Scheduling Over Non-Stationary Wireless Channels with Finite Rate Sets

Matthew Andrews and Lisa Zhang

***Abstract*— We consider a wireless basestation transmitting high-speed data to multiple mobile users in a cell. The channel conditions between the basestation and the users are time-varying and user-dependent. Our objective is to design a scheduler that determines which user to schedule at each time step. Previous work on this problem has typically assumed that the channel conditions are governed by a *stationary stochastic process*. In this setting a popular algorithm known as Max-Weight has been shown to have good performance.**

However, the stationarity assumption is not always reasonable. In this paper we study a more general worst-case model in which the channel conditions are governed by an adversary and are not necessarily stationary. In this model, we show that the non-stationarities can cause Max-Weight to have extremely poor performance. In particular, even if the set of possible transmission rates is *finite*, as in the CDMA 1xEV-DO system, Max-Weight can produce queues size that are exponential in the number of users. On the positive side, we describe a set of Tracking Algorithms that aim to track the performance of a schedule maintained by the adversary. For one of these Tracking Algorithms the queue sizes are only quadratic.

We discuss a number of practical issues associated with the Tracking Algorithms. We also illustrate the performance of Max-Weight and the Tracking Algorithms using simulation.

1 Introduction

High-speed wireless networks for data service are becoming increasingly common and optimizing such networks is the subject of active research. In particular, the *scheduling* of high-speed data is vital to the performance of modern wireless systems. In this paper we study the situation of a single basestation transmitting data to a set of mobile terminals in a cell. (See Figure 1.)

We follow a model that is motivated by Qualcomm's High Data Rate (HDR) system [6, 11]. The HDR system forms the basis for the CDMA 1xEV-DO standard and is becoming accepted as a standard model. In this model, time is slotted. In every time slot each mobile calculates the rate at which it can receive data and informs the basestation in a *Data Rate Control* (DRC) message. The basestation then determines which mobile to serve. If mobile i is chosen for service at time slot t, the rate at which data can be transmitted is called the DRC rate $r_i(t)$. The value of $r_i(t)$ depends on the quality of the channel between the basestation and mobile i. In the HDR system, at most one mobile can be served per time slot. The job of the scheduler is to choose which mobile to serve.

Both authors are with Bell Laboratories, Murray Hill, NJ. {andrews,ylz}@research.bell-labs.com. Corresponding author: Matthew Andrews. Address: 600 Mountain Avenue 2C-356, Murray Hill, NJ 07974. Tel: (908) 582-2644. Fax: (908) 582-3340.

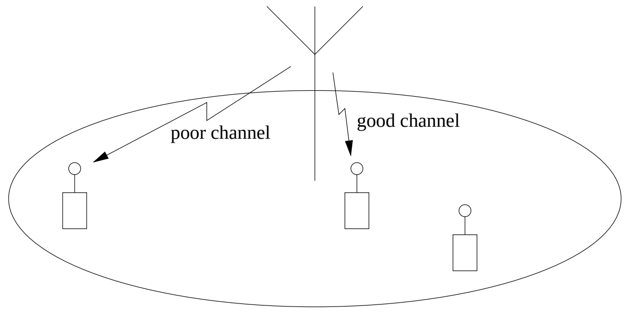


Figure 1: A wireless cell.

We emphasize that the DRC rates can be different for different users and can change over time due to user mobility and channel fading. This provides a contrast with the classical wireline scheduling problem where service rates are the same for each user and do not change over time.

A great deal of work has recently be done on this problem. For example, a popular algorithm known as *Max-Weight*, which always serves the user that maximizes the product of the DRC rate and the current queue size, has been shown in the past to work well. However, almost all previous work on wireless scheduling assumes that the channel condition between the basestation and a mobile user is governed by a *stationary* stochastic process such as an ergodic Markov chain. This is not always a realistic assumption. For example, consider a mobile user moving at a slow speed away from the basestation. In this case the quality of the channel will have a slow negative drift.

In this paper our focus is on scheduling algorithms that perform well for *non-stationary* wireless channels.

1.1 Model and preliminaries

We consider a set of N mobiles (or users) that receive (downlink) data transmissions from a basestation in a cell. Each user has a queue that is fed by an arrival process. We use the following notation.

- $r_i(t)$: the DRC rate at time t for user i, i.e. the amount of data that can be transmitted to user i during time slot t if i is chosen;
- $\vec{r}(t) = (r_0(t), \ldots r_{N-1}(t))$: the DRC vector at time t.
- $q_i(t)$: the amount of data queued for user i at the end of time slot t;

- $a_i(t)$: the amount of data that arrives for user i in time slot t.

The scheduler at the basestation chooses at most one user to service for each time slot. For example, the Max-Weight algorithm always chooses the user that maximizes $r_i(t)q_i(t)$ for service in time slot t. If user i is chosen, queue $q_i(t)$ is reduced according to $r_i(t)$. Data then arrives for all users and their queues are increased. More precisely,

$$\begin{aligned} q_i(t+1) &\leftarrow q_i(t) - \min\{q_i(t), r_i(t)\} + a_i(t) \\ q_j(t+1) &\leftarrow q_j(t) + a_j(t) \quad \forall j \neq i. \end{aligned}$$

In almost all previous work, the DRC process and the arrival process are modeled by stationary stochastic processes. More precisely, the DRC process is defined by an ergodic Markov Chain with state space M such that if the state at time t is $m \in M$ then the DRC vector $(r_0(t), \ldots, r_{N-1}(t))$ can be written as $(r_0^m, \ldots, r_{N-1}^m)$, i.e. the DRC values are dependent on the state m. In addition, the arrival process for user i is usually modeled by a Bernoulli process with intensity λ_i. In this scenario it is known (e.g. [22, 23]) that Max-Weight performs well.

However, as argued before stationary stochastic processes are not always appropriate for modeling mobility. In this paper we focus on a *worst case model*, in which the DRC process and the arrival process are generated by an adversary. By "worst case" we mean that the adversary is attempting to cause the chosen scheduling algorithm as many problems as possible. Scheduling in the context of "adversarial queueing" has received much attention recently, especially for the wireline environment [7, 2].

We assume that the scheduling process against an adversary is as follows. At the beginning of each time slot t the adversary generates a DRC vector, $\vec{r}(t)$. The scheduling algorithm then makes a decision as to which user it is going to serve. Finally the adversary injects data into the queues.

In order for the problem to be interesting we require that the system is not overloaded. More formally, we say that a sequence of DRCs and arrivals is *admissible* if there is some schedule that can serve all the data. That is, there exist integers $x_i(t) \in \{0, 1\}$ (which we call the *adversary's schedule*) such that for some window w and some loading parameter ε,

$$\sum_{t=\tau}^{\tau+w-1} a_i(t) \leq \sum_{t=\tau}^{\tau+w-1} (1-\varepsilon)x_i(t)r_i(t) \qquad \forall i, \tau, \qquad (1)$$

$$\sum_i x_i(t) = 1 \qquad \forall t \qquad (2)$$

In other words, in any window of w time steps the adversary is able to schedule the users so that each user receives more service than the amount of data injected.

However, we stress that the adversary's schedule cannot be computed by an *online* algorithm. Note that at the end of each window $[\tau, \tau + w)$, a (computationally unbounded) online algorithm could examine the data that arrived during the window and could deduce the adversary's schedule. However, it is *too late* for the server to implement this schedule. The rate vectors that appear during $[\tau+w, \tau+2w)$ could be entirely different from the rate vectors that appeared during $[\tau, \tau + w)$. Our aim is to design an *online* scheduling algorithm that keeps the queues small under all admissible DRC and arrival processes.

