flow which is denied service because of channel error is guaranteed to eventually receive service after its channel becomes good since its service tag does not change. Thus, backlogged flows receive precedence in channel allocation when their channels become error-free.

The separation property of WFQ (which guarantees that the worst case delay for the head-of-line packet of a flow i, $d^{max}_{WFQ} \leq L_P/C + (L_P.\sum_{i \in F} r_i)/(r_i.C)$, is independent of the behavior of other flows) is only valid for IWFQ with the following bounds: $d^{max}_{IWFQ} \leq d^{max}_{WFQ} + B/C$.

Finally, a critical difference between IWFQ and WFQ lies in the way that packets are discarded in each algorithm. In WFQ, packets are discarded if the flow gets backlogged

[1] Note, that an error-free weighted fair queueing service will lag by at most one packet.

by more than its maximum buffer size. In addition to the above, packets may be discarded in IWFQ if a flow lags by more than B_i packets. We provide a mechanism to separate the following two questions: 'how many packets should be discarded', and 'which packets should be discarded'. This mechanism is also useful for handling lost/corrupted packets when we remove the idealized assumption that the channel can always be predicted accurately.

4.2 Slot Queues and Packet Queues

Since a flow i is allowed to lag by at most B_i packets, during prolonged error bursts, it may be forced to discard packets from its queue. If the flow discards packets from the head of the queue, its service tag increases; thus we can no longer preserve the guarantee that a lagging flow eventually has the lowest service tag (and hence has highest precedence to access the channel). On the other hand, for delay-sensitive but loss-tolerent flows, retaining packets which have already waited in the queue for a long time is meaningless. Essentially, *lagging flows should have the ability to discard any packets from a backlogged queue without losing precedence in channel access.* This points to the need to decouple service tags from the packet queues.

In order to achieve such a decoupling, we maintain two queues for each flow: a *slot queue*[2] and a *packet queue*. When a new packet arrives, the following actions are taken: (a) the packet joins the packet queue, (b) a new slot is created and assigned the start tag and finish tag corresponding to the packet, (c) the slot joins the slot queue. At any time, the maximum number of lagging slots is bounded by B_i for a flow i, and slots with the lowest finish tags are retained in order to preserve the precedence of channel access for the lagging flow. The service tag for a flow is the finish tag of the head of line slot in its slot queue.

A separate mechanism deletes packets from the packet queue depending on the requirements of the flow. For example, a packet may be deleted after a fixed number of retransmissions or after a delay bound has expired. When a slot is selected for transmission, the head of line packet in the packet queue is transmitted - thus the mapping between slots and packets is dynamic. By decoupling slot queues from packet queues, we can handle multiple types of delay and loss requirements for flows while still maintaining the precedence in channel access for lagging flows.

5 Throughput and delay guarantees for the IWFQ

The following facts should be obvious from the IWFQ algorithm described in Section 4:

Fact 1 *At any time t, the number of lagging bits of flow i, denoted by $b_i(t)$, satisfies $\sum_{i\in F} b_i(t) \le \sum_{i\in F} B_i L_P = B$, where L_P is the packet length in bits.*

Fact 2 *For any lagging slot s of flow i at time t, its finish tag is no greater than that of any non-lagging slot, i.e. it is served with higher priority.*

The following results regarding error-free fluid fair queueing (FFQ) and error-free WFQ[3] have been proved in [12], and are included here for quick reference.

[2] In this section, slots refer to logical slots rather than physical time slots.

[3] Error-free FFQ and error-free WFQ refer to FFQ and WFQ, respectively, when all the channels are error free.

Lemma 1 *[12] Let $\tilde{S}_i(\tau,t)$ and $S_i^*(\tau,t)$ be the amount of flow i traffic (in bits, not packets) served under the fluid fair queueing and the error-free WFQ in the interval $[\tau,t]$, for all times τ and flows i:*

$$\tilde{S}_i(0,\tau) - S_i^*(0,\tau) \le L_P, \quad S_i^*(0,\tau+\frac{L_P}{C}) \ge \tilde{S}_i(0,\tau) \qquad (5)$$

For all packets p, let $\tilde{F}_p$ and F_p^* be the time at which packet p departs under the fluid fair queueing and the WFQ,

$$F_p^* - \tilde{F}_p \le L_P/C, \qquad (6)$$

where C is the channel capacity (in bits per second) and L_P is the packet length in bits. □

The delay and throughput results in this section are given separately for two types of channels: (a) Error-free Channel: an error-free channel is one which is always in the good state at all time slots, and (b) Error-prone Channel: an error-prone channel is one which is not always in the good state.

5.1 Error-Free Channels

Based on Facts 1 and 2, we can show the following result:

Lemma 2 *Any slot s on a error-free channel g[4] completes its service in IWFQ by time $t_{ef} + \bar{d}_g$, with*

$$\bar{d}_g = B/C. \qquad (7)$$

where t_{ef} is the finish time of slot s in the error-free WFQ algorithm, F denotes the set of all flows, L_P is the packet length (in bits) and C is the channel capacity (in bits per second).

Proof. Let us consider an arbitrary time t, at which slot s is the head-of-line slot for flow g.

If slot s has received service in IWFQ before or at t_{ef}, i.e. flow g is leading at this time, then the result is trivially true. In the following, we only consider the case that slot s receives its service later than t_{ef} in the IWFQ.

Denote the current virtual time as $v(t)$ (i.e. the bit round in progress as defined in equation 4. At virtual time $v(t)$, in the error-free WFQ, let the slot sequence waiting to be served be $m, m+1, \ldots, s-1, s, s+1, \ldots, k$ where slot sequence $0, 1, \ldots, m-1$ has been transmitted by the time t in the error-free WFQ. We also denote the finish tag associated with a slot i as T_i. Let s belong to flow g. The remaining waiting time (after t) in terms of time slots for s under error-free WFQ is $s - m$.

Consider the scenario for the IWFQ algorithm at time t. Let $B(t)$ denote the set of the lagging slots (ordered by their finish tags) by all flows at time t, i.e. when m becomes eligible for service in the slot sequence. Note that the total number of slots that have been transmitted in IWFQ, denoted by q, is no greater than m, i.e. $q \le m$, due to the work-conserving nature of the server. Thus, the slot with the lowest tag in IWFQ is either m or the slot with the lowest finish tag in $B(t)$.

