Scheduling Over a Time-Varying User-Dependent Channel with Applications to High Speed Wireless Data

Matthew Andrews
Bell Laboratories
and
Lisa Zhang
Bell Laboratories

In a wireless network, a basestation transmits data to mobiles at *time-varying, mobile-dependent* rates due to the ever changing nature of the communication channels. In this paper we consider a wireless system in which the channel conditions and data arrival processes are governed by an *adversary*. We first consider a single server and a set of users. At each time step t the server can only transmit data to one user. If user i is chosen the transmission rate is $r_i(t)$. We say that the system is (w, ε)-*admissible* if in any window of w time steps the adversary can schedule the users so that the total data arriving to each user is at most $1 - \varepsilon$ times the total service it receives.

Our objective is to design on-line scheduling algorithms to ensure stability in an admissible system. We first show, somewhat surprisingly, that the admissibility condition alone does not guarantee the existence of a stable online algorithm, even in a subcritical system (i.e. $\varepsilon > 0$). For example, if the nonzero rates in an infinite rate set can be arbitrarily small, then a subcritical system can be unstable for any deterministic online algorithm.

On a positive note, we present a tracking algorithm that attempts to mimic the behavior of the adversary. This algorithm ensures stability for all (w, ε)-admissible systems that are not excluded by our instability results. As a special case, if the rate set is finite, then the tracking algorithm is stable even for a critical system (i.e. $\varepsilon = 0$). Moreover, the queue sizes are independent of ε. For subcritical systems, we also show that a simpler max weight algorithm is stable as long as the user rates are bounded away from zero.

The offline version of our problem resembles the problem of scheduling unrelated machines and can be modeled by an integer program. We present a rounding algorithm for its linear relaxation and prove that the rounding technique cannot be substantially improved.

Categories and Subject Descriptors: C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*; F.2.m [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous

General Terms: Algorithms, Design, Theory

Additional Key Words and Phrases: Scheduling, stability, time-varying, user-depedent, wireless channel

Author's address:
Bell Labs, 600 Mountain Avenue, Murray Hill, NJ 07974. {andrews,ylz}@research.bell-labs.com.

1. INTRODUCTION

In the classical packet scheduling problem, it is assumed that each server has a fixed rate. If multiple queues of data are contending for the same server, the server serves one of them at the fixed rate. This is an accurate model in *wireline* networks since the electronics in most wired communication devices are designed to process data at a constant bit rate.

However, for *wireless* systems the constant rate server model no longer applies since data rates are determined by channel conditions. Consider a fixed basestation transmitting data to a set of mobile users. (See Figure 1.) If a user close to the basestation receives a signal along an unobstructed path, the amount of energy needed to transmit each bit is low. Therefore, the data rate for that user can be high. In contrast, if a user is far away or if the signal is somehow obstructed, then the data transmission requires high energy and the rate has to be low. Moreover, the transmission rate for a fixed user can even vary over time due to issues such as user mobility and Rayleigh fading (an effect that causes received power to vary on a fast time scale). Hence we have a situation where the data transmission rate is both *user dependent* and *time dependent*.

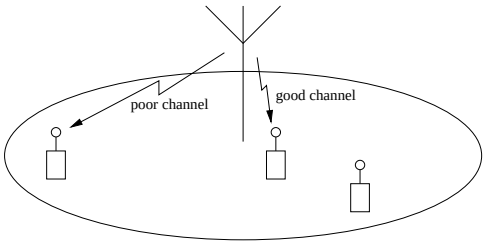

Fig. 1. A wireless system.

The problem of scheduling over variable channel conditions in wireless systems has recently received a great deal of attention. However, previous work usually assumes that channel conditions and data arrival processes can be modeled by ergodic stochastic processes such as ergodic Markov chains. In this paper we examine the problem when an *adversary* governs the channel conditions and data arrival processes. This seems more suitable since user mobility can destroy any type of stationarity in the channel conditions. Our objective is to keep the system stable, i.e. keep all the queues in the system bounded.

1.1 The Model

Our model is motivated by the Qualcomm High Data Rate (HDR) system [11] for high speed data transmission in 3rd generation Code Division Multiple Access (CDMA) wireless networks. Many wireless service providers are expected to deploy HDR in the near future. We provide a description of the HDR system in Appendix A.

We consider a single server (e.g. a basestation) and a set of N users (e.g. mobiles), each of which has its own separate queue. (See Figure 2.) Time is divided into slots.

In each slot the server can transmit data to *at most one user*. If the server selects user i in time slot t then the amount of data that can be transmitted is denoted $r_i(t)$. This is a *non-negative real number* that we refer to as the *user rate* or *channel condition*. To describe the arrival process, we use the real number $a_i(t)$ to denote the amount of data that arrives for user i in time slot t.

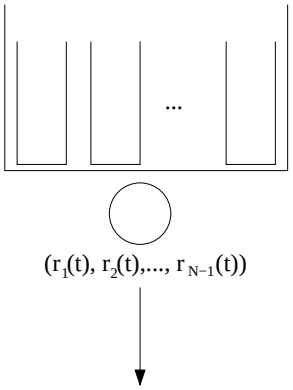

Fig. 2. A single server.

We assume that an *adversary* governs both the channel conditions and the arrivals, i.e. we have no underlying statistical information about how they behave. At each time slot t the adversary reveals the rate vector $(r_0(t), \ldots, r_{N-1}(t))$ *before* the server makes its scheduling decision. Once the decision has been made the adversary injects data, i.e. the arrival vector is revealed. Our objective is to design scheduling algorithms for the server so as to achieve *stability*, i.e. we want to maintain bounded queues at all times.[1]

In order for stability to be feasible it is necessary to impose some restrictions on the adversary. In particular, we assume that the system is (w, ε)-*admissible* for some fixed w, ε. This means in any window of w time steps, the adversary can define a schedule in which the amount of data arriving for each user is at most $(1 - \varepsilon)$ times the total service that the user receives. More formally, there exists a *binary* assignment vector $x_i(t)$ that indicates whether or not user i is served at time t and satisfies the following conditions.[2]

$$\sum_i x_i(t) = 1 \quad \forall t,$$

[1]We observe that if the rates $r_i(t)$ can be zero then bounded queues do not necessarily imply bounded delays. This is because data for queue i can be held up indefinitely if $r_i(t) = 0$ for an extended period.