If $\varepsilon = 0$ then we say that the system is *critically* loaded. If the $\varepsilon > 0$ then we say that the system is *subcritically* loaded.

bits/slot	k-bits/sec
0	0
64	38.4
128	76.8
256	153.6
512	307.2
1024	614.4
1536	921.6
2048	1228.8
3072	1843.2
4096	2457.6

Figure 2: The 10 DRC rates used in the EV-DO standard.

1.2 Our results

In this paper we concentrate on non-stationary DRC processes over a finite set of DRC rates. In particular we use the 10 rates specified by the EV-DO standard. In the EV-DO standard each time slot is 1.667ms, i.e. there are 600 slots per second. The allowable values for the DRC rates in terms of *bits per slot* and in terms of *kilobits per second* are presented in Figure 2. From now on we shall refer to this rate set as the EV-DO rate set.[1]

- In Section 2 we show that for non-stationary channels Max-Weight can give extremely poor performance. In particular we give an example using the EV-DO rate set for which Max-Weight produces queues that are *exponential* in the number of users. We feel that it is important to study Max-Weight since it has been proposed so often in the literature.

A natural question to ask is whether there is an algorithm that has a polynomial bound on queue size for non-stationary channels. In [5] we showed that there is an algorithm, which we refer to as the *Quadratic Tracking Algorithm*, for which the bound on queue size is linear in the number of users and quadratic in the number of possible DRC rates.

The idea of this algorithm is that at the end of each window it calculates what the adversary's schedule was dur-

[1]We remark that in actual EV-DO systems, it is not the case that we have a separate DRC in each time slot. In reality, data is divided into physical packets of size 1024, 2048, 3072 or 4096 bits. The packet is then transmitted over multiple slots. The actual achieved transmission rate may be slightly more or less than the DRC rate due to coding issues. These details do not have a major effect on system performance and so we ignore them in this paper for the sake of readability.

ing the window. In other words, it continually works out *what it should have done in the past.* It then maintains a quadratic number of counters to measure the difference between its *own* schedule in the past and the adversary's schedule. It uses these counters to decide on its schedule in the future. For the sake of completeness we describe the Quadratic Tracking Algorithm in the Appendix.

There are a number of ways in which the Quadratic Tracking Algorithm could be improved. First, we would like a bound on queue size that is linear in the number of DRC rates. Second, the definition of the algorithm assumes that we know the value of w for which the inputs are admissible. This assumption could be unrealistic. Lastly, the calculation of the adversary's schedule at the end of each window requires the solution of an integer program. Even though it turns out that we only need to solve a linear program, it is unlikely that this could be performed in hardware. In this paper we address these issues.

- In Section 3.3 we present a heuristic *Linear Tracking* algorithm that only uses a linear number of counters for each user. As long as these counters remain bounded, the bound on queue size is linear in the number of DRC rates and does not depend on the number of users. We refer to the algorithm as a heuristic because we cannot prove that the counters remain bounded. However, we do not know of a situation where they are unbounded.
- In Section 3.4 we examine the situation where we do not know the correct value of w. If, when calculating the adversary's schedule, it is impossible to serve all the data in the window, we use linear programming to maximize a weighted sum of the data served.
- In Section 3.5 we describe a method to calculate the adversary's schedule when it is impractical to solve a linear program. We present a simple combinatorial algorithm that can approximate the solution of the linear program arbitrarily closely.
- In Section 4 we illustrate the behavior of Max-Weight and the Linear Tracking Algorithm through simulations. We first simulate the example from Section 2 to show that Max-Weight does indeed produce exponentially large queues. In contrast, the Linear Tracking Algorithm maintains extremely small queues. We then present another simple non-stationary example where once again the Linear Tracking Algorithm outperforms Max-Weight. Lastly we simulate a stationary example where Max-Weight would be expected to do well. We show that here the performance of the Linear Tracking Algorithm is close to that of Max-Weight.

Remark 1. In our model, $a_i(t)$ represents the amount of data injected into the queue for user i at time t. However, it is sometimes unwise to base a scheduling algorithm on actual queue contents since a misbehaving user could potentially gain more service by injecting an excessive amount of data into its queue. A standard way to solve this problem is for each user to have a *token buffer* into which tokens are injected at a fixed rate. We then use this token buffer for scheduling rather the real data queue. If we are able to keep the token buffer bounded then each user will receive a long-term service rate equal to its token rate. For this reason we feel that an important special case of our problem is when $a_i(t)$ is independent of time since we can then use $a_i(t)$ to represent the number of tokens arriving for user i in time slot t. For our negative results the bad examples that we study all have the property that $a_i(t)$ is constant. However, we stress that our positive results apply to the case of variable $a_i(t)$.

Remark 2. In this paper, we focus on finite rate sets only. For the case of infinite rate sets, we showed in [5] that if the DRC rates can be arbitrarily close to 0 then no online algorithm can keep the queues bounded.

1.3 Previous work

As mentioned earlier, almost all work on scheduling over time-varying channels has assumed that the channel process is stationary. Under this assumption, Max-Weight was first shown to perform well by Tassiulas and Ephremides [22, 23]. Other papers that study Max-Weight include [4, 3, 15]. Two variants of Max-Weight are Max-Delay [4, 3] and Exp [18, 19]. The Max-Delay algorithm always serves the user that maximizes $\Delta_i(t)r_i(t)$ where $\Delta_i(t)$ denotes the Head-of-Line delay for user i at time t. Exp is a more complex algorithm that provides more control over the relative delays that the users experience. For the case in which each packet has a deadline and the objective is to meet all the deadlines, [17] shows that the Earliest-Deadline-First algorithm is not always optimal (in contrast to the wireline case).

In our model we assume that each queue is fed by an arrival process and we wish to keep the queues small. In a different model that is sometimes studied, each queue has an infinite backlog and we wish to optimize some function of the service rates that each user receives. For this model a popular and widely implemented algorithm is known as Proportional Fair [24]. Proportional Fair always serves the user that maximizes $r_i(t)/R_i(t)$ where $R_i(t)$ is an exponentially smoothed average of the service rate received by user i. It is known [21, 12] that for stationary channels Proportional Fair maximizes the objective $\log R_i$. However, it is also known [1] that if (as in our model) the queues are fed by an arrival process then Proportional Fair can be unstable (i.e. the queues can grow without bound).

The problem of providing a *target* service rate to each user was studied in [9]. Algorithms for optimizing utility functions subject to fairness requirements are presented in [13, 14]. In [8], Borst examines the user-level performance of scheduling algorithms such as Proportional Fair under the assumption that each user has a finite service demand and leaves the system once this demand is satisfied.

2 Max-Weight

In this section we present examples to show that Max-Weight can give extremely poor performance when the DRC process is non-stationary. In particular we present an admissible arrival and DRC process for which Max-Weight produces queues that are *exponential* in the number of users. We restrict ourselves to the EV-DO rates shown in Figure 2 and constant arrival processes.