By Fact 2, it follows that any slot in $B(t)$ has no greater finish tag than the sequence $T_m, T_{m+1}, \ldots$. Therefore, the *largest possible* sequence of slots to be served at time t observes the order $B(t)$, $m, m+1, \ldots, s-1, s, s+1, \ldots, k$. Hence, the maximum number of slots (after current time t) to be served before s in the IWFQ is $|B(t)| + s - m$, where $|B(t)|$ is the cardinality of set $B(t)$. Based on Fact 1, it follows that $|B(t)| \le B$. for any time t. Hence, slot s

[4] In this case, the flow is also denoted as g with slight abuse of notation.

on error-free channel g completes service in IWFQ no later than time $t_{ef} + B/C$ as compared to error WFQ by noting that the server rate is C. Therefore, the result follows for slot s. The arbitrary choice of starting time t also implies that the arbitrary choice of slot s; hence, the result holds true for any slot s of flow g, which concludes the proof. □

Theorem 1 *(delay guarantee) For any packet i on a error-free channel g, its maximum delay $\bar{D}_{g,\ IWFQ}$ in IWFQ satisfies:*

$$\bar{D}_{g,\ IWFQ} \leq \bar{D}_{g,\ WFQ} + \bar{d}_g \tag{8}$$

where $\bar{D}_{g,\ WFQ}$ is the maximum packet delay of flow g in the error-free WFQ, and $\bar{d}_g$ is given by equation (7).

Proof. For lagging slots, the proof follows from Lemma 2. For a leading slot, by definition, its delay is less than what it would have been under error-free WFQ. □

Though the worst-case packet delay of a leading flow does not increase, its packets can be well ahead of their schedule. Thus its new queue delay[5] has a bounded increase, as shown in Corollary 1.

Theorem 2 *(long-term throughput guarantee) For a error-free channel g, let $S_g(0,t)$ denote the aggregate service (in bits) received by channel g in the interval $[0,t]$ in the IWFQ, and $S_g^*(0,t)$ denote the aggregate service (in bits) received by channel g in the interval $[0,t]$ in the error-free WFQ service, then the following inequality holds:*

$$S_g(0, t + \bar{d}_g) \geq S_g^*(0,t) \tag{9}$$

where $\bar{d}_g$ is given by equation (7).

Proof. Let t_N be the finish (real) time of the Nth packet under error-free WFQ and t'_N be the finish time of the same packet under IWFQ. Then, by Lemma 2, $t'_N \leq t_N + \bar{d}_g$. Also, let $S_g^*(0,t) \geq NL_P$, for some integer N. We use the well-known relationship $S_g^*(0,t) \geq NL_P \Leftrightarrow t_N \leq t$. From above, $t'_N - \bar{d}_g \leq t$, which leads to $S_g(0, t + \bar{d}_g) \geq NL_P$. Hence, for any $N > 0$,

$$S_g^*(0,t) \geq NL_P \Rightarrow S_g(0, t + \bar{d}_g) \geq NL_P.$$

which leads to the inequality (9). □

Based on Lemmas 1 and 2, the following result is easily derived for new queue delay.

Theorem 3 *(new queue delay bound) For a flow g on error-free channel, its maximum new queue delay $\bar{D}_{HOL}$ is given by*

$$\bar{D}_{HOL} = \bar{d}_g + \bar{d}_{WFQ} + \bar{T}_g, \tag{10}$$

$$\bar{T}_g = \frac{l_g(\sum_{j \in F_g} r_j)}{Cr_g}, \quad \bar{d}_{WFQ} = \frac{L_P}{C} + \frac{L_P}{C}\frac{\sum_{c \in F} r_c}{r_g}, \tag{11}$$

where $\bar{d}_g$ is given by (7) and $F_g = F \setminus g$.

Proof. $\bar{d}_{WFQ}$ is the time spent in the head-of-the line (HOL) if the HOL packet contends for service immediately. $\bar{T}_g$ is the maximum amount of time that one has to wait before contending at the HOL due to the fact that flow g might be leading even if the HOL packet arrived at an empty queue. To derive the expression for $\bar{T}_g$, note that flow g can lead by l_g bits or equivalently, l_g/r_g bit rounds. Thus, the number of other flows' bits that can take precedence over a newly arrived packet is $\frac{l_g}{r_g}\sum_{j \in F_g} r_j$. Finally, $\bar{d}_g$ is the wait due to lagging flows. □

[5] New Queue Delay is the maximum delay for a packet that arrives at an empty queue.

Theorem 4 *(short-term throughput guarantee) Given any time t, for a error-free channel g, assume that the service which flow g receives in error-free WFQ is given by $S^*(t,t_1)$ during time interval $[t,t_1]$ for some $t_1 > t + T_g(t)$. Then the service that flow g receives under IWFQ satisfies*

$$S(t,t_1) \geq S^*(t + T_g(t), t_1), \tag{12}$$

$$T_g(t) = \left(\sum_{j \in F_g} b_j(t) + \frac{l_g(t)}{r_g} \sum_{j \in F_g} r_j \right) \frac{1}{C}. \tag{13}$$

where $b_j(t)$ is the number of lagging bits of flow j and $l_g(t)$ is the number of leading bits of flow g at time t.

Proof. At any time t, the amount of time that flow g has to wait to begin service is determined by the number of lagging bits of other flows as well as the amount by which it is leading. The amount of time due to other lagging sources is $\sum_{j \in F_g} b_j(t)/C$, from the definition of $b_j(t)$. In addition, flow g has to possibly wait for $l_g(t)/r_g$ bit-by-bit rounds and the maximum amount of time due to this is bounded by $\frac{l_g(t)}{r_g}\sum_{j \in F_g} r_j/C$. □

Note that the above theorem is trivially valid when all channels are error-free because in that case, $b_j(t) = l_g(t) = 0$. In addition, one cannot give short-term guarantees over intervals smaller than $T_g(t)$. This highlights our observation in Sections 1 and 2 that trying to provide very short-term guarantees in wireless channels will not allow enough flexibility to hide short error-bursts from users.

One can trivially remove the dependence on t in the lower bound of the above theorem as follows.

Corollary 1 *Assume that flow g is backlogged during $[t,t_1]$ in the error-free WFQ, then the service it receives in the IWFQ satisfies*

$$S(t,t_1) \geq S^*(t + T_g, t_1) \tag{14}$$

where $T_g = \left(\sum_{j \in F_g} b_j + \frac{l_g}{r_g}\sum_{j \in F_g} r_j\right)\frac{1}{C}$, if $t_1 - t > T_g$. □

5.2 Error-prone channels

Theorem 5 *(delay bound for an error-prone channel) Given any packet of flow e on an error-prone channel, its maximum packet delay $\bar{D}_{IWFQ,\ e}$ is given by*

$$\bar{D}_{IWFQ,\ e} \leq \bar{D}_{WFQ,\ e} + T_{e,(M+1)} \tag{15}$$

where $\bar{D}_{WFQ,\ e}$ is the maximum packet delay of flow e under error-free WFQ, and M is the maximum number of lagging slots of all flows other than flow e, $M = \sum_{j \in F_e} b_j/L_P$, and $T_{e,(M+1)}$ is the maximum time it takes for flow e to have its $(M+1)^{th}$ good slot starting from any time t.