[2]We note that the adversary does not necessarily schedule all the data that arrived in $[T, T + w)$ within the window $[T, T+w)$ since data for user i may arrive after the time slots that the adversary assigns to user i. However, it is easy to verify that the adversary's schedule is indeed a stable schedule.

$$\sum_{t=T}^{T+w-1} a_i(t) \leq (1-\varepsilon) \sum_{t=T}^{T+w-1} r_i(t)x_i(t) \quad \forall i, \forall [T, T+w).$$

We observe that if $r_i(t) = 1$ for all i, t then we have the standard admissibility condition for the Adversarial Queueing Model [8; 3], i.e. the total amount of data that arrives for the server in a window of size w is at most $(1-\varepsilon)w$.

Note that at the end of each window $[T, T+w)$, 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 $[T+w, T+2w)$ could be entirely different from the rate vectors that appeared during $[T, T+w)$.

1.2 Our Results

Let $\mathcal{R}$ be the set of rates that can be assigned to the users. For all our results we assume that $\mathcal{R}$ has an upper bound, since otherwise stability would be trivially impossible. Let $R^{\sup} = \sup\{r \in \mathcal{R}\}$ and $R^{\inf} = \inf\{r \in \mathcal{R} : r > 0\}$. (Note that the inclusion of 0 in $\mathcal{R}$ does not affect $R^{\inf}$).[3]

—In Section 2 we show that if the rate set $\mathcal{R}$ has certain properties then the admissibility condition does not by itself guarantee that stability is feasible. We construct two examples using an infinite rate set $\mathcal{R}$. In the first example $\varepsilon > 0$ and $R^{\inf} = 0$. In the second example $\varepsilon = 0$ and $R^{\inf} > 0$. In both cases no deterministic online scheduling algorithm can be stable.
We believe that these results are of interest since they show that our model is fundamentally different from the Adversarial Queueing Model in which the admissibility condition alone *is* sufficient to guarantee the existence of stable algorithms.

—In Section 3 we present a positive result. We construct a Tracking algorithm which attempts to keep track of the adversary's schedule and aims to give similar amounts of service to each user. The Tracking algorithm is stable for all admissible systems that are not excluded by our instability results. If the rate set is *finite*, the Tracking algorithm is stable even if $\varepsilon = 0$. Moreover, the queue sizes are independent of ε. These properties are of interest since many stability proofs rely heavily on the positivity of ε. If the rate set is *infinite*, the Tracking algorithm is stable as long as $\varepsilon > 0$ and $R^{\inf} > 0$. From our instability results it is clear that neither of these conditions can be removed.
The results of Sections 2 and 3 imply that when $\varepsilon > 0$ the existence of a stable algorithm is crucially dependent on whether or not $R^{\inf} > 0$. This is somewhat surprising since *a priori* it would be tempting to think that the existence of arbitrarily small rates has no effect on stability.

—In Section 4 we study a well-known algorithm called Max Weight in the adversarial context. At time t Max Weight always serves the user i that maximizes

[3] If $R^{\inf} = 0$ then the nonzero rates can be arbitrarily small. At first, this may seem strange since it would seem difficult to transmit less than 1 bit per slot. However, when channel conditions are poor, the data is heavily coded and the codewords could extend over multiple slots. If the number of code bits per data bit becomes arbitrarily large then the data transmission rate becomes arbitrarily small.

$r_i(t)q_i(t)$, where $q_i(t)$ is the queue size of user i at time t. We show that MAX WEIGHT is stable as long as $0 \notin \mathcal{R}$, $\varepsilon > 0$ and $R^{\inf} > 0$.
We observe that the Tracking algorithm provides stability in a wider context since it allows for $0 \in \mathcal{R}$ and even allows for $\varepsilon = 0$ as long as the rate set is finite. To model a wireless system it is important to have stability when 0 is a member of the rate set since a mobile user could temporarily move into a place where no transmission is possible.

—In Section 5 we consider the offline problem in which all data is present at time 0 and all the rate vectors are known in advance. This off-line version is NP-hard which can be shown with an easy reduction from the Partition problem. We present an algorithm for rounding the fractional solution together with lower bounds which show that our rounding techniques cannot be substantially improved.

1.3 Previous Work

The basic structure of the Qualcomm HDR system is described by Jalali et al. in [11]. The default scheduling algorithm in HDR was proposed by Tse in [21] and is known as Proportional Fair. A variation of Proportional Fair was presented in [14]. The case in which the channel conditions and arrival process can be modeled by ergodic Markov chains has been extensively studied. In this setting stable algorithms include MAX WEIGHT [5; 4; 13], MAX DELAY [5; 4] and EXP [16; 17]. MAX WEIGHT selects the user that maximizes $q_i(t)r_i(t)$, whereas MAX DELAY selects 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 keeps the user delays in desired proportions. An earlier paper [20] deals with the special case of $r_i(t) \in \{0, 1\}$. We emphasize that all these results make use of stationarity properties that do not exist in our *adversarial* setting.

In [9], Borst and Whiting examine the problem of meeting target throughput levels to each user. In [15], Shakkottai and Srikant examine a situation where $r_i(t) \in \{0, 1\}$ and show that if the objective is to meet an assigned deadline for each packet then the Earliest-Deadline-First algorithm is not always optimal (in contrast to the constant rate case).

Some recent papers have addressed the issue of routing in *dynamic networks*. In [7], Awerbuch et al. consider an adversarial network model where edges come and go over time. The aim is to route and schedule packets so that the system remains stable. A similar problem was considered by Anshelevich et al. in [6]. However, there is an important distinction between our model and the models of [7; 6], in addition to the fact that their rates can only be 0 or 1. In [7; 6], each node can transmit packets to *multiple neighbors simultaneously*, assuming that the corresponding edges are present. However, in our model a basestation can only transmit to *one mobile user at a time*. This makes stability harder to achieve.

Our analysis in Section 4 makes use of techniques similar to those used by Aiello et al. for combined routing and scheduling in adversarial networks [1]. We also observe that our model has a similar flavor to the classical scheduling problem on *unrelated machines* (e.g. [18]). In this problem when a job is assigned to a machine the increase in work on that machine is machine-dependent. Typical metrics in the classical problem include makespan, average completion time and average response

time. Our problem is somewhat different in that we address stability in an online setting.