Let us consider a system of N users. User 0 has a special role. Users 1 through $N-1$ all act in a similar manner. The arrival process for all users is constant. The arrival rate for each user $i \geq 1$ is 1 bit per slot; the arrival rate for user 0 is 64 bits per slot. The DRC process is periodic with period 64 slots and is defined as follows:

$$r_0(t) = \begin{cases} 4096 & t = 0 \bmod 64 \\ 4096 & t = -1 \bmod 64 \\ 0 & \text{o.w.} \end{cases} \tag{3}$$

$$r_i(t) = \begin{cases} 64 & t = -i \bmod 64 \\ 128 & t = -(i+1) \bmod 64, \\ & \text{and } q_i(t) > 2q_{i+1}(t) \\ 0 & \text{o.w.} \end{cases} \tag{4}$$

	user 0	user 1	user 2	user 3
t=1				
t=2				
			0 or 128	64
		0 or 128	64	
t=w−1	4096	64		
t=w	4096			

Figure 3: DRC process that creates exponential queues for Max-Weight. The rates in the empty slots are zero.

The arrival and DRC processes described above are admissible for $w = 64$ and $\varepsilon = 0$ (i.e. the system is critically loaded). To see this, let $x_i(t) = 1$ whenever $t = -i \bmod w$ and let $x_i(t) = 0$ otherwise. (In other words, the adversary serves user i in time slots $t = -i \bmod w$.) Then,

$$\sum_{t=\tau}^{\tau+w-1} a_i(t) = \sum_{t=\tau}^{\tau+w-1} x_i(t) r_i(t) \qquad \forall i, \tau,$$

$$\sum_i x_i(t) = 1 \qquad \forall t$$

Theorem 1 *Under Max-Weight, the queue size for user $N-1$ is exponential in N.*

Proof: Let us define a sequence,

$$f_1 = 4096 \times 62 \quad \text{and} \quad f_i = 2f_{i-1} \text{ for } i \geq 2.$$

We prove the following statement by induction on the number of users. Consider a system of N users and any queue configuration at time t. The queue for the last user $q_{N-1}(t)$ is either greater than f_{N-1} or there exists a future slot $t' > t$ for which the queue $q_{N-1}(t')$ is greater than f_{N-1}.

The base case is 2 users. Note that user 1 can only be served in time slots $t = -1 \bmod w$. At time $t = -1 \bmod w$, the queue $q_0(t)$ for user 0 is at least 64×62 since the DRC rate for user 0 is 0 in the preceding 62 time slots. If the queue $q_1(t)$ for user 1 is smaller than f_1, then $q_0(t) \times 4096 > q_1(t) \times 64$. Hence, Max-Weight chooses to serve user 0 at time t. This implies that the queue for user 1 never receives service as long as it is under f_1. Therefore the queue for user 1 eventually grows to f_1.

Inductively, we assume the statement holds for a system of $N = n$ users where $n < w$ and we prove it for $N = n+1$ users. We use a second induction on the queue size of the last user n. For $k < f_n$ we show that for any slot t either $q_n(t) > k$ or $q_n(t') > k$ for some $t' > t$. This is trivially true for $k = 0$ since $a_i(t) > 0$ for all slots but $r_i(t) = 0$ for all but 2 slots out of 64. Assume now that the statement holds for all $k' < k$.

Consider a time t. By the second inductive hypothesis there exists at time $t' > t$ when $q_n(t') > k - 1$. If $q_n(t') > k$ then we are done and so we suppose $q_n(t') = k$. We claim that there must be a time t'' in the future when $t'' = -n \bmod w$ and $q_{n-1}(t'') > k/2$. If not then $r_{n-1}(t'')$ never equals 128 and so users 0 through $n-1$ act as though they are a system of n users. That is, users 0 through $n-1$ act independently of user n. Therefore, by the first inductive hypothesis, at some time $t'' > t'$, $q_{n-1}(t'') \geq f_{n-1} = f_n/2 > k/2$. During the time interval $[t', t'')$, $q_n(t)$ essentially remains constant. During the window containing t'', user $n-1$ is served with a DRC rate of 128 and user n is not served. Therefore, after time t'', $q_n(t)$ becomes larger than k. This proves the second induction which in turn immediately implies the first induction. □

Of course the arrival rates, DRC rates and window sizes we have chosen above have no particular significance. In general let us choose a window size of w and let N be at most w. The arrival rate for user 0 is $a_0 = 4096/w$ where 4096 represents the largest DRC rate; the arrival rate for each user $i > 0$ is still 1. We define the DRC process as follows.

$$r_0(t) = \begin{cases} 4096 & t = 0, -1 \bmod w \\ 0 & \text{o.w.} \end{cases}$$

$$r_i(t) = \begin{cases} w & t = -i \bmod w \\ 2w & t = -(i+1) \bmod w, \\ & \text{and } q_i(t) > 2q_{i+1}(t) \\ 0 & \text{o.w.} \end{cases}$$

Again the arrival and DRC processes are admissible for a critically loaded system. By an argument similar to that in Theorem 1 the queue for user 1 grows to $(w-2) \cdot a_0 \cdot 4096/w$ and the queue for user $i > 1$ grows to be twice that for user $i-1$. Hence, Theorem 1 holds for,

$$f_1 = (w-2) \cdot 4096^2/w^2 \quad \text{and} \quad f_i = 2f_{i-1} \text{ for } i \geq 2.$$

For the EV-DO rate set, we can set $w = 2048$. The queue can grow as high as $8184 \cdot 2^N$ for N as large as 2048.

Our proof for Theorem 1 and the description of the arrival and DRC processes are for a critically loaded system. However, if the system is subcritically loaded for a small $\varepsilon > 0$ we observe in our simulations that the statement of Theorem 1 still holds. See Section 4.

3 Tracking Algorithms

In this section we present a family of algorithms that provide much better performance for examples with non-stationary channel conditions. We refer to these algorithms as *Tracking Algorithms*. They are motivated by the following observation. Recall from the Introduction that a sequence of DRCs and arrivals is admissible if there exist integers $x_i(t) \in \{0,1\}$ (which represent the adversary's schedule) such that for some window w and some loading parameter ε,

$$\begin{aligned}\sum_{t=\tau}^{\tau+w-1} a_i(t) &\leq \sum_{t=\tau}^{\tau+w-1} (1-\varepsilon)x_i(t)r_i(t) \quad \forall i,\tau,\\ \sum_i x_i(t) &= 1 \quad \forall t.\end{aligned}$$

We let $\mathcal{R}$ be the DRC rate set and $R^{\max}$ be the largest rate from $\mathcal{R}$. The following lemma is easy to prove.

Lemma 2 *Suppose we know the values of $x_i(t)$ for which the admissibility condition is satisfied. If we always serve the user for which $x_i(t) = 1$ then the queue for user i is bounded by $(w+1)R^{\max}$.*

Unfortunately, it is clear that the adversary's schedule is not implementable for two reasons. First, we cannot calculate $x_i(t)$ online since $x_i(t)$ can depend on $a_i(t')$ and $r_i(t')$ for $t' > t$. Second, even if we did know the future arrivals and DRCs, finding a set of $x_i(t)$ that satisfy the admissibility condition for a given window involves solving an instance of the GENERALIZED ASSIGNMENT PROBLEM (GAP) which is NP-hard [20].

The key aim of the Tracking Algorithms is to "track" the adversary's schedule as close as possible whilst still being realizable. We make the algorithm online by trying to track the adversary's schedule from the past, rather than the adversary's current schedule. We circumvent the problem of NP-hardness by only calculating a fractional schedule for the adversary rather than an integral schedule.