Proof We assume the flow e is lagging. As in Lemma 2, if there are no further errors after time t the delay of a packet of e is increased by the lagging slots of other flows M. However, we have to additionally account for possible errors in the channels of flow e and other lagging flows. Suppose the $(M+1)^{th}$ good state for flow e after time t, occurs at $t + T_{e,(M+1)}$, then we claim that the head-of-the-line packet at time t for flow e would be transmitted no later than $t + T_{e,(M+1)}$. Suppose this were not true, then it would lead to the following conclusion: during all the $M+1$ good states, one of the other flows had a slot with a lower finish tag. This contradicts the upper bound of M on the number of lagged slots.

Assume a packet arrives when a flow is leading. If the packet finishes when the flow is still leading, then the statement of the theorem is trivially true. If it finishes when the flow is lagging, then there is a time instant before the packet's departure when it is in the queue and the flow starts lagging. Then the above proof holds. □

Note that the previous result does not take into account a specific model for channel errors. Any channel error model that deterministically or probabilistically bounds $T_{e,M+1}$, could be easily incorporated into the bound.

Based on the above result on delay bound, the result on throughput follows readily along the lines of Theorem 2.

Theorem 6 *(long-term throughput guarantee) For a flow e on an error-prone channel, let $S_e(0,t)$ denote the aggregate service (in bits) received by flow e in the interval $[0,t]$ in the IWFQ, and $S_e^*(0,t)$ denote the aggregate service (in bits) received by channel e in the interval $[0,t]$ in the error-free WFQ service, then the following inequality holds:*

$$S_e(0, t + T_{e,M+1}) \geq S_e^*(0,t) \quad (16)$$

Moreover, we can further show the following result for short-term throughput:

Theorem 7 *(short-term throughput guarantee) Given any time t, for a continuously backlogged flow e on an error-prone channel during time interval $[t,t_1]$, the aggregate service (in bits) received by flow e in the interval $[t,t_1]$ in the IWFQ, denoted by $S_e(t,t_1)$, satisfies:*

$$S_e(t,t_1) \geq (N_G - N(t)) \frac{r_e}{\sum_{i \in F} r_i} L_P - L_P, \quad (17)$$

$$N(t) = \left(\sum_{i \in F_e} b_i(t) + l_e(t) \frac{\sum_{i \in F_e} r_i}{r_e} \right) / L_P, \quad (18)$$

where N_G is the number of time slots in good state for flow e in $[t,t_1]$,, $b_i(t)$ is the number of lagging bits of flow i and $l_e(t)$ is the number of leading bits of flow e at time t. □

The proof of the above theorem follows along the lines of the proof of Theorem 3.

6 Implementation Issues in Wireless Packet Scheduling

In previous sections, we developed an idealized wireless fair queueing algorithm in the presence of bursty and location-dependent errors, assuming full knowledge of the channel. However, when implementing a practical wireless fair scheduling algorithm, we need to address the following important constraints: (a) the channel state is not known in advance and cannot be predicted with complete accuracy, (b) due to error and incorrect channel prediction, transmitted packets may be lost, (c) detection of reception errors is not just a matter of sensing the carrier, since errors at the source do not imply errors at the destination and vice-versa, (d) the base station performs the scheduling, but does not have a full knowledge of which uplink flows have packets to transmit or how many packets a backlogged flow has, and (e) since errors are typically bursty, giving precedence to packets of lagging flows (as IWFQ does) will cause error-prone channels to be polled more often, which increases the scheduling overhead. Due to our channel model, problems of hidden and exposed stations across multiple shared channel cells are not addressed.

Several of the above constraints either pertain to, or can be effectively addressed at, the medium access layer. Hence, one key conclusion for a practical implementation of wireless fair scheduling is that it must be closely coupled with the MAC protocol. In this section, we first identify some MAC-level instruments to address the above issues, and then briefly describe our wireless medium access protocol. In the next section, we describe our wireless fair scheduling algorithm.

6.1 Techniques to address wireless channel issues

As before, we assume that packets are small and of fixed size; these are very reasonable assumptions for wireless networks. Time is slotted, and each data slot accomodates some control information, a data packet and an acknowledgement.

Acknowledgement: In our approach, each packet transmission is followed by a short acknowledgement from the destination to the source. Using acknowledgements serves a number of purposes. The most important purpose is to detect loss of packets during transit. As a side-effect, acknowledgements also imply that the base station transmits either the data packet or the ack packet in every transmission - we use this feature to piggyback important control information for future slots on the base station's transmission in the current slot. Acknowledgements have been used in several medium access protocols [3, 10] for similar purposes.

One-Step Prediction: Since errors are bursty and errors in successive slots are highly correlated, we perform a *one-step channel prediction* by monitoring the channel condition in the previous slot. Since the base station transmits either a data packet or an ack packet in every slot, each host in the cell monitors the channel during each slot for packets from the base station. If a host can sense activity on the channel but does not receive a good packet from the base station, it detects an error during the current slot. A host predicts that its next slot will be in the same state as the current slot, due to the high correlation of channel state across slots. While the one-step prediction is obviously not perfect, our simulation results show that it is very effective for typical wireless channel error models.

One undesirable consequence of the one-step prediction approach is that every host (with a backlogged uplink flow) has to monitor every slot, which can increase its power consumption. In the future, we plan to experiment with periodic snooping of the channel and dynamically estimate the optimal snooping period in order to alleviate the problem of having the host be in promiscuous mode all the time.

Set of known backlogged flows: Since the base station must schedule packets for both downlink and uplink flows, it needs to know at least which uplink flows are backlogged at any time. In order to allocate slots only to flows which have packets to transmit, the base station keeps a set of 'known backlogged flows' and only allocates slots among the flows in this set. The set is updated by the following mechanisms: (a) for downlink flows, the base station has a precise information about the queue lengths, (b) when an uplink flow is allocated a slot, it piggybacks the queue size information on its data packet, (c) when a new uplink flow is created or becomes backlogged, if there is an ongoing backlogged flow from the same mobile host, the information is piggybacked in (b) above, and (d) the base station periodically solicits notifications from new (and newly backlogged) uplink flows by issuing of a control slot. One of the highlights of our approach is the way in which control and data slots are integrated in the MAC framework.

6.2 Wireless Medium Access Protocol

Our wireless medium access protocol has its origins in DQRUMA [9]. We divide time into frames, and each frame into slots (as described in Section 7, the frame size is not fixed, and the number of slots in a frame changes over time). A slot may be either a data slot or a control slot. Each data slot is subdivided into three parts: a *control sub-slot* which consists of *four mini-slots*, a *data sub-slot* and an *ack sub-slot*. Each control slot is divided into a *notification sub-slot* and an *advertisement* sub-slot. By means of the scheduling algorithm described in Section 7, the base station allocates the slots in a frame among known backlogged flows before the start of the frame. Due to lack of space, this section provides only a brief outline of key actions of the MAC protocol which relate to scheduling, i.e. the mechanics of slot allocation, and the mechanics of identification of newly backlogged flows. For a more detailed general discussion on MAC protocol issues, we refer the reader to [3, 9].