2. INSTABILITY EXAMPLES

In this section we show that for certain rate sets $\mathcal{R}$, the admissibility condition alone does not guarantee that stability is feasible for deterministic online algorithms, even in subcritical systems where $\varepsilon > 0$. This provides a marked contrast to the traditional Adversarial Queueing Model. Recall that $R^{\inf} = \inf\{r \in \mathcal{R} : r > 0\}$. We first present an example with $\varepsilon > 0$ and $R^{\inf} = 0$ in which no deterministic online algorithm can be stable. In conjunction with the stability results of Section 3, we have the surprising fact that the existence of a stable algorithm depends crucially on $R^{\inf}$. In other words, stability can be affected by the existence of arbitrarily small positive rates. We then consider critical systems where $\varepsilon = 0$ and show that even if $R^{\inf} > 0$ no deterministic online algorithm is stable.

Intuition for the proofs. All of the instability proofs use a similar type of analysis. Consider a situation with 2 users, user 0 and user 1. At time t the online algorithm chooses a user i where $i \in \{0, 1\}$. The adversary creates a rate vector and injects data to ensure that the online algorithm has made the "wrong" decision and should have chosen user $1 - i$ at time t.

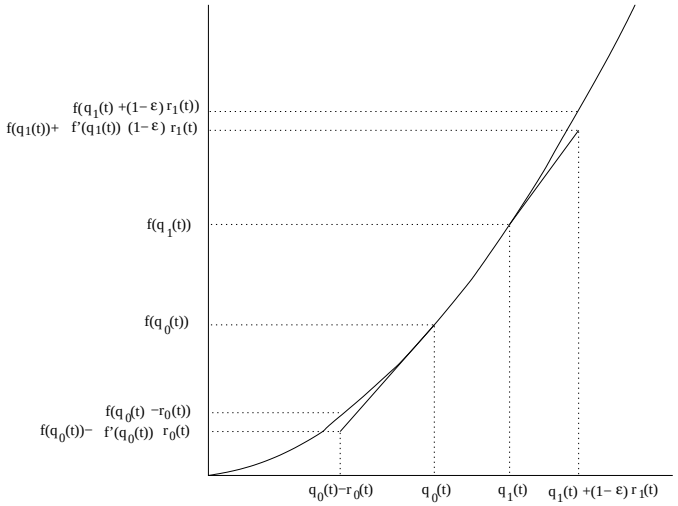

Fig. 3. The change in potential when the online algorithm serves user 0.

More precisely, let $f(\cdot)$ be a twice differentiable function that satisfies $f'(x) > 0$ and $f''(x) > 0$ for $x \geq -1$ and also satisfies $f(x) \to \infty$ as $x \to \infty$, i.e. $f(x)$ is unbounded, increasing and convex on $[-1, \infty)$. (See Figure 3.) Let $q_0(t)$ and $q_1(t)$ be the queue lengths of the two users at time t. The adversary defines a user rate vector that satisfies $r_0(t), r_1(t) \leq 1$ and,

$$f'(q_0(t))r_0(t) = f'(q_1(t))r_1(t) \quad \forall t.$$

If the online algorithm chooses to serve user i then the adversary creates arrivals for user $1-i$ only. In particular,

$$a_i(t) = 0 \text{ and } a_{1-i}(t) = (1-\varepsilon)r_{1-i}(t).$$

Clearly, this is a $(1,\varepsilon)$-admissible arrival process. However, we show that the online algorithm has unbounded queues, and is therefore unstable, by showing that the potential function $L(t) = f(q_0(t)) + f(q_1(t))$ is unbounded as $t \to \infty$. We have,

$$\begin{aligned}
L(t+1) &= f(q_i(t+1)) + f(q_{1-i}(t+1)) \\
&\geq f(q_i(t) - r_i(t)) + f(q_{1-i}(t) + (1-\varepsilon)r_{1-i}(t)) \\
&= f(q_i(t)) + f(q_{1-i}(t)) \\
&\quad -f'(q_i(t))r_i(t) + f'(q_{1-i}(t))(1-\varepsilon)r_{1-i}(t) \\
&\quad +\frac{1}{2}f''(c_i)(r_i(t))^2 + \frac{1}{2}f''(c_{1-i})((1-\varepsilon)r_{1-i}(t))^2 \\
&= L(t) \\
&\quad -\varepsilon f'(q_i(t))r_i(t) + \frac{1}{2}f''(c_i)(r_i(t))^2 \qquad (1) \\
&\quad +\frac{1}{2}f''(c_{1-i})((1-\varepsilon)r_{1-i}(t))^2, \qquad (2)
\end{aligned}$$

for some $c_i \in [q_i(t) - r_i(t), q_i(t)]$, $c_{1-i} \in [q_{1-i}(t), q_{1-i}(t) + (1-\varepsilon)r_{1-i}(t)]$. Note that the queue lengths at time $t+1$ satisfy $q_i(t+1) \geq q_i(t) - r_i(t)$ and $q_{1-i}(t+1) = q_{1-i}(t) + (1-\varepsilon)r_{1-i}(t)$. Hence, the inequality follows from the monotonicity of $f(\cdot)$. The second to last equality follows from Taylor's theorem. The last equality follows from the definition of the user rate vector. To show $L(t+1) > L(t)$, it suffices to show that the sum of the two terms (1) and (2) is positive.

For our examples, we follow this basic framework and show that $L(t)$ is unbounded as $t \to \infty$, which implies instability. We note however that for Theorem 2.2 instead of a single potential function we use a whole family of potential functions.

Before we begin we need a useful lemma about potential functions.

LEMMA 2.1. *Let $g(t) > 0$ be a function of time. Suppose there exists a function $h(x)$ and a constant c such that $g(t+c) \geq g(t) + h(g(t))$ for all t and $\inf\{h(x) : x \in [0,b]\} > 0$ for all intervals $[0,b]$. Then the function $g(t)$ is unbounded.*

PROOF. Suppose not. Let $B = \sup\{g(t) : t \geq 0\}$ and let $\delta = \inf\{h(x) : x \in [0,B]\}$. By assumption $\delta > 0$. Then for all n, $g((n+1)c) > g(nc) + \delta$. Hence there exists an n such that $g(nc) > B$. This is a contradiction. □

THEOREM 2.2. *For any deterministic[4] online scheduling algorithm the adversary can create instability with a rate set that satisfies $\varepsilon > 0$ and $R^{\inf} = 0$.*

PROOF. The example has 2 users, 0 and 1. We construct instability in phases

[4]We note that our instability examples also apply to randomized algorithms and *adaptive* adversaries. It would be interesting to know if there is a randomized algorithm that is stable against *oblivious* adversaries.