More formally, we divide time into windows of w steps, $[0,w),[w,2w),[2w,3w)\ldots.$ (We assume for now that we know w. We address how to run the algorithm if this is not the case later on in Section 3.4.) Then at the *end* of each window $[(\ell-1)w,\ell w)$ we find *fractional* values $x_i(t)$, $(\ell-1)w \leq t < \ell w$ that satisfy,

$$\sum_{t=(\ell-1)w}^{\ell w-1} a_i(t) \leq \sum_{t=(\ell-1)w}^{\ell w-1} x_i(t)r_i(t) \quad \forall i,\ell, \tag{5}$$

$$\sum_i x_i(t) = 1 \quad \forall t, \tag{6}$$

$$0 \leq x_i(t) \leq 1 \quad \forall i,t. \tag{7}$$

Since we only require fractional values we can calculate $x_i(t)$ in polynomial time using a linear programming algorithm. Note that we know the values of $a_i(t)$ and $r_i(t)$ since we are solving the program for time steps that are *in the past.* We also note that although in hardware implementations we are unlikely to be able to solve linear programs, we can approximate the solution arbitrarily closely using simple combinatorial algorithms. See Section 3.5.

Once we have calculated the solution we use it to update a set of *counters* which measure how far away the Tracking Algorithm is from the adversary's schedule. Then, during the next window we use these counters to decide which user we are actually going to serve. There are three versions of the Tracking Algorithm that differ in how the counters are defined and used. We now describe them in detail. We assume throughout this section that the rate set $\mathcal{R}$ is finite.

3.1 Exponential Tracking Algorithm

We begin with an Exponential Tracking Algorithm. Although this algorithm has an exponential bound on queue size it introduces the main ideas. We define a counter $C_i(\vec{r})$ for each of the $|\mathcal{R}|^N$ possible DRC vectors $\vec{r}$ and for each user i. This counter is a measure of the number of the times user i is serviced by the (fractional) adversary's schedule when the DRC vector is $\vec{r}$, minus the number of times user i is serviced by the Tracking Algorithm when the DRC vector is $\vec{r}$. More precisely, at time t the Tracking Algorithm always serves user,

$$j \in \arg\max C_i(\vec{r}(t)),$$

with ties broken arbitrarily. If user j is chosen for service and $C_j(\vec{r}(t)) > 0$ then $C_j(\vec{r}(t))$ is decremented by 1. When we reach the end of the window containing time step t and have calculated the (fractional) values $x_i(t)$ we increment $C_i(\vec{r}(t))$ by $x_i(t)$ for each i. Pseudocode of the algorithm is contained in Figure 4.

We analyze the Exponential Tracking Algorithm by partitioning time steps according to the DRC vector. Formally, let,

$$T_{\vec{r}} = \{t : \vec{r}(t) = \vec{r}\}.$$

For a time window $[\tau,\tau+w)$ let p be the number of times that the DRC vector is $\vec{r}$, i.e. $p = |T_{\vec{r}} \cap [\tau,\tau+w)|$. By the choice of user to serve, either $\sum_i C_i(\vec{r})$ is decremented by p during the window or else $\sum_i C_i(\vec{r}) = 0$ at some point during the window. At the end of the window, $\sum_i C_i(\vec{r})$ is incremented by p. From this it is clear that $\sum_i C_i(\vec{r}) \leq w$. Intuitively, we have now bounded the difference between the amount of service the adversary gives to user i and the amount of service the Tracking algorithm gives to user i. It is now straightforward to prove that,

Lemma 3 *For the Exponential Tracking Algorithm the queue size for user i is upper bounded by $(w+2)R^{\max}+3w|\mathcal{R}|^N R^{\max}$.*

Proof: Suppose for the sake of simplicity that if $C_j(\vec{r}(t)) = 0$ for all j then Exponential Tracking does not serve any

```
Part I (performed at every time step $\tau$)
1  $j \leftarrow \arg\max_i C_i(\vec{r}(\tau))$
2  Serve user $j$
3  $C_j(\vec{r}(\tau)) \leftarrow \max\{C_j(\vec{r}(\tau)) - 1, 0\}$.                /* Decrement counter */

Part II (performed at the end of each window when $\tau = \ell w - 1$ for $\ell = 1, 2, \ldots$)
4  Compute a fractional adversary schedule $\{x_i(t)\}$ for the past window where $(\ell - 1)w \le t < \ell w$
5  for $t = (\ell - 1)w, \ldots, \ell w - 1$
6      for $i = 0, 1, 2, \ldots, N - 1$
7          $C_i(\vec{r}(t)) \leftarrow C_i(\vec{r}(t)) + x_i(t)$        /* Increment counters according to $\{x_i(t)\}$ */
```

Figure 4: Exponential Tracking Algorithm.

```
Part I (performed at every time step $\tau$)
1  $j \leftarrow \arg\max_i C_i(r_i(\tau))$
2  Serve user $j$
3  $C_j(r_j(\tau)) \leftarrow \max\{C_j(r_j(\tau)) - 1, 0\}$.                /* Decrement counter */

Part II (performed at the end of each window when $\tau = \ell w - 1$ for $\ell = 1, 2, \ldots$)
4  Compute a fractional adversary schedule $\{x_i(t)\}$ for the past window where $(\ell - 1)w \le t < \ell w$
5  for $t = (\ell - 1)w, \ldots, \ell w - 1$
6      for $i = 0, 1, 2, \ldots, N - 1$
7          $C_i(r_i(t)) \leftarrow C_i(r_i(t)) + x_i(t)$        /* Increment counters according to $\{x_i(t)\}$ */
```

Figure 5: Linear Tracking Algorithm. The only difference between Exponential Tracking and Linear Tracking is that the counter is defined on each DRC rate for Linear Tracking and the counter is defined on length-N DRC vectors for Exponential Tracking.

user. Let $y_j(t) = 1$ if Exponential Tracking serves user j at time t and $y_j(t) = 0$ otherwise. For any time τ let τ' be the last time that $q_i(\tau' - 1) < R^{\max}$. This implies that $q_i(\tau') \le (w + 1)R^{\max}$. We have,

$$q_i(\tau) = q_i(\tau') - \sum_{t=\tau'}^{\tau} y_i(t)r_i(t) + \sum_{t=\tau'}^{\tau} a_i(t).$$

Then by the admissibility condition,

$$q_i(\tau) \le (w+1)R^{\max} - \sum_{t=\tau'}^{\tau} y_i(t)r_i(t) + wR^{\max} + \sum_{t=\tau'}^{\tau} x_i(t)r_i(t).$$

By partitioning time according to the DRC vector we have,

$$q_i(\tau) \le (2w + 1)R^{\max} + \sum_{\vec{r}} \sum_{t \in T_{\vec{r}}: \tau' \le t < \tau} (x_i(t) - y_i(t))r_i(t).$$

The value of $\sum_{t \in T_{\vec{r}}: \tau' \le t < \tau}(x_i(t) - y_i(t))$ is essentially the difference between the value of the counter $C_i(\vec{r})$ at time τ' and the value of $C_i(\vec{r})$ at time τ. Hence $\sum_{t \in T_{\vec{r}}: \tau' \le t < \tau}(x_i(t) - y_i(t)) \le 3w$. (The reason it is $3w$ rather than w is due to the fact that the counters are only incremented at the end of a window.) In addition, at all times $t \in T_{\vec{r}}$, the value of $r_i(t)$ is equal to the ith component of $\vec{r}$. Hence,

$$\sum_{t \in T_{\vec{r}}: \tau' \le t < \tau} (x_i(t) - y_i(t))r_i(t) \le 3wR^{\max},$$

which implies $q_i(\tau) \le (2w + 1)R^{\max} + 3w|\mathcal{R}|^N R^{\max}$. □

3.2 Quadratic Tracking Algorithm

The Quadratic Tracking Algorithm was presented and analyzed in [5]. We briefly describe it in the Appendix for completeness. Pseudocode for the algorithm is contained in Figure 12. The motivation for the Quadratic Tracking Algorithm is that we would like to have a polynomial bound on queue size. For this purpose, we now define our counters on pairs of rates rather than on length-N DRC vectors. For each pair of users $i < j$ and for each pair of rates $a, b \in \mathcal{R}$ we maintain a counter, $C_{ij}(a, b)$. The following lemma is proved in [5].