Identification of New and Backlogged Flows: The base station has a special downlink 'broadcast' flow called the *control flow*, which has a flow id of <0, downlink, $0>$. From the scheduling perspective, a control flow is identical to a backlogged data flow of unit weight on an error-free channel. However, when the control flow is allocated a slot, the MAC layer at the base station issues a control slot as opposed to a data slot. The control slot consists of two phases: a notification sub-slot during which mobile hosts contend in order to notify the base station of new or newly backlogged flows, and an advertisement sub-slot during which the base station broadcasts the newly received notifications as an acknowledgement to the successfully contending mobile hosts.

The notification sub-slot has a sequence of mini-slots. If a mobile host has a newly backlogged flow but does not have an ongoing backlogged flow on which to piggyback this information, it selects a random mini-slot during which it transmits the notification. During the advertisement sub-slot, the mobile host knows if it's notification was successfully received. This contention mechanism is novel in the way control and data flows are integrated. However, it is simplistic in that contending mobile hosts can only transmit once in a control slot. Using Slotted Aloha to contend in the control slot will improve the probability of successfully sending notifications. Note, that the above contention mechanism impacts the delay and throughput bounds of new flows in Section 5; the changes are easy to compute using results from Slotted Aloha.

Data Slot Allocation: Since all flows are either uplink or downlink, in each data slot the base station must transmit either the data packet or the acknowledgement packet. Piggybacked on the packet, the base station provides the ids of the flows which are allocated the next three slots (as a special case, a control slot is identified by setting all the flow ids to <0, downlink, $0>$). Since every host in the cell is within range of the base station, a source of an identified flow will be able to hear the packet if its channel is good. In the control phase of the next slot, the source of flow i ($1 \leq i \leq 3$) transmits a *channel good* flag in mini-slot i if it predicts that the channel will be good (based on one-step prediction). In the fourth mini-slot, the base station identifies the flow which has been chosen for transmission during the current slot, which is the first among the three flows to send the good flag in its mini-slot. If it turns out that all the identified flows are in error, then the base station picks any one downlink flow for transmission.

When an uplink flow transmits a packet, it piggybacks the number of packets in its queue. When this number reaches zero, the base station removes the flow from its set of known backlogged flows.

7 Wireless Scheduling Protocol

In this section, we describe a wireless packet scheduling (WPS) algorithm that approximates the idealized algorithm while addressing the issues of practical implementation.

Within the constraints identified in Section 6, the following are the key requirements of the wireless packet scheduling algorithm: (a) it should provide fair channel access among flows which are known to be backlogged, (b) it should utilize the location-dependent channel error property in order to *locally swap* slots (preferably within a frame) between flows in order to accommodate short error bursts[6], (c) across frames, it should provide a system of maintaining *credits* for lagging flows and *debits* for leading flows in case swapping within a frame is not possible (as in IWFQ, both credits and debits should be bounded in order to provide separation), (d) since errors are known to be bursty in wireless channels, it should *spread* the slots allocated to each flow as well as possible within the frame, (e) since errors are bursty, flows which perceive channel error should not be repeatedly polled in subsequent slots (the tagging mechanism in IWFQ will end up doing this since it gives higher precedence to flows that have been lagging the longest), (f) well-behaved flows with error-free channels should be affected as less as possible while still accommodating flows which perceive errors, and (g) the scheduling algorithm should be simple.

The major departure in WPS from IWFQ is that we have moved from the fair queueing to the weighted round robin paradigm. This was motivated by the fact though weighted round robin is much simpler to implement, in our environment, weighted round robin and fair queueing will result in identical error-free service for the following reasons: (a) the base station allocates slots only among known backlogged flows, (b) packets are of fixed size, (c) the base station can only periodically know when an empty flow has been backlogged (for the uplink case); in particular, if a backlogged flow drains its queue during a frame, it drops out of contention for slots until the next new queue phase even if it becomes backlogged subsequently, and (d) when all flows contending for the channel are backlogged, by spreading slot allocation appropriately within each frame, we can exactly replicate the WFQ or WF^2Q service for error-free service.

Thus, WPS modifies the basic weighted round robin scheduling algorithm in order to accommodate location-dependent and bursty wireless channel errors. The following are the key features of the WPS algorithm:

- *Spreading*: generates a slot allocation identical to WFQ [11] or WF^2Q when all flows are backlogged.
- *Swapping within frame*: when a flow cannot transmit in its slot because of channel error, it tries to swaps its slot with another backlogged flow which has (a) been allocated a slot later in the same frame, and (b) perceives a good channel at the current time; intra-frame

[6] In Section 6, we have used the term 'slot' to mean physical time slots as well as the logical slots which compose a frame. In this section, we only deal with logical slots when we refer to 'slots'.

swapping is a first level mechanism to accommodate location-dependent errors.

- *Credit adjustment*: when a flow f_1 cannot transmit during its slot and cannot swap slots within a frame, but there is at least one backlogged flow f_2 that can transmit at the current time (f_2 does not have any slots during the remainder of the frame), f_1's credit is incremented and f_2's credit is decremented (both within bounds); the effective weight of each flow at the start of a frame is the aggregate of its default weight and its credit, and the spreading algorithm generates a slot allocation with respect to the effective weights of the flows. Thus, credit adjustment compensates lagging flows at the expense of leading flows in future frames.

- *One-step prediction*: predicts that the channel state for the current time slot will be the same as the monitored channel state during the previous time slot.