indexed by $k > 2$. Let,

$$T_k = \frac{1-\varepsilon}{3\varepsilon}\left(\frac{(k-1)(1-\varepsilon)^2}{2} - \varepsilon\right) - 1.$$

We move from phase k to phase $k+1$ at the first time that $q_0(t) > T_k$ *or* $q_1(t) > T_k$. The key properties of the sequence $\{T_k\}$ are that it is unbounded and increasing and,

$$\binom{k}{2}(1-\varepsilon)^2 - \frac{3\varepsilon k}{1-\varepsilon}(T_k+1) = k\varepsilon.$$

Suppose that we are in phase k. We use a potential function,

$$L_k(t) = (q_0(t)+1)^k + (q_1(t)+1)^k.$$

The adversary works as follows in phase k. If $q_0(t) \le q_1(t)$ then the adversary sets,

$$r_0(t) = 1 \text{ and } r_1(t) = \frac{(1+\varepsilon)(q_0(t)+1)^{k-1}}{(1-\varepsilon)(q_1(t)+1)^{k-1}}.$$

If $q_0(t) > q_1(t)$ the adversary sets,

$$r_0(t) = \frac{(1+\varepsilon)(q_1(t)+1)^{k-1}}{(1-\varepsilon)(q_0(t)+1)^{k-1}} \text{ and } r_1(t) = 1.$$

It is clear that these rates can be arbitrarily small, hence $R^{\inf} = 0$. Note also that $R^{\sup} = \frac{1+\varepsilon}{1-\varepsilon}$. If the online algorithm serves user i in slot t then the adversary sets,

$$a_i(t) = 0 \text{ and } a_{1-i}(t) = (1-\varepsilon)r_{1-i}(t).$$

It is clear that this is an admissible injection pattern. The strategy of the adversary is to always serve the opposite user to the one served by the online algorithm. We analyze the behavior of $L_k(t)$. We focus on two cases. In the first case $q_0(t) \le q_1(t)$ and the online algorithm serves user 0. In the second case $q_0(t) \le q_1(t)$ and the online algorithm serves user 1. The cases in which $q_0(t) > q_1(t)$ are similar.

Case 1. $q_0(t) \le q_1(t)$ and the online algorithm serves user 0. We have,

$$\begin{aligned} q_0(t+1) &= \max\{q_0(t)-1, 0\} \\ q_1(t+1) &= q_1(t) + (1-\varepsilon)r_1(t). \end{aligned}$$

Since the function $(x+1)^k$ is monotone increasing on $[-1,\infty)$ we have,

$$\begin{aligned} (q_0(t+1)+1)^k &\ge (q_0(t)+1-1)^k \\ (q_1(t+1)+1)^k &= (q_1(t)+1+(1-\varepsilon)r_1(t))^k. \end{aligned}$$

Therefore,

$$\begin{aligned} L_k(t+1) &= (q_0(t+1)+1)^k + (q_1(t+1)+1)^k \\ &\ge (q_0(t)+1-1)^k + (q_1(t)+1+(1-\varepsilon)r_1(t))^k \\ &\ge (q_0(t)+1)^k - k(q_0(t)+1)^{k-1} + (q_1(t)+1)^k \\ &\quad +k(q_1(t)+1)^{k-1}(1-\varepsilon)r_1(t) \text{ by convexity} \\ &= L_k(t) - k(q_0(t)+1)^{k-1} + k(1+\varepsilon)(q_0(t)+1)^{k-1} \\ &\ge L_k(t) + k\varepsilon. \end{aligned}$$

Case 2. $q_0(t) \leq q_1(t)$ and the online algorithm serves user 1. Note that since we are in phase k we must have, $q_0(t) \leq T_k$. We have,

$$\begin{aligned} q_0(t+1) &= q_0(t) + 1 - \varepsilon \\ q_1(t+1) &= \max\{q_1(t) - r_1(t), 0\}. \end{aligned}$$

We assume that ε is small enough that $\frac{1+\varepsilon}{1-\varepsilon} < 2$ (which implies $r_1(t) < 2$). Then we have,

$$\begin{aligned} (q_0(t+1)+1)^k &= (q_0(t) + 1 + 1 - \varepsilon)^k \\ (q_1(t+1)+1)^k &\geq (q_1(t) + 1 - r_1(t))^k. \end{aligned}$$

Therefore,

$$\begin{aligned} L_k(t+1) &= (q_0(t+1)+1)^k + (q_1(t+1)+1)^k \\ &\geq (q_0(t) + 1 + 1 - \varepsilon)^k + (q_1(t) + 1 - r_1(t))^k \\ &\geq (q_0(t)+1)^k + k(1-\varepsilon)(q_0(t)+1)^{k-1} + \binom{k}{2}(1-\varepsilon)^2(q_0(t)+1)^{k-2} \\ &\quad +(q_1(t)+1)^k - k(q_1(t)+1)^{k-1} r_1(t) \\ &= L_k(t) + k(1-\varepsilon)(q_0(t)+1)^{k-1} - k\frac{1+\varepsilon}{1-\varepsilon}(q_0(t)+1)^{k-1} \\ &\quad +\binom{k}{2}(1-\varepsilon)^2(q_0(t)+1)^{k-2} \\ &\geq L_k(t) + (q_0(t)+1)^{k-2}\left(\binom{k}{2}(1-\varepsilon)^2 - \frac{3\varepsilon k}{1-\varepsilon}(q_0(t)+1)\right) \\ &\geq L_k(t) + k\varepsilon \qquad \text{since } q_0(t) \leq T(k). \end{aligned}$$

The above analysis shows that $L_k(t+1) \geq L_k(t) + k\varepsilon$ in all cases. This means that there exists a time t' for which either $q_0(t') > T_k$ or $q_1(t') > T_k$, i.e. we only stay in each phase for a finite amount of time. Since the sequence $\{T_k\}$ is increasing and unbounded this means that the queue sizes cannot be bounded. Hence we have instability. □

For the special case of MAX WEIGHT the proof can be significantly simplified. We present it separately since MAX WEIGHT has received a lot of attention in the literature.

THEOREM 2.3. *The adversary can make the* MAX WEIGHT *algorithm unstable with a rate set that satisfies* $\varepsilon > 0$ *and* $R^{\inf} = 0$.