Lemma 4 ([5]) *For the Quadratic Tracking Algorithm the queue size for user i is upper bounded by $wR^{\max} + 2N|\mathcal{R}|^2 R^{\max}$.*

3.3 Linear Tracking Algorithm

The Quadratic Tracking Algorithm has a much better bound on queue size than the Exponential Tracking Algorithm. However, it is still quadratic in the size of the rate set which may be unacceptable in some situations. In addition as we can see from Figure 12 the algorithm for updating the counters is somewhat involved. We now introduce a simple heuristic Linear Tracking Algorithm which only uses a linear number of counters. As long as these counters remain bounded this algorithm allocates service in a manner that is extremely close to the adversary and so it achieves good performance. The pseudocode for the linear tracking algorithm is contained in Figure 5.

For each $r \in \mathcal{R}$ and for each user i we have a counter $C_i(r)$. This is a measure of the number of the times user i is serviced by the (fractional) adversary's schedule when $r_i(t) = r$ minus the number of times user i is serviced by the Tracking Algorithm when $r_i(t) = r$.

At time t the Linear Tracking Algorithm always serves user,

$$j \in \arg\max C_i(r_i(t)),$$

with ties broken arbitrarily. If user j is chosen for service then $C_j(r_j(t))$ is decremented by 1. When we reach the end of the window containing time step t and have calculated the (fractional) values $x_i(t)$ we increment $C_i(r_i(t))$ by $x_i(t)$ for each i.

If the counters remain bounded then we immediately have a bound on queue size. The proof of the following lemma is almost identical to the proof of Lemma 3.

Lemma 5 *Suppose that during the operation of the Linear Tracking Algorithm all counters $C_i(r)$ remain between $-\alpha$ and α. Then the queue size for user i is upper bounded by* $(2w+1)R^{\max} + 2(w+\alpha)|\mathcal{R}|R^{\max}$.

3.4 Incorrect Window Size

The above algorithms rely on our ability to compute a fractional schedule for the adversary by finding a solution to the constraints (5)-(7). However, this relies on knowing a value of w for which (5)-(7) are satisfiable for all windows. (From now on we shall refer to a window for which (5)-(7) are satisfiable as a satisfiable window.) One solution is to first guess a value of w and then whenever we encounter an unsatisfiable window we simply double w. However, in reality it may be the case that the minimum value of w for which 100% of the windows are satisfiable is much larger than the minimum value of w for which 99% of the windows are satisfiable. In this case we would prefer to use the smaller window size.

To achieve this, we now have to extend the definition of our algorithms to unsatisfiable windows. For this purpose, instead of trying to satisfy (5)-(7) in window $[(\ell-1)w, \ell w)$ we now solve the following the linear program,

$$\max \quad \sum_i (d_i)^2 \lambda_i \tag{8}$$

subject to

$$d_i = \gamma_{i,\ell} + \sum_{t=(\ell-1)w}^{\ell w-1} a_i(t) \quad \forall i, \tag{9}$$

$$\lambda_i d_i = \sum_{t=(\ell-1)w}^{\ell w-1} x_i(t) r_i(t) \quad \forall i, \tag{10}$$

$$\lambda_i \leq 1 \quad \forall i, \tag{11}$$

$$\sum_i x_i(t) \leq 1 \quad \forall t, \tag{12}$$

$$x_i(t) \geq 1 \quad \forall i, t. \tag{13}$$

The parameter d_i is the amount of user-i data that we wish to serve during the window. We emphasize that d_i is a parameter, not a variable. It equals $\sum_{t=(\ell-1)w}^{\ell w-1} a_i(t)$, the data injected for user i during the window, plus $\gamma_{i,\ell}$, the *residual* user-i data that was left over from the previous window. The variable λ_i is the fraction of d_i that the adversary is able to serve during the window. We define the residual data for the next window by,

$$\gamma_{i,\ell+1} = d_i(1-\lambda_i).$$

We note that λ_i is completely determined by the values of $x_i(t)$. If $\lambda_i = 1$ for all i then all the data is served and we say that the window is satisfied. If the window cannot be satisfied then the form of the objective function means that we give preference to users with more data to serve. We keep track of the average fraction of windows that are satisfied. If this fraction exceeds a suitable threshold, say 99%, then we double the window size w, otherwise we leave the window size unchanged. We remark that if all windows are satisfied then the solutions to (8)-(13) will also be solutions to (5)-(7).

3.5 Efficiently Solving the Linear Program

Although linear programs can be solved in polynomial time, schedulers for wireless data systems are typically implemented in ASICs or DSPs. It is unreasonable to expect that a generic linear program solver could be implemented in such a device. However, using techniques of Garg and Könemann [10] (which were in turn based on earlier techniques of [16] and [25]) we can approximate (8)-(13) arbitrarily closely using a simple combinatorial algorithm. In this section we sketch the ideas.

For any dual variables u_i and v_t, define,

$$D(\vec{u}, \vec{v}) = \sum_i u_i + \sum_t v_t$$

$$\alpha(\vec{u}, \vec{v}) = \min_{i,t} \left(u_i + \frac{d_i v_i}{r_i(t)} \right).$$

Using the theory of complementary slackness, it can be shown that the dual is equivalent to,

$$\min \frac{D(\vec{u}, \vec{v})}{\alpha(\vec{u}, \vec{v})}.$$

(Note that there are no constraints in this formulation.) Our algorithm for finding an approximate solution to (8)-(13) is an iterative algorithm that works as follows.

- Initially set $x_i(t) = 0$, $u_i = 1$ and $v_t = 1$ for all i, t.
- At each iteration find the (i,t) pair that minimizes
$$\frac{1}{(d_i)^2} \left(u_i + \frac{d_i v_i}{r_i(t)} \right).$$
For this pair let $f = \max\{1, d_i/r_i(t)\}$. We then update primal and dual variables according to,
$$x_i(t) \leftarrow x_i(t) + f$$
$$u_i \leftarrow u_i(1 + \theta \cdot f \cdot r_i(t)/d_i)$$
$$v_t \leftarrow v_t(1 + \theta \cdot f),$$

for some parameter θ.

- Terminate after $(N+w)\lceil 1/\theta \log_2(1+\theta)\rceil$ iterations. The primal variables $x_i(t)$ will in general be infeasible. We create a feasible solution by setting,

$$x_i(t) \leftarrow \frac{x_i(t)}{\max\{\sum_j x_j(t), \sum_{t'} x_i(t')r_i(t')/d_i\}}$$

The following lemma follows directly from the analysis of Garg and Könemann for packing linear programs.