We show through intuitive arguments and simulation results that a combination of the above features will address all of the above requirements for fair slot allocation in wireless channels, while also closely approximating IWFQ for the average case.

```
schedule_frame() /* main procedure to schedule packets in frame */
  compute_effective_weights();
  spread_new_frame(); /* spread according to WFQ;
                         ignore flows with effective credit < 0 */
  marker = first_slot_of_frame;
  while(marker != NULL)
    f = get_next_flow();
    marker->marker->next;
    if (f != NULL) transmit_head_of_line_packet(f);
           /* transmit the hol packet and increment f->attempts */

compute_effective_credits() /* compute effective credits for all
                               flows at the start of a frame */
  for each flow f,
    f->credit = min(max(f->effective_weight - f->attempts,
                    -(f->debit_limit)), f->credit_limit);
    f->effective_weight = f->weight + f->credit;
    f->attempts = 0;

get_next_slot() /* return flow that can transmit in current slot;
                   perform swapping and credit/debit allocation */
  while(marker != NULL)
    if (queue_empty(marker->flow)) /* case 1: flow has no queue */
      delete_flow_slots_from_frame(marker->flow);
      delete_flow_from_backlogged_set(marker->flow);
      marker = marker->next;
    else if (exception_case()) /* case 2: no flow can transmit */
      (marker->flow)->attempts ++; /* no credit for missed slot */
      marker = marker->next;
      return NULL;
    else if (slot_state(marker->flow) == ERROR)
                            /* case 3: channel is in error */
      for(s = marker->next;s != NULL;s = s->next)
        if ((!(queue_empty(s->flow)))&&(slot_state(s->flow)==CLEAN))
          break; /* flow pointed by s can swap with marker->flow */
      if (s != NULL)
        swap_flows(marker, s);  /* case 3a: intra frame swap */
      else
        return find_next_good_flow(); /* case 3b: no swap; credit/
            debit is implicit due to how f->attempts is updated */
    else /* case 4: connection has packet and slot is good */
        return marker->flow;
  return NULL;
```

The above pseudo code describes the essential parts of the WPS algorithm. We now comment briefly on some noteworthy points in the WPS algorithm.

A flow which is unable to transmit in its slot receives credit only if some other flow is able to transmit in its place. When a flow transmits more slots in a frame than it is initially granted, its credit becomes negative. Hence, even when we cannot swap within a frame, the system of credit/debit adjustment implicitly preserves the notion of a swapping, just as lag and lead implicitly preserves the notion of swapping in IWFQ.

A flow f_i with a negative credit of c_i will not receive any channel capacity for $\lfloor |c_i| \rfloor / r_i$ frames (where the size of a frame is $\sum_{i \in B(t)} w_i$ slots, and B(t) is the set of known backlogged flows at the start of the frame, and w_i is the effective weight of flow f_i).

The credit adjustment policy above compensates all the credits of a lagging flow in the next frame. For a flow that has accumulated a large number of credits, this could potentially result in the flow capturing the channel for a long time after its channel becomes good (IWFQ also has a similar effect in case a flow has been lagging for a long time). In order to compensate lagging flows over a longer period of time, we could bound the number of credits that can be reclaimed in a single frame by any flow, thus amortizing the compensation over several frames.

In the average case, WPS closely approximates IWFQ because is tries to achieve separation and compensation by similar instruments (credit/debit similar to lag/lead, and bounds as in IWFQ). However, there is a difference in worst-case delay since we compensate by swapping rather than by giving precedence in channel access to longest lagging flows. By swapping, we delay the next attempt of the flow to access the channel to a later slot, not necessarily the next slot. Thus, swapping loses the precedence history which IWFQ maintains. While this is good for a practical implementation (otherwise the base station will need to poll the most error-prone channel most often), it can starve out a flow under some pathological conditions. Consider an example in which a flow always perceives an error in precisely the exact slots when it is scheduled to transmit, but has some good slots in between when other flows are scheduled to transmit. In IWFQ, the flow will eventually have the minimum service tag and gets highest precedence for transmission in any slot; in WPS, the flow can contend only in designated slots and will be starved. Thus, though we expect the average delay of packets to be very close for WPS and IWFQ, the worst case delay of WPS is ∞.

One important question is the following: when a backlogged flow is unable to transmit because of channel error, and is unable to swap slots within its current frame, how does it choose a flow in a future frame with which it can swap slots? Of course, if we generated slot allocations for several future frames in advance, it would be possible to simply pick the first flow in a future frame that can transmit in the current slot. However, we do not maintain future frames. Instead, we generate a weighted round robin ring (with $\mathrm{WF}^2\mathrm{Q}$ spreading) based on the default weights for all known backlogged flows after each new queue phase. A marker in this ring identifies the last flow that was selected for swapping across frames. When intra-frame swapping fails, we simple advance the marker around the ring until we find a flow that can swap with the current slot.

8 Simulation Results

This section presents the simulation results for the WPS algorithm. As described in Section 7, there are four key components of the algorithm: spreading, swapping, credit adjustment, and prediction. In order to isolate the effect of each of these components, we simulated several different algorithms, with different combinations of the above components.

The following are the algorithms we simulated and compare in this section:

- *Blind WRR* spreads slots according to $\mathrm{WF}^2\mathrm{Q}$, but does not attempt to predict the channel state.
- *WRR* modifies Blind WRR by skipping the slot if the channel for the flow is known (in WRR-I) or predicted (in WRR-P) to be in error.
- *NoSwap* combines spreading and credits (but no debits), but does not have any intra-frame swapping. If the channel for the current flow is known (NoSwap-I) or predicted (NoSwap-P) to be in error, it gives a credit to the flow and skips to the next slot.
- *SwapW* combines spreading, swapping and credits (but no debits). If the channel for the current flow is known (SwapW-I) or predicted (SwapW-P) to be in error, it first tries to swap within the frame. Otherwise, it gives a credit to the flow and skips to the next slot.
- *SwapA* combines spreading, swapping, and credit/debit adjustment. SwapA is identical to the WPS algorithm described in Section 7.

We start by illustrating the key ideas using examples with only two sources. This allows us to demonstrate the effect of the various parameters clearly. Later we consider examples with more sources to illustrate some differences in the performance when there are small number of sources as opposed to a large number of sources.

Example 1: We consider an example with two loss-sensitive sources with WFQ weights $r_1 = 1, r_2 = 1$. For the purposes of simulation, we assume that the channel for Source 2 has no errors and the channel for Source 1 evolves according to a two-state discrete Markov Chain. Let p_g be the probability that the next time slot is *good* given that the current slot is in error, and p_e be the probability that the next time slot is in error given that the current slot is *good*.

Then, the steady-state probabilities P_G and P_E of being in the *good* and *bad* states, respectively, are given by

$$P_G = \frac{p_g}{p_g + p_e}, \quad P_E = \frac{p_e}{p_g + p_e}.$$

The arrival processes are assumed to be as follows:

- Source 1 is a Markov-modulated Poisson process (MMPP) where the modulated process is a continuous-time Markov chain which is in one of two states ON or OFF. The transition rate from ON to OFF is 9 and OFF to ON is 1. When the Markov chain is in the ON state, arrivals occur according to a Poisson process of rate 2.
- Source 2 has constant inter-arrival time of 2.