PROOF. We have 2 users, 0 and 1. The initial condition is that $q_0(0) = 1$ and $q_1(0) = 1$. In even time slots the adversary sets,

$$\begin{aligned} r_0(t) &= 1 \qquad r_1(t) = \frac{1}{(1+\varepsilon)q_1(t)}, \\ a_0(t) &= 0 \qquad a_1(t) = \frac{1-\varepsilon}{(1+\varepsilon)q_1(t)}. \end{aligned}$$

In odd time slots the adversary sets,

$$r_0(t) = 1/(1-\varepsilon) \qquad r_1(t) = 0,$$

$$a_0(t) = 1 \qquad a_1(t) = 0.$$

It is clear that this is an admissible injection pattern. The strategy for the adversary is to assign even time slots to user 1 and odd time slots to user 0.

Suppose inductively that when t is even we have $q_0(t) = 1$. (This certainly holds for $t = 0$). By the rate definition we have, $q_0(t)r_0(t) > q_1(t)r_1(t)$ which implies that MAX WEIGHT algorithm assigns time slot t to user 0. Therefore $q_0(t+1) = 0$ and $q_1(t+1) = q_1(t) + \frac{1-\varepsilon}{(1+\varepsilon)q_1(t)}$. Since user 0 now has no data to serve and $r_1(t+1) = 0$ we must have $q_0(t+2) = 1$ and $q_1(t+2) = q_1(t) + \frac{1-\varepsilon}{(1+\varepsilon)q_1(t)}$. This proves the inductive hypothesis and shows that $q_1(t)$ increases. By Lemma 2.1 $q_1(t)$ is unbounded. □

We next consider critical systems where $\varepsilon = 0$. The following theorem provides an additional situation where instability can occur.

THEOREM 2.4. *For any deterministic online scheduling algorithm the adversary can create instability with a rate set that satisfies $\varepsilon = 0$ and $R^{\inf} > 0$.*

PROOF. The example has 2 users, 0 and 1. Let $f(x) = \frac{1}{x+2} + \frac{5x}{4} - \frac{1}{2}$. We note here that $f'(x) = \frac{5(x+2)^2-4}{4(x+2)^2}$ and $f''(x) = \frac{2}{(x+2)^3}$. We analyze the system using the potential function $L(t) = f(q_0(t)) + f(q_1(t))$.

For each user i, the adversary sets,

$$r_i(t) = \frac{1}{f'(q_i(t))} = \frac{4(q_i(t)+2)^2}{5(q_i(t)+2)^2 - 4}.$$

From this we have that $\frac{4}{5} < r_i(t) \leq 1$ for $i \in \{0, 1\}$ and for all t. Therefore $R^{\inf} = \frac{4}{5} > 0$ and $R^{\sup} = 1 < \infty$.

If the algorithm decides to service user 0 at time t then the adversary injects $r_1(t)$ data into the queue for user 1. Conversely, if the algorithm decides to service user 1 at time t then the adversary injects $r_0(t)$ data into the queue for user 0. It is clear that this is an admissible injection pattern (with $\varepsilon = 0$). The adversary's strategy is to always serve the opposite user from the one served by the algorithm.

We must analyze $L(t+1)$. Suppose that the algorithm serves user 0. The other case is symmetrical. We have,

$$\begin{aligned} q_0(t+1) &\geq q_0(t) - r_0(t) \\ \Rightarrow f(q_0(t+1)) &\geq f(q_0(t) - r_0(t)). \end{aligned}$$

(This is true even if $r_0(t) \geq q_0(t)$ since $r_0(t) \leq 1$ and $f(\cdot)$ is monotone increasing on $[-1, \infty)$.) Similarly we have,

$$\begin{aligned} q_1(t+1) &\geq q_1(t) + r_1(t) \\ \Rightarrow f(q_1(t+1)) &\geq f(q_1(t) + r_1(t)). \end{aligned}$$

Now, by Taylor's theorem and the fact that $f'(\cdot)$ is increasing on $[-1, \infty)$ we have,

$$f(q_0(t) - r_0(t)) \geq f(q_0(t)) - f'(q_0(t))r_0(t) = f(q_0(t)) - 1$$

by the definition of $r_0(t)$. Similarly, by the fact that $f''(\cdot)$ is decreasing on $[-1, \infty)$ we have,

$$f(q_1(t) + r_1(t))$$

$$\begin{aligned} &\geq f(q_1(t)) + f'(q_1(t))r_1(t) + \frac{1}{2}f''(q_1(t)+r_1(t))(r_1(t))^2 \\ &= f(q_1(t)) + 1 + \frac{1}{2}f''(q_1(t)+r_1(t))(r_1(t))^2. \end{aligned}$$

by the definition of $r_1(t)$. Therefore,

$$\begin{aligned} &L(t+1) \\ &= f(q_0(t+1)) + f(q_1(t+1)) \\ &\geq f(q_0(t)) + f(q_1(t)) - 1 + 1 + \frac{1}{2}f''(q_1(t)+r_1(t))(r_1(t))^2 \\ &= L(t) + \frac{1}{2}\left(\frac{2}{(q_1(t)+r_1(t)+2)^3}\right)(r_1(t))^2 \end{aligned}$$

By the definition of $L(\cdot)$ we have,

$$q_1(t) \leq \frac{4}{5}\left(L(t) + \frac{1}{2}\right).$$

Recall also that $\frac{4}{5} < r_1(t) \leq 1$. Therefore,

$$L(t+1) \geq L(t) + \frac{1}{2}\left(\frac{2}{(\frac{4}{5}L(t) + \frac{17}{5})^3}\right)\left(\frac{4}{5}\right)^2.$$

By Lemma 2.1, $L(t)$ is unbounded. Hence by the definition of $f(\cdot)$ one of the queues is unstable. □

Instability of Proportional Fair. On a different note, we point out that the Proportional Fair algorithm is unstable. (See Appendix A for more details.) This is the algorithm used in most implementations of the Qualcomm HDR system [11] and it works as follows. At each time step t Proportional Fair serves the user that maximizes $\frac{r_i(t)}{\mu_i(t)}$ where $\mu_i(t)$ is an exponentially smoothed average of the service rate to user i up to time t. There are in fact different implementations of Proportional Fair depending on how users for which $q_i(t) < r_i(t)$ are dealt with. However, it is shown in [2] that all these implementations can be unstable. The instability holds even against a benign adversary which only injects data at constant arrival rates and which only uses two rate vectors that appear in a periodic fashion.