Lemma 6 *Let β be the optimum solution to (8)-(13). The value of the feasible solution produced by the above iterative algorithm is at least $(1-\theta)^2\beta$.*

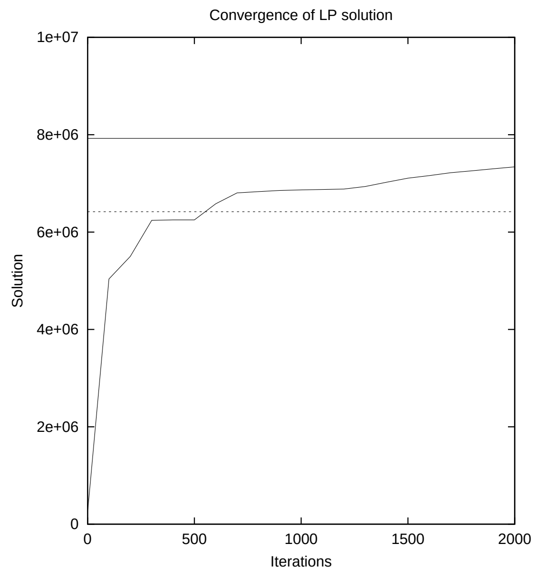


Figure 6: Convergence of the combinatorial algorithm to the optimum solution.

In Figure 6 we illustrate the convergence of the iterative algorithm for one window consisting of 60 time slots and 40 users. The value of θ is 0.1. In the figure we plot the value of the feasible solution versus the number of iterations. We also plot β and $(1-\theta)^2\beta$. We observe that the value of the feasible solution exceeds $(1-\theta)^2\beta$ after slightly more that 500 iterations.

4 Simulation Results

4.1 Non-Stationary Channels

We first validate Theorem 1 from Section 2. The simulation setup is as follows. We choose a window size $w = 62$ so that the system is subcritically loaded with $\varepsilon \approx 0.03$. In order to keep the plots simple we look at 4 users only (although it is easy to show exponential queue sizes for larger numbers of users). User 0 has constant arrivals of 64 bits per time slot and users 1 through 3 have constant arrivals of 1 bit per time slot. The DRC rates are defined in (3)-(4) and they are also illustrated in Figure 3.

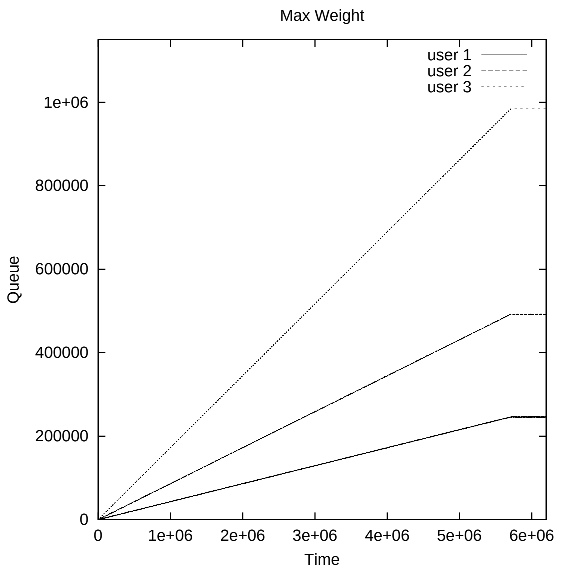


Figure 7: Exponential queues for Max-Weight.

Figure 7 shows the queue growth for users 1, 2 and 3 under Max-Weight. We can see that the queue size for user 3 is twice that of user 2, which is twice that of user 1. These queues grow continuously for the first 5.7×10^6 slots before they flatten out. The queue for user 3 grows close to 10^6.

On the contrary, the Tracking Algorithms produce much smaller queues. Figure 8 plots the total queue size over all users when Linear Tracking is implemented. As we can see, the total queue size stays below 3800 bits throughout.

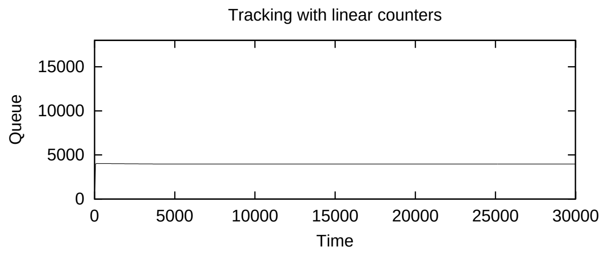


Figure 8: Small queues for Linear Tracking.

We now present a non-stationary example that is closer to reality. Consider a highway passing by a sequence of evenly spaced basestations. When traveling through a cell associated with one of the basestations, a car first experiences increasingly stronger DRCs as it approaches the basestation and later weaker DRCs as it moves farther away. Suppose that at the edge of the cell the DRC is 64 and the distance to the basestation is d. Then the DRC doubles to 128 at distance $2^{-1/3}d = 0.79d$ if we assume the signal strength is inversely proportional to the cube of distance. The following list of numbers shows the fraction of time that the car experiences each DRC as it moves towards the basestation. The same numbers apply for the situation when the car moves away from the basestation.

64	128	256	512	1024	1536	2048	3072	4096
0.21	0.16	0.13	0.10	0.05	0.03	0.04	0.03	0.25

We consider a scenario in which a stream of evenly spaced cars pass by the basestations. At any given time the number of cars per cell is 40. Each car repeats a sequence of DRCs which goes through every DRC from 64 up to 4096 and then from 4096 down to 64. The relative duration of each DRC is shown in the above chart. Suppose the cell size is 1 mile, then at a constant speed of 60 miles per hour the duration of the sequence equals 1 minute, which is $T = 60 * 600 = 36000$ slots. Each car starts the sequence with an offset of $T/40$. Suppose also each car injects at a rate of 100 bits per slot. This is an admissible injection process, since at every slot the total bits injected is $100 \times 40 = 4000$ and some user has DRC equal to 4096. Therefore, by serving this user stability can be accomplished. Figure 9 (top) is a scatter plot of the DRC rates that Max-Weight uses at each time slot. We can see that Max-Weight frequently uses rate 3072. Figure 9 (bottom) is a scatter plot of the DRC rates that Linear Tracking uses. Linear Tracking initially uses the small DRC rates but it starts to use rate 4096 exclusively around time T. The duration of the simulation is $12T$. From Figure 10, we can see that Max-Weight produces a much larger total queue size than Linear Tracking. This is consistent with the DRC rates that the two algorithms choose to serve.

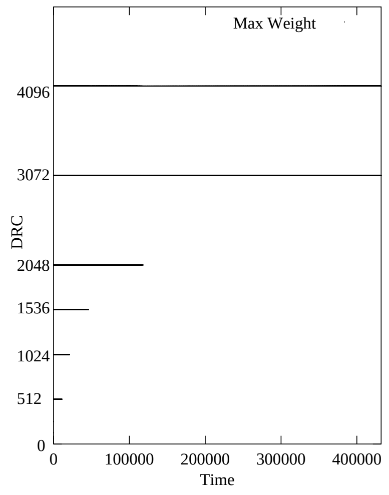
The behavior of Linear Tracking and Max-Weight we have observed remains unchanged qualitatively under a number of variations. For example with respect to T, the total duration of the DRC sequence, we have tried small values such as $T = 200$ time slots for which DRCs can change in a matter of few slots to large values such as $T = 72000$ slots for which DRCs stay unchanged for thousands of slots. We have also tested a number of users ranging from 10 to 80. In all these variations, as long as the total injected bits per slot over all users is close to 4096 and in every time slot some user has a DRC of 4096, Linear Tracking is able to accomplish stability. In contrast, Max-Weight uses smaller DRCs sufficiently often and therefore its total queuesize grows much larger.