Note that the channel being Markov is not necessary for our algorithm, it is just used for the purposes of simulation. For the two-state Markov chain describing the channel process for Source 1, if we let the *bad* state be 0 and the *good* state be 1, it is easy to see that the one-step autocovariance function is

$$C(1) \equiv E(X(t)X(t+1)) - E(X(t))E(X(t+1)) \\ = P_G P_E(1 - (p_g + p_e)).$$

If $p_g + p_e \leq 1$, then $C(1) \geq 0$. Further $C(1)$ is a decreasing function of $p_g + p_e$, and therefore, as $p_g + p_e \uparrow 1$, successive time slots become less correlated. Thus, it is a natural to test our prediction algorithm for various values of $p_g + p_e$ with P_G and P_B fixed. We fix the steady-state probability for Channel 1 as $P_G = 0.7$. For each packet, we limit the maximum number of retransmissions to 2, i.e., a packet is dropped if it is not successfully transmitted after three attempts. We also limit the number of credits and number of debits to four.

	d_1	l_1	d_1^{max}	σ_{d_1}	d_2	l_2	d_2^{max}	σ_{d_2}
Blind WRR	19.5	0.15	127	19.4	0	0	0	0
WRR-I	43.6	0	266	41.3	0	0	0	0
NoSwap-I	25.3	0	185	26.6	1.7	0	6	2.3
SwapW-I	25.1	0	185	26.5	1.7	0	6	2.3
SwapA-I	21.6	0	166	22.8	2.3	0	10	3.4
WRR-P	54.7	0	297	52.0	0	0	0	0
NoSwap-P	28.2	0	225	29.1	1.8	0	6	2.4
SwapW-P	28.1	0	225	29.1	1.8	0	6	2.4
SwapA-P	24.1	0	190	25.3	2.5	0	10	3.5

Table 1: Example 1. Results for $p_g + p_e = 0.1$

	$\bar{d}_1$	l_1	d_1^{max}	σ_{d_1}	$\bar{d}_2$	l_2	d_2^{max}	σ_{d_2}
Blind WRR	21.3	0.058	152	20.6	0	0	0	0
WRR-I	28.4	0	176	26.3	0	0	0	0
NoSwap-I	11.4	0	92	10.7	1.0	0	6	1.5
SwapW-I	11.2	0	92	10.6	1.0	0	6	1.6
SwapA-I	11.0	0	92	10.2	1.1	0	10	1.8
WRR-P	115.5	0.003	369	79.4	0	0	0	0
NoSwap-P	18.1	0.003	138	16.8	1.5	0	6	1.9
SwapW-P	17.7	0.003	136	16.5	1.6	0	6	1.9
SwapA-P	16.8	0.003	123	15.3	1.8	0	10	2.4

Table 2: Example 1. Results for $p_g + p_e = 0.5$

Simulation results are presented in Tables 1– 3. The performance of the various scheduling algorithms are compared using the following three performance measures for Source i, $i = 1, 2$:

- $\bar{d}_i$: Average delay of successfully transmitted packets
- l_i : Loss Probability, i.e., fraction of packets that are dropped after four transmission attempts
- d_i^{max} : Maximum delay of successfully transmitted packets
- σ_{d_i}: The standard deviation of the delay

Our main conclusions from the simulation results are:

- The scheduling algorithms assuming perfect state information, WRR-I, NoSwap-I, SwapW-I and SwapA-I always perform better than the Blind WRR algorithm. This means that our basic idea of credits significantly improves performance if we have perfect knowledge about the state of each channel. Note that we have assumed that the sources are loss sensitive, and thus our objective is to make the loss probabilities close to zero.

	d_1	l_1	d_1^{max}	σ_{d_1}	d_2	l_2	d_2^{max}	σ_{d_2}
Blind WRR	24.3	0.029	158	22.6	0	0	0	0
WRR-I	28.8	0	170	26.5	0	0	0	0
NoSwap-I	10.2	0	91	9.3	0.63	0	6	1.0
SwapW-I	9.9	0	91	9.1	0.65	0	6	1.0
SwapA-I	9.9	0	91	9.1	0.65	0	6	1.0
WRR-P	6301.5	0.029	11930	3430	0	0	0	0
NoSwap-P	26.8	0.028	179	22.6	1.7	0	6	1.6
SwapW-P	26.4	0.027	182	22.5	1.7	0	6	1.6
SwapA-P	26.4	0.027	182	22.5	1.7	0	8	1.6

Table 3: Example 1. Results for $p_g + p_e = 1.0$

- In all the cases where $p_g + p_e < 1$, the one-step prediction algorithms WRR-P, NoSwap-P, SwapW-P and SwapA-P perform significantly better than the Blind WRR. Thus, when consecutive time slots are positively correlated, our simple one-step prediction works remarkably well. In general, prior studies of wireless channel errors have indicated that errors occur in bursts although models for the error can vary. Thus, our algorithm works very well if the channel errors are indeed bursty.

- When $p_g + p_e = 1$, our prediction algorithms perform poorly. In fact they perform worse than Blind WRR. The fact that $p_g + p_e = 1$ implies that the channel states are Bernoulli random variables with the probability of being *good* chosen as P_G. Thus, channel states at successive time slots are uncorrelated and it is not surprising that the one-step prediction performs poorly. However, this is a very unrealistic model of a wireless channel which contradicts all prior claims of bursty channel errors. We chose to perform this experiment to show when our algorithms can break down.

- NoSwap, SwapW and SwapA perform better than WRR in the following sense: They all reduce the delay for Source 1 significantly while increasing the delay slightly for Source 2. This illustrates the advantage of compensating a source for time slots during which its channel is in error. The difference between the algorithms NoSwap, SwapW and SwapA as compared to WRR is even more dramatic when one does not have perfect knowledge of the channel state.

- As is to be expected, swapping will perform better when the channel errors are bursty. The idea is that when a channel is in a sustained bursty state, it is advantageous to let other channels transmit and get compensation later. Thus, the tables show that SwapA is the preferred algorithm especially when the channel state is strongly positively correlated. However, Source 2's delay is slightly higher with swapping as compared to the case when there is no swapping. However this can be adjusted by changing the debit parameter as illustrated in a later example with five sources.

Example 2: We consider an example with the same parameters as in Example 1 , except that instead of setting an upper limit on the number of retransmission attempts per packet, we set an upper limit on the maximum delay of a packet to be 100. If a packet is in the system for more than 100 time slots, then it is dropped; this could possibly happen even before it reaches the head-of-the-queue. Thus, we now assume that the sources are also delay-sensitive. We let $p_g + p_e = 0.1$ and the results are shown in Table 4. From the results, it should be clear that this example complements Example 1 by leading to the same conclusions regarding the relative performances of the various algorithms when the sources are both delay and loss sensitive.

Example 3 We consider a three-source example with the channel and source parameters as in Table 5. Source 1's arrivals occur according to an MMPP process with the modulating Markov chain as in Example 1, Source 2's arrivals are Poisson and Source 3's arrivals have a constant inter-arrival time. The arrival rate for Source i is denoted by λ_i. The maximum number of credits, debits and retransmissions are chosen as in Example 1.