3. STABILITY OF A TRACKING ALGORITHM

In this section we propose a Tracking algorithm that attempts to keep track of the adversary's schedule and mimic it wherever possible. We first show that the algorithm is stable for any (w, ε)-admissible arrival process when the rate set $\mathcal{R}$ is finite. This is even true when $\varepsilon = 0$, i.e. when the system is critically loaded. If the rate set $\mathcal{R}$ is infinite, the Tracking algorithm remains stable as long as $\varepsilon > 0$ and $R^{\inf} > 0$. Hence we are stable in all situations not covered by the instability examples of Section 2.

We first consider a simplified scenario in which at the end of each window the adversary reveals to the Tracking algorithm the schedule that it used during the window. (Of course, it is now too late for the Tracking algorithm to use this schedule since the window is over.) We then consider the more realistic scenario in which we

have to *compute* the adversary's schedule at the end of each window based on the data that arrived. Unfortunately this is not feasible in general for a polynomially bounded algorithm since it involves the solution of an integer program. However, we show that in fact it is sufficient to only compute a fractional schedule for each window.

Let $X_i^{trk}(t)$ (resp. $X_i^{adv}(t)$) be the amount of service given to user i by the Tracking algorithm (resp. the adversary) up to time t. Throughout this section our key aim is to bound the difference $|X_i^{trk}(t) - X_i^{adv}(t)|$. This is because,

LEMMA 3.1. *If $|X_i^{trk}(t) - X_i^{adv}(t)| \leq \Delta$ for all t then $q_i(t) \leq 2\Delta + wR^{\sup}$ for all t. Hence the queue for user i is stable.*

PROOF. Let s be the last time before t that the queue for user i was empty. Let k be the largest integer such that $s + kw \leq t$. Then, since the system is (w, ε)-admissible,

$$\begin{aligned} q_i(t) &\leq \sum_{\tau=s}^{t-1} a_i(\tau) - X_i^{trk}(t-1) + X_i^{trk}(s-1) \\ &\leq \sum_{\tau=s}^{t-1} a_i(\tau) - X_i^{adv}(t-1) + X_i^{adv}(s-1) + 2\Delta \\ &\leq \sum_{\tau=s}^{s+kw-1} a_i(\tau) - X_i^{adv}(s+kw-1) + X_i^{adv}(s-1) + 2\Delta + wR^{\sup} \\ &\leq 2\Delta + wR^{\sup}. \end{aligned}$$

□

Suppose that the rate set is finite. Let $F = |\mathcal{R}|$ and let N be the number of users in the system. We now motivate the Tracking algorithm. Let $w = 1$ and suppose that the adversary reveals which user it serves in *every* time step *after* we have made our choice. We keep track of the most recent choice that the adversary has made for each rate vector. For example suppose that the user rate vector at time t is $\vec{r}(t)$ and the adversary chooses user i. If $t' > t$ is the next time that this rate vector appears, i.e. $\vec{r}(t) = \vec{r}(t')$, then we choose user i to serve at time t'. In this way, the number of slots that we have assigned to user i never differs from the number of slots the adversary has assigned to user i by more than F^N, the total number of rate vectors. Therefore the difference in service assigned to user i by the two schedules is at most $F^N \cdot R^{sup}$. By Lemma 3.1 we have stability.

The downside of the above proposal is that the difference in service can be exponential in the number of users. In Section 3.1 we present the Tracking algorithm and show that the difference between its service to user i and the adversary's service to user i is at most $N \cdot F^2 \cdot R^{\sup}$. In Section 3.2 we consider the case of general w in which the adversary reveals its schedule at the end of each window. The difference in service is now $N \cdot F^2 \cdot R^{\sup} \cdot w$. In Section 3.3 we no longer assume that the adversary reveals its schedule at the end of each window. The Tracking algorithm now has to compute the schedule based on the data that arrived. We show that the algorithm does not need to solve an integer program. It is able to track a fractional

schedule for each window. In Section 3.4 we conclude with a discussion of infinite rate sets.

3.1 Tracking Algorithm with Window Size 1

We define the Tracking Algorithm in Figure 4 using a sequence of schedules, $S_{-1}, S_0, \ldots, S_{N-1}$, where S_{-1} is the adversary's schedule and S_{N-1} is the schedule of the Tracking algorithm. The purpose of the intermediate schedule S_i is to "track" schedule S_{i-1} and decide when the Tracking algorithm should serve user i. In particular, the Tracking algorithm serves user i only if S_i serves user i. Whether or not S_i serves i at time t depends on the positivity of a set of counters defined below. For rate pair (a, b) and $0 \leq i < j < N$, consider all the time steps $t' < t$ at which $r_i(t') = a$ and $r_j(t') = b$. We use a counter $C_{i,j}(a, b)$ to denote the number of times that schedule S_i serves user j minus the number of times that schedule S_{i-1} serves user j during these time steps. The basic rule is that the Tracking algorithm does not serve user i at time unless $C_{i,j}(r_i(t), r_j(t)) > 0$ for all $j > i$. Initially, all counters are set to 0.

Figure 4 describes the Tracking algorithm for $w = 1$. In the algorithm description, $S_i(t)$ denotes the user served by S_i at time t and $C(r_i(t), r_j(t))$ is a shorthand for $C_{i,j}(r_i(t), r_j(t))$. In part I we loop through i and decide whether or not schedule S_i should serve user i. The intuition for the algorithm is that with respect to the current rate values, we always want the service that S_i has given user j to be *more* than the service that S_{i-1} has given user j, for all $j > i$. Hence schedule S_i can only serve user i if it is already "ahead" of S_{i-1} in service to user j for the current rate values, i.e. $C(r_i(t), r_j(t))$, for all $j > i$. This will allow us to prove that the Tracking algorithm is never too far behind the adversary in its service to any user. If schedule S_i does serve user i then schedule S_j also serves user i for all $j > i$.

For $i < N - 1$, $S_i(t)$ may not be defined in part I of the algorithm and so we define it in part II after the adversary reveals its choice. If $S_{i-1}(t) > i$ then we simply let $S_i(t)$ be the same as $S_{i-1}(t)$. If $S_{i-1}(t) = i$ then S_i serves a user $j > i$ that prevented S_i serving user i in part I. (In particular, schedule S_i is not "ahead" of S_{i-1} in service to user j for the current rate values.) In part II we also update the necessary counters. We shall see below that no counter is ever negative, i.e. we maintain the invariant that with respect to the current rate values, the service that S_i has given user j is *more* than the service that S_{i-1} has given user j, for all $j > i$.