We also considered scenarios in which the signal strength is inversely proportional to the square, or the 4th power, of distance. In this case the relative durations of the DRCs are quite different from those under the cubic rule. (The following table shows the fractions under the square rule.) However, we observed that Max-Weight and Linear Tracking are not sensitive to the exact duration of each DRC. As long as the sequence cycles through all DRCs from low to high and then from high to low, the relative performance of the two algorithms is unchanged.

64	128	256	512	1024	1536	2048	3072	4096
0.29	0.21	0.15	0.10	0.05	0.03	0.03	0.02	0.13

4.2 Stationary Channels

We conclude by considering a 40-user stationary case in which each user has a DRC trace that fluctuates around



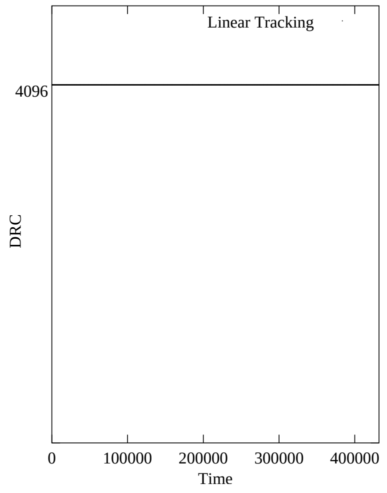


Figure 9: (Top) The DRC rates that Max-Weight chooses. (Bottom) The DRC rates that Linear Tracking chooses. (These are scatter plots. The dots appear like lines since they are so dense.)

a mean value according to 3km/h Rayleigh fading. Since this example is stationary we would expect Max-Weight to perform well. We observe in Figure 11 that Max-Weight does indeed produce smaller queues than Linear Tracking. However, the difference is much less dramatic than the difference between the two algorithms in the non-stationary cases.

5 Conclusions

In this paper we have studied the problem of scheduling over non-stationary wireless channels. We created an example showed analytically that the popular Max-Weight protocol can produce queue sizes that are exponential in the number of users. In contrast, we presented Tracking algorithms that try to keep close to the adversary's schedule and generally produce much smaller queue sizes. We also showed via simulation that, for an example of traffic moving on a highway, Max-Weight has inferior performance

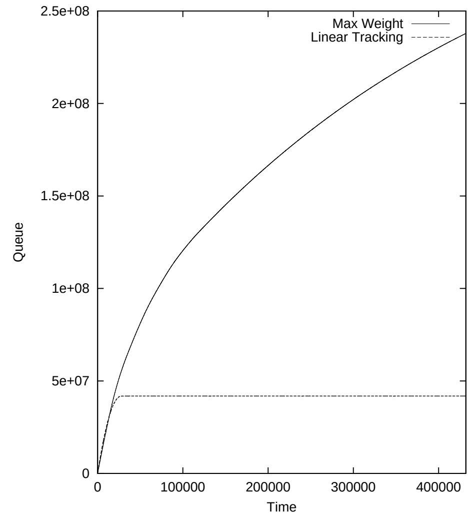


Figure 10: The total queue size produced by Max-Weight and Linear Tracking for highway traffic.

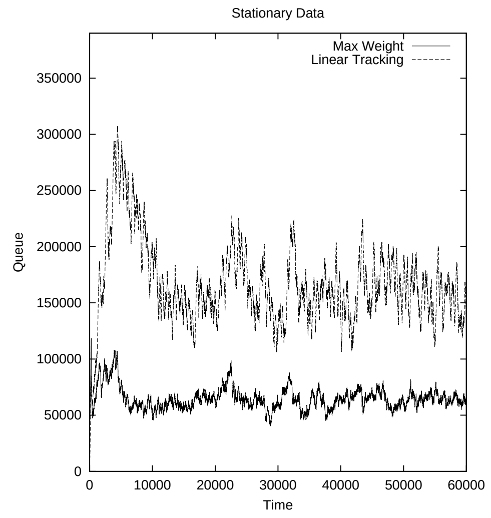


Figure 11: The total queue size produced by Max-Weight and Linear Tracking for a stationary example.

than the Linear Tracking algorithm.

We believe there are a number of open questions related to the Tracking algorithms. First, we would like to know if the counters for the Linear Tracking algorithm do indeed remain bounded. Second, are there better Tracking algorithms that keep even closer to the adversary's schedule than the algorithms that we have presented? Third, we note that the Tracking algorithms have to perform different amounts of computation in each time step since we only compute the adversary's schedule and increase the counters at the end of each window. We wonder if there is a smoother schedule that calculates a portion of the adversary's schedule in each slot. Lastly, our bad example for Max-Weight generated exponential queue sizes. It would be interesting to know if the example could be extended to an unstable scenario for Max-Weight where the queue sizes are unbounded. We note that for this to happen the DRC process could not be periodic.

At a high level, relatively little is known as to how to best characterize non-stationary wireless channels. The admissible condition presented in this paper is extremely general. It includes "periodic" behavior (such as the highway traffic) for which some form of prediction may help, as well as "adversarial" behavior (such as in a military setting) for which worst-case analysis is perhaps the most suitable. We need to better understand both types of non-stationary traffic. Of course, the real challenge is design an algorithm that performs well for all stationary and non-stationary traffic.

References

[1] M. Andrews. Instability of the proportional fair scheduling algorithm for HDR. *IEEE Transactions on Wireless Communications*, 3(5), 2004.

[2] M. Andrews, B. Awerbuch, A. Fernández, J. Kleinberg, T. Leighton, and Z. Liu. Universal stability results and performance bounds for greedy contention-resolution protocols. *Journal of the ACM*, 48(1):39–69, January 2001.

[3] M. Andrews, K. Kumaran, K. Ramanan, A. Stolyar, R. Vijayakumar, and P. Whiting. CDMA data QoS scheduling on the forward link with variable channel conditions. *Bell Labs Technical Memorandum*, April 2000.

[4] M. Andrews, K. Kumaran, K. Ramanan, A. Stolyar, R. Vijayakumar, and P. Whiting. Providing quality of service over a shared wireless link. *IEEE Communications Magazine*, February 2001.

[5] M. Andrews and L. Zhang. Scheduling over a time-varying user-dependent channel with applications to high speed wireless data. In *Proceedings of the 43nd Annual Symposium on Foundations of Computer Science*, Vancouver, Canada, November 2002.

[6] P. Bender, P. Black, M. Grob, R. Padovani, and N. Sindhushayana A. Viterbi. CDMA/HDR: A bandwidth efficient high speed data service for nomadic users. *IEEE Communications Magazine*, July 2000.

[7] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queueing theory. *Journal of the ACM*, 48(1):13–38, January 2001.

[8] S. Borst. User-level performance of channel-aware scheduling algorithms in wireless data networks. In *Proceedings of IEEE INFOCOM '03*, San Francisco, CA, April 2003.

[9] S. Borst and P. Whiting. Dynamic rate control algorithms for CDMA throughput optimization. In *Proceedings of IEEE INFOCOM '01*, Anchorage, AK, April 2001.

[10] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 300 – 309, Palo Alto, CA, November 1998.

[11] A. Jalali, R. Padovani, and R. Pankaj. Data through-

put of CDMA-HDR a high efficiency-high data rate personal communication wireless system. In *Proceedings of the IEEE Semiannual Vehicular Technology Conference, VTC2000-Spring*, Tokyo, Japan, May 2000.

[12] H. Kushner and P. Whiting. Asymptotic properties of proportional-fair sharing algorithms. In *40th Annual Allerton Conference on Communication, Control, and Computing*, 2002.