The delay and loss performance of Blind WRR, WRR-P and SwapA-P are shown in Table 6. Again this example illustrates that SwapA-P trades off the performance of a severely errored channel (in this case Channel 1) against the less error-prone channels in a better fashion than WRR-P. For instance compare to WRR-P, d_1 is decreased by 26% while the delay for d_2 increase 6% and the d_3 increases by 15%. The increases in the delays of Sources 2 and 3 can be further controlled by suitable choice of upper limits on credits and debits as will be shown in a later example. The main conclusion from the examples so far is that SwapA-P allows greater flexibility in hiding errors from a source. However, when the number of sources is larger, the differences between WRR and SwapA depend on how heavily loaded the system is as shown in the following examples.

	d_1	l_1	d_1^{max}	σ_{d_1}	d_2	l_2	d_2^{max}	σ_{d_2}
Blind WRR	32.4	0.025	100	26.9	0	0	0	0
WRR-I	32.4	0.025	100	26.9	0	0	0	0
NoSwap-I	23.2	0.007	100	22.6	1.6	0	6	2.3
SwapW-I	23.1	0.007	100	22.6	1.7	0	6	2.3
SwapA-I	20.7	0.003	100	20.8	2.3	0	10	3.4
WRR-P	34.9	0.035	100	27.7	0	0	0	0
NoSwap-P	25.3	0.009	100	23.7	1.8	0	6	2.3
SwapW-P	25.2	0.009	100	23.6	1.8	0	6	2.4
SwapA-P	22.5	0.006	100	21.9	2.5	0	10	3.5

Table 4: Results for Example 2

Source	λ_i	p_g	p_e
1	0.2	0.07	0.03
2	0.25	0.095	0.005
3	0.25	0.09	0.01

Table 5: Example 3. Source and Channel Parameters

Example 4: We consider an example with five sources. The channels for all five sources are assumed to evolve according to independent discrete-time two-state Markov chains. Sources 2 and 4 are assumed to generate arrivals according to independent Poisson processes, and Sources 1, 3 and 5 are MMPP sources with the underlying Markov chain having the same parameters as that of the MMPP source in Example 1. The arrival rate of source i is denoted by λ_i and the parameters of the sources and channels are given in Table 7. The WFQ weights r_i, $i = 1, 2, 3, 4, 5$, are all assumed to be 1, the maximum number of retransmissions for Sources $1, 2, 3$ and 5 is set to 2, and is set equal to zero for Source 4. Note that the arrival rates for Sources 2 and 4 are so high that their delays would be unbounded. In other words, Sources 2 and 4 have packets to send almost all the time, and the only measure of performance that is relevant to these sources is the throughput, or equivalently, packet loss probability. The maximum number of debits and credits for all sources is set equal to 4. The performance results are presented in Table 8. In order of avoid excessive use of space, we only present average delay and loss probability in the table. The main conclusion from Table 8 is that SwapW-P is clearly superior to the other algorithms that use one-step prediction. Both Examples 1 and 4 show that there is not a significant advantage to swapping slots within a frame, but swapping slots across multiple frames, and using credits and debits is clearly superior to WRR.

	d_1	l_1	d_2	l_2	d_3	l_3
Blind WRR	41.6	0.134	1.1	0.021	10.9	0.038
WRR-P	59.2	0	1.7	0	8.0	0
SwapA-P	44.0	0	1.8	0	9.2	0

Table 6: Example 3. Average Delay and Loss Performance

Source	λ_i	p_g	p_e
1	0.08	0.09	0.01
2	8.0	0.095	0.005
3	0.08	0.08	0.02
4	8.0	0.07	0.03
5	0.08	0.035	0.015

Table 7: Example 4. Source and Channel Parameters

	d_1	l_1	l_2	d_3	l_3	l_4	d_5	l_5
Blind WRR	40.4	0.025	0.011	75.4	0.057	0.43	253.6	0.12
WRR-I	14.5	0	0	26.0	0	0	85.4	0
NoSwap-I	14.8	0	0	19.6	0	0	64.1	0
SwapW-I	14.7	0	0	19.4	0	0	64.6	0
SwapA-I	14.4	0	0	17.4	0	0	46.0	0
WRR-P	15.8	0	0	29.6	0.0	0.03	96.5	0
NoSwap-P	15.8	0	0	22.0	0	0.04	73.3	0
SwapW-P	15.7	0	0	21.8	0	0.03	73.7	0
SwapA-P	15.6	0	0	19.9	0	0.03	52.5	0

Table 8: Example 4. Delay and Loss Performance

Example 5: We now present a situation where WRR-P performs as well as SwapA-P. Consider the same parameters as in Example 4, except that the arrival rates for Sources 2 and 4 are now assumed to be equal to 0.07. This system is stable since

$$\sum_{i=1}^{5} \lambda_i / P_{G_i} < 1,$$

and

$$\lambda_i < P_{G_i}, \ \forall i.$$

Average delay and loss probabilities are shown in Table 9. The performance of WRR-P and SwapA-P are virtually identical. The reasons for this are two-fold:

- Since the number of sources is 5, the frame size for WRR is 5 and thus the chance of the same channel being in error in multiple frames is small. Thus, credits are not accumulated very often in SwapA-P.
- WRR-P naturally allocates "credits" because of the stability of the system. In other words, if a source is skipped over during a frame due to error in its channel, it will automatically be compensated later because other sources will eventually "run out" of packets due to the system stability condition.

However, as the following example shows, it would be erroneous to conclude from the previous example that WRR and SwapA are virtually identical when the number of sources is large and the stability condition is satisfied.

Example 6: Consider a five source example with the channel and source parameters given in Table 10. Note that Sources 1 through 4 are identical and Source 5's channel has a higher steady-state error probability than the rest. We limit the maximum delay to 200, and the number of credits and debits to 4 each. The average delay and loss probabilities for Source 1 (recall that Sources 1 through 4 are identical) and Source 5 are shown in Table 11 as a function of the maximum number of debits D for Sources 1-4, and the maximum number of credits C for Source 5. SwapA-P performs much better than WRR-P in the sense that Source 5's performance can be traded off more effectively against the

	d_1	l_1	d_2	l_2	d_3	l_3	d_4	l_4	d_5	l_5
WRR-P	4.1	0	1.3	0	7.3	0	6.7	0.03	19.7	0
SwapA-P	4.5	0	1.5	0	7.3	0	6.0	0.03	19.1	0

Table 9: Example 5. Delay and Loss Performance

Source(s)	λ_i	p_g	p_e
1-4	0.095	0.005	0.19
5	0.07	0.03	0.1

Table 10: Example 6. Source and Channel Parameters

	D	C	d_1	l_1	σ_{d_1}	d_5	l_5	σ_{d_5}
WRR-P	-	-	167.5	0.30	28.9	155.1	0.23	58.1
SwapA-P	4	4	171.8	0.36	29.0	94.7	0.04	58.1
SwapA-P	2	4	171.8	0.36	28.9	100.4	0.05	58.7
SwapA-P	0	4	170.6	0.34	30.0	136.6	0.13	51.1
SwapA-P	0	1	169.5	0.31	31.2	145.3	0.17	47.7

Table 11: Example 6. Delay and loss performance

performance of the other sources. Further, it allows one to control this trade-off by using upper bounds on credits and debits. For example, Sources 1-4 could be low priority video sources which are loss tolerant. Thus, Source 5's quality has been dramatically improved by SwapA-P without significantly degrading the performance of Sources 1 through 4 as compared with WRR-P. The reason for this is as follows:

- Under WRR-P, even though the system is stable, it takes a long time for Source 5 to be compensated for errors in its channel since the other sources load the system rather heavily.
- In contrast, SwapA-P "hides" short error bursts from Source 5 by providing compensation over shorter timescales by using credits. It further avoid penalizing the less error-prone sources (Sources 1 thorough 4) by upper bounding the number of credits and debits.