In Lemma 3.2 we verify that the Tracking algorithm is well defined. We then prove the stability of the algorithm in Lemma 3.4 and Theorem 3.5.

LEMMA 3.2. *The Tracking algorithm is well defined. For every t,*

(1) $S_{N-1}(t)$ is computed before the adversary reveals its choice $S_{-1}(t)$;
(2) $S_i(t)$ is defined for all i.

PROOF. Since $S_{N-1}(t)$ is computed in Part I of the algorithm, the Tracking algorithm decides which user to serve at each time t before it knows the adversary's choice at time t. We show item 2 by induction on i. Since $S_{-1}(t)$ is computed the basis holds. Inductively, if $S_i(t)$ is `undefined` after Part I, then $S_{i-1}(t)$ has to be at least i. Hence, Case 2 and Case 3 in Part II cover all possible situations and so $S_i(t)$ is defined for all i by the end of Part II. □

The following is a useful fact about the Tracking algorithm.

FACT 3.3. *If $j < i$ then $S_i(t)$ serves user j if and only if $S_{i-1}(t)$ serves j.*

We also have,

LEMMA 3.4. *For $a \in \mathcal{R}$, $b \in \mathcal{R}$ and $0 \leq i < j < N$, the counter $C_{i,j}(a,b)$,*

(1) *remains either 0 or 1;*
(2) *counts the number of times that S_i serves j minus the number of times that S_{i-1} serves j during the time steps for which $r_i(t) = a$ and $r_j(t) = b$.*

PROOF. All counters are initialized to 0. To prove item 1, we assume the statement holds inductively at all times before t. In part I of the algorithm, we define $S_i(t)$ to be i if all the counters $C(r_i(t), r_j(t))$ for $j > i$ are positive. (See Line 3.) By induction, these counters are all 1. Hence, when we decrement a counter in Case 1, we decrement it from 1 to 0. In Case 2 when $S_i(t)$ is `undefined` we find a counter that is at most zero and increment it by 1. By induction, this counter has to be zero before the increment and therefore becomes 1 after the increment.

To see item 2, assume it holds inductively before time t. Let us examine how the counters are updated. Suppose $S_i(t) \leq i$, (i.e. $S_i(t)$ is defined in Part I). By Fact 3.3 the only situation in which $S_i(t) \neq S_{i-1}(t)$ is when $S_i(t) = i$ and $S_{i-1}(t)$ equals some $j > i$. This is Case 1 of the Tracking algorithm and the counter $C_{i,j}(r_i(t), r_j(t))$ is decremented. Suppose $S_i(t) > i$, (i.e. $S_i(t)$ is defined in Part II). In Case 2 of the algorithm, $S_i(t)$ equals some $j > i$, $S_{i-1}(t) = i$ and the corresponding counter is incremented. In Case 3 of the algorithm, $S_i(t) = S_{i-1}(t)$ and no counters are changed. In all cases item 2 is maintained. □

THEOREM 3.5. *For each user j, the difference between the Tracking algorithm's service to user j and the adversary's service to user j is at most $N \cdot F^2 \cdot R^{\sup}$. By Lemma 3.1, every queue length is bounded by $2N \cdot F^2 \cdot R^{\sup} + R^{\sup}$. Hence, the system is stable and the queue length is independent of ε.*

PROOF. For each user j let us bound the difference in service it receives from the adversary's schedule S_{-1} and the Tracking algorithm's schedule S_{N-1}. By Fact 3.3 if $i > j$ then schedule S_i serves user j if and only if schedule S_{i-1} serves user j. Hence, it suffices to bound the difference in service that user j receives from S_j and S_{-1}.

We first define $C_{i,j} = \sum_{a,b} C_{i,j}(a,b)$ for $i < j$. (Recall that $C_{i,j}(a,b)$ is not defined for $i \geq j$.) By Lemma 3.4, $C_{i,j}$ is the number of times that S_i serves j minus the number of times that S_{i-1} serves j over all time steps. Furthermore, $0 \leq C_{i,j} \leq F^2$. Therefore, for any user j, the amount of service schedule S_{j-1} gives to user j minus the amount of service schedule S_{-1} gives to user j is at least 0 and at most,

$$\sum_{i=0}^{j-1} C_{i,j} \cdot R^{\sup} \leq j \cdot F^2 \cdot R^{\sup}.$$

The difference between the service that S_j gives to user j and the service that S_{j-1} gives to j can be bounded as follows. Consider user i. If $i < j$ then S_j serves i

Time t
Part I
1 Let $S_i(t) :=$`undefined` for $i = 0, \ldots, N-1$.
2 for $i = 0, 1, 2, \ldots, N-2$
/* S_i only serves user i if it is ahead of S_{i-1} in service to all $j > i$ */
3 if $C(r_i(t), r_j(t)) > 0$ for all $j > i$
let $S_i(t) := i$
let $S_j(t) := i$ for all $j > i$
exit Part I
4 Let $S_{N-1}(t) := N-1$.

Part II
5 Adversary reveals $S_{-1}(t)$.
/* Define $S_i(t)$ if it is `undefined` and update counters */
6 for $i = 0, 1, 2, \ldots, N-2$
7 Case 1: if $S_i(t) = i$ and $S_{i-1}(t) > i$
decrement $C(r_i(t), r_j(t))$ by 1, where $j = S_{i-1}(t)$
8 Case 2: if $S_i(t) =$ `undefined` and $S_{i-1}(t) = i$
there exists some $j > i$ s.t. $C(r_i(t), r_j(t)) \leq 0$
let $S_i(t) := j$ for any such j
increment $C(r_i(t), r_j(t))$ by 1.
9 Case 3: if $S_i(t) =$`undefined` and $S_{i-1}(t) > i$
let $S_i(t) := S_{i-1}(t)$.