[13] X. Liu, E. Chong, and N.B.Shroff. Opportunistic transmission scheduling with resource-sharing constraints in wireless networks. *IEEE Journal on Selected Areas in Communications*, 19(10), 2001.

[14] X. Liu, E. Chong, and N.B.Shroff. A framework for opportunistic scheduling in wireless networks. *Computer Networks*, 41(4):451–474, 2003.

[15] M. Neely, E. Modiano, and C. Rohrs. Power and server allocation in a multi-beam satellite with time varying channels. In *Proceedings of IEEE INFOCOM '02*, New York, NY, June 2002.

[16] S. Plotkin, D. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Math of Operations Research*, 20:257–301, 1995.

[17] S. Shakkottai and R. Srikant. Scheduling real-time traffic with deadlines over a wireless channel. In *Proceedings of ACM Workshop on Wireless and Mobile Multimedia*, Seattle, WA, August 1999.

[18] S. Shakkottai and A. Stolyar. Scheduling algorithms for a mixture of real-time and non-real-time data in HDR. In *Proceedings of 17th International Teletraffic Congress (ITC-17)*, pages 793 – 804, Salvador da Bahia, Brazil, 2001.

[19] S. Shakkottai and A. Stolyar. Scheduling for multiple flows sharing a time-varying channel: The exponential rule. *Analytic Methods in Applied Probability*, 207:185 – 202, 2002.

[20] D. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming A*, 62:461 – 474, 1993. Also in *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1993.

[21] A. Stolyar. On the asymptotic optimality of the gradient scheduling algorithm for multiuser throughput allocation. *Operations Research*, 53:12 – 25, 2005.

[22] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 37(12):1936 – 1948, December 1992.

[23] L. Tassiulas and A. Ephremides. Dynamic server allocation to parallel queues with randomly varying connectivity. *IEEE Transactions on Information Theory*, 30:466 – 478, 1993.

[24] D. Tse. Multiuser diversity in wireless networks.

[25] N. Young. Randomized rounding without solving the linear program. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 170–178, 1995.

Appendix

We briefly describe the Quadratic Tracking algorithm which was presented and analyzed in [5]. Pseudocode for the algorithm is contained in Figure 12. For each pair of users $i < j$ and for each pair of rates $a, b \in \mathcal{R}$ we maintain a counter, $C_{ij}(a, b)$. These counters are kept constant during the window. The algorithm to decide which user to serve is simple. At time t we serve user,

$$i = \min\{i : C_{ij}(r_i(t), r_j(t)) > 0 \quad \forall j > i\}.$$

(We provide a motivation for this scheduling rule below.) The algorithm to update the counters at the end of a window is more complicated. We sketch the method here. For details see [5]. The counters $C_{ij}(a, b)$ are updated by a set of temporary counters $c_{ij}(a, b)$ which are set to zero at the start of the update process. These temporary counters are updated iteratively using a sequence of fractional schedules $S_{-1}, S_0, S_1, \ldots, S_{N-1}$. The amount of service that schedule S_i gives to user j at time t is denoted $s_{i,j}(t)$. We remark that we require $\sum_j s_{i,j}(t) = 1$.

The schedule S_{-1} is equivalent to the adversary's schedule, i.e. $s_{-1,j}(t) = x_j(t)$. The schedule S_{N-1} is equivalent to the Tracking algorithm's schedule, i.e. if the Tracking algorithm serves user j then $s_{N-1,j}(t) = 1$. The schedule S_i has the property that if the Tracking algorithm serves user j for $j \leq i$ then $s_{i,j}(t) = 1$.

The fractional schedule S_i and the temporary counters $c_{ij}(a, b)$ are defined inductively. Suppose that S_{i-1} and the temporary counters $c_{(i-1)j}(r_{i-1}, r_j)$ are already defined.

- If the Tracking algorithm serves user i at time t then we set $s_{i,i}(t) = 1$ (and hence $s_{i,j}(t) = 0$ for $j \neq i$). For all $j > i$ we decrement counter $c_{ij}(r_i(t), r_j(t))$ by $s_{i-1,j}(t)$.
- If the Tracking algorithm serves user $i' > i$ at time t then by the definition of the algorithm, it must be true that $C_{ij}(r_i(t), r_j(t)) \leq 0$ for some j. For this j we set $s_{i,j}(t) = s_{i-1,j}(t) + s_{i-1,i}(t)$ and we increment counter $c_{ij}(r_i(t), r_j(t))$ by $s_{i-1,i}(t)$. We also set $s_{i,i}(t) = 0$ and $s_{i,k}(t) = s_{i-1,k}(t)$ for $k \neq i, j$.
- If the Tracking algorithm serves user $i' < i$ at time t then we set $s_{i,i'}(t) = 1$. No counters are changed.

Once all the temporary counters have been defined, $C_{ij}(r_i(t), r_j(t))$ is incremented by $c_{ij}(r_i(t), r_j(t))$ (which may be negative). It is shown in [5] that $C_{i,j}(a, b)$ measures the number of times that schedule S_i (fractionally) serves user j minus the number of times that schedule S_{i-1} (fractionally) serves user j during the time steps that $r_i(t) = a$ and $r_j(t) = b$.

The key idea of the stability analysis is that a temporary counter is only incremented if the corresponding permanent counter is negative and it is only decremented if the corresponding permancent counter is positive. This allows us to prove that the counters remain bounded. The complete proof that the queue sizes are quadratic can be found in [5].

```
**Part I** (performed at every time step $\tau$)
1  for $i = 0, 1, 2, \ldots, N-2$
2      if $C_{ij}(r_i(\tau), r_j(\tau)) > 0$ for all $j > i$
           serve user $i$
           exit Part I
3  serve user $N-1$

**Part II** (performed at the end of each window when $\tau = \ell w - 1$ for $\ell = 1, 2, \ldots$)
4  Compute a fractional adversary schedule $\{x_i(t)\}$ for the *past* window $(\ell-1)w \leq t < \ell w$
5  Initialize temporary counters, let $c_{ij}(\cdot,\cdot) := 0 \quad \forall i, j$
6  for $t = (\ell-1)w, \ldots, \ell w - 1$
7      for $i = 0, 1, 2, \ldots, N-2$
8          Case 1: if the Tracking algorithm served user $i$ in time step $t$
               set $s_{i,i}(t) = 1$ and $s_{i,j}(t) = 0$ for $j \neq i$
               for all $j > i$, decrement $c_{ij}(r_i(t), r_j(t))$ by $s_{i-1,j}(t)$
9          Case 2: if the Tracking algorithm served user $i' > i$ in time step $t$
               there exists some $j > i$ s.t. $C_{ij}(r_i(t), r_j(t)) \leq 0$
               let $s_{i,j}(t) := s_{i-1,j}(t) + s_{i-1,i}(t)$, increment $c_{ij}(r_i(t), r_j(t))$ by $s_{i-1,i}(t)$
               let $s_{i,i}(t) := 0$
               for $k \neq i$ and $k \neq j$ let $s_{i,k}(t) = s_{i-1,k}(t)$.
10         Case 3: if the Tracking algorithm served user $i' < i$ in time step $t$
               set $s_{i,i'}(t) = 1$ and $s_{i,j}(t) = 0$ for $j \neq i'$
11 Update permanent counters, let $C_{ij}(r_i(t), r_j(t)) := C_{ij}(r_i(t), r_j(t)) + c_{ij}(r_i(t), r_j(t))$.
```

Figure 12: Quadratic Tracking Algorithm.