9 Related Work

Fair packet scheduling algorithms have been the subject of intensive study in networking literature, particularly since the *weighted fair queueing* (WFQ) algorithm was proposed in [5]. The properties of WFQ were analyzed in [5, 12]. Several modifications to WFQ have been proposed to address its computational complexity or improve the performance of WFQ, two notable ones being the self-clocked fair queueing (SCFQ) [7] algorithm and WF^{2}Q[1]. Recent modifications to accommodate time-varying server capacity include STFQ [8]. While most of the above algorithms can handle time-varying server capacity with minor or no modifications, none of them handle the case when the variation in the capacity is location-dependent. In particular, we are not aware of any scheduling approach that reasons carefully about what to do when the shared channel is available to only a subset of the backlogged flows at any time. In addition to the fair queueing paradigm, there are several other paradigms for fair allocation of a shared channel (surveyed comprehensively in [14]). One such approach, which proposes to achieve long-term fairness at the expense of penalizing flows that use otherwise idle capacity, is Virtual Clock [13]. Section 3 points out the fundamental differences in the compensation model for IWFQ and Virtual Clock.

While wireline fair scheduling has been extensively researched, wireless fair scheduling is relatively unchartered territory. Typically, most related work on fairness in wireless channels approach it from a network-centric view (e.g. probability of channel access is inversely proportional to the contention perceived at the location of the host [3]), or provide fairly simplistic definitions of fairness [6]. In particular, most of the work in this area has been performed from the perspective of wireless medium access, where the emphasis has been on the mechanisms of channel access once the scheduling algorithm has been worked out [6, 9, 10], rather than the other way around.

In recent related work, some solutions to providing performance guarantees in the presence of the channel contention and dynamic reservation problems have been explored [4, 10]. The underlying idea is to combine the best features of some contention-based schemes like CSMA and contention-free schemes like TDMA. Performance analysis in terms of throughput and delay has been obtained [4, 10]. However, there are three major limitations of this approach. Firstly, channel errors and packet loss during transmission are ignored; second, the issue of location-dependent channel capacity is addressed only partially (as a function of contention); lastly, the scheduling issues in the higher level are typically unaddressed. As we argue in Section 1, they have to be studied together for an effective solution in the wireless domain.

Finally, a recent work on channel state dependent packet (CSDP) scheduling does address the issues of wireless medium access with a view to handling not only contention but also location-dependent error bursts [2]. However, it does not address the issues of fairness, throughput and delay guarantees.

10 Conclusions

Emerging indoor and outdoor packet cellular networks will seek to support communication-intensive applications which require sustained quality of service over scarce, dynamic and shared wireless channels. One of the critical requirements for providing such support is fair scheduling over wireless channels. Fair scheduling of delay and rate-sensitive packet flows over a wireless channel is not addressed effectively by most contemporary wireline fair scheduling algorithms because of two unique characteristics of wireless media: (a) bursty channel errors, and (b) location-dependent channel capacity and errors. Besides, in packet cellular networks, the base station typically performs the task of packet scheduling for both downlink and uplink flows in a cell; however a base station has only a limited knowledge of the arrival processes of uplink flows. In this work, we first propose a new model for fairness in wireless scheduling. This model adapts fluid fair queueing to wireless channels. We then describe an idealized wireless packetized fair queueing algorithm which approximates the wireless fluid fairness model under the assumption that the channel error is fully predictable at any time. For our idealized algorithm, we derive throughput and delay bounds for both error-free and error-prone channels. Finally, we describe a practical wireless scheduling algorithm which closely emulates the idealized algorithm, addresses several implementation-related wireless medium access issues, and uses simple one-step channel prediction. We observe that though the worst-case performance of the scheduling algorithm is much worse than the idealized algorithm, the average-case performance is remarkably close.

References

[1] J.C.R. Bennett and H. Zhang, "$\mathrm{WF}^2\mathrm{Q}$: Worst-case fair weighted fair queueing," *Proc. IEEE INFOCOM'96*, March 1996.

[2] P. Bhagwat, P. Bhattacharya, A. Krishma and S. Tripathi, "Enhancing throughput over wireless LANs using channel state dependent packet scheduling," *to appear on Proc. of IEEE INFOCOM'97.*

[3] V. Bharghavan, A. Demers, S. Shenker and L. Zhang, "MACAW: A Medium Access Protocol for Indoor Wireless LANs," *Proc. ACM SIGCOMM'94.*

[4] C. Chang, J. Chang, K. Chen and M. You, "Guaranteed quality-of-service wireless access to ATM," *preprint*, 1996.

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

[6] M. Gerla and J. T. Tsai. "Multicluster Mobile Multimedia Network," *ACM Baltzer Journal of Wireless Networks*, August 1995.

[7] S. Golestani, "A self-clocked fair queueing scheme for broadband applications," *Proc. IEEE INFOCOM'94*, June 1994.

[8] P. Goyal, H. Vin, and H. Cheng, "Start-time fair queuing: a scheduling algorithm for integrated services packet switching networks," *Proc. of SIGCOMM'96*, August 1996.

[9] M.J. Karol, Z. Liu, and K.Y. Eng. "An efficient demand-assignment multiple access protocol for wireless packet (ATM) networks," *ACM Journal on Wireless Networking*, December 1995.

[10] A. Muir and J. J. Garcia-Luna-Aceves, "Supporting real-time multimedia traffic in a wireless LAN," *Proc. SPIE Multimedia Computing and Networking 1997*, February 1997.

[11] 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.

[12] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM Transactions on Networking*, 1(3), pp. 344-357, June 1993.

[13] L. Zhang, "Virtual Clock: a new traffic control algorithm for packet switching networks," *ACM Trans. Comput. Syst.*, vol. 9, pp. 101-124, May 1991.

[14] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proc. of IEEE*, 83(10), October 1995.