Fig. 4. Tracking algorithm for window size 1.

if and only if S_{j-1} serves i. If $i > j$ then the number of times S_j serves i minus the number of times S_{j-1} serves i is $C_{j,i}$. Hence, the number of times S_j serves j minus the number of times S_{j-1} serves j is $-\sum_{i>j} C_{j,i}$. Therefore, the amount of service schedule S_j gives to j minus the amount of service S_{j-1} gives to j is at most 0 and at least,

$$-\sum_{i=j+1}^{N-1} C_{j,i} \cdot R^{\sup} \geq -(N-1-j) \cdot F^2 \cdot R^{\sup}.$$

The above results imply that the difference between the amount of service that the Tracking algorithm gives to user j and the amount of service that the adversary gives to user j is at most,

$$\max\{j \cdot F^2 \cdot R^{\sup}, (N-1-j) \cdot F^2 \cdot R^{\sup}\}.$$

□

3.2 Tracking with Window Size w

We now modify our algorithm for the case of general w. We partition time into windows of w time steps. At the end of each window the adversary reveals the choices that it made during the window. However, we continue to make scheduling

decisions every time step. The modifications to the Tracking algorithm are minor and are presented in Figure 5.

For each window, we execute Part I of the algorithm w times, once at each time step in the window. We use the same rule as before to decide whether or not $S_i(t)$ serves user i. At the end of the window the adversary reveals its choices and we execute Part II of the algorithm w times. We again consider three cases to update counters and decide which user $S_i(t)$ serves if $S_i(t)$ is `undefined` in Part I. The difference now is that we increment and decrement a set of temporary counters $c(r_i(t), r_j(t))$ during the execution of Part II. These temporary counters are initialized to zero at the beginning of Part II and the permanent counters $C(r_i(t), r_j(t))$ are updated using the temporary counters only at the end of Part II. The reason for this additional complexity is that we want the counters used by Part I to remain constant during the window. The following statements are analogous to Lemma 3.4 and Theorem 3.5.

Part I (performed at every time step t)
1 Part I is the same as in Figure 4.

Part II (performed at the end of window ℓ)
2 Initialize temporary counters, let $c(\cdot,\cdot) := 0$
3 for time steps $\ell w, \ell w + 1, \ldots, (\ell+1)w - 1$
4 execute Part II and increment/decrement $c(\cdot,\cdot)$ only.
5 Update permanent counters, $C(r_i(t), r_j(t)) := C(r_i(t), r_j(t)) + c(r_i(t), r_j(t))$.

Fig. 5. Tracking algorithm for window size w.

LEMMA 3.6. *For $a \in \mathcal{R}$, $b \in \mathcal{R}$ and $i < j$ the expression $C_{i,j}(a,b) + c_{i,j}(a,b)$,*

(1) remains in the range $[-w, w]$.

(2) counts the number of times that S_i serves j minus the number of times that S_{i-1} serves j during the time steps for which $r_i(t) = a$ and $r_j(t) = b$.

PROOF. To prove item 1, we assume the statement holds inductively at the beginning of a window. During the window, a temporary counter is decremented only when the corresponding permanent counter is positive and it is decremented at most w times. A temporary counter is incremented only when the corresponding permanent counter is non-positive and it is incremented at most w times. Item 2 holds as in Lemma 3.4. □

THEOREM 3.7. *For each user j, the difference between the Tracking algorithm's service to user j and the adversary's service to user j is at most $N \cdot F^2 \cdot R^{\sup} \cdot w$. By Lemma 3.1, every queue length is bounded by $2N \cdot F^2 \cdot R^{\sup} \cdot w + w \cdot R^{\sup}$. Hence, the system is stable and the queue length is independent of ε.*

PROOF. As in Theorem 3.5, it suffices to bound the difference in service that a user j receives from schedule S_j and S_{-1}. We define $C_{i,j} = \sum_{a,b}(C_{i,j}(a,b) + c_{i,j}(a,b))$ for $i < j$ and so $C_{i,j}$ is the number of times that S_i serves j minus the

number of times that S_{i-1} serves j over all time steps. By Lemma 3.6, $|C_{i,j}|$ is at most $F^2 \cdot w$. Following the same argument as in Theorem 3.5, we obtain that the number of times schedule S_{j-1} serves j minus the number of times S_{-1} serves j equals,

$$\sum_{i=0}^{j-1} C_{i,j}.$$

The number of times schedule S_j serves j minus the number of times schedule S_{j-1} serves j equals,

$$-\sum_{i=j+1}^{N-1} C_{j,i}.$$

Therefore the difference between the amount of service that the Tracking algorithm gives to user j and the amount of service that the adversary gives to user j is at most,

$$\sum_{i=0}^{j-1} |C_{i,j}| \cdot R^{\sup} + \sum_{i=j+1}^{N-1} |C_{j,i}| \cdot R^{\sup} \leq N \cdot w \cdot F^2 \cdot R^{\sup}.$$

□

3.3 Computing the Adversary's Schedule

In this section we no longer assume that the adversary reveals its schedule at the end of each window of length w. The Tracking algorithm now has to compute the schedule based on the rate vectors and the data arrivals during the window. Fortunately, we are able to redefine the Tracking algorithm so that it tracks a fractional schedule rather than an integer schedule. Hence we only need to solve a linear program at the end of each window rather than an integer program. At the end of window ℓ, the linear program is as follows.

$$\begin{aligned}
0 \leq x_j(t) \leq 1 \quad & \text{for } 0 \leq j < N, \ell w \leq t < (\ell+1)w, \\
\sum_j x_j(t) = 1 \quad & \text{for } w \leq t < (\ell+1)w, \\
\sum_{t=\ell w}^{(\ell+1)w-1} a_j(t) \leq (1-\varepsilon) \sum_{t=\ell w}^{(\ell+1)w-1} r_j(t) x_j(t) \quad & \text{for } 0 \leq j < N.
\end{aligned}$$

The new version of the Tracking algorithm is presented in Figure 6. We specify each schedule $S_i(t)$ using an n-tuple $(s_{i,0}(t), s_{i,1}(t), \ldots, s_{i,N-1}(t))$, where $\sum_j s_{i,j}(t) = 1$ and $s_{i,j}(t) \geq 0$. The value of $s_{i,j}(t)$ represents the fractional assignment of time slot t to user j by schedule S_i. Let e_j be the vector that has a 1 in the jth position and 0 elsewhere. We use the expression $S_i(t) = e_j$ to denote the fact that schedule S_i assigns the *entire* time slot t to user j. For some feasible solution x of the above linear program we set $s_{-1,j}(t) = x_j(t)$. This defines the adversary's schedule $S_{i-1}(t)$. The schedule $S_{N-1}(t)$ is computed in Part I of the algorithm. It is easy to see that the Tracking algorithm is still well defined. In particular,

LEMMA 3.8. *For every t, $S_{N-1}(t)$ is computed before the adversary reveals its*

Part I (performed at every time step t)
1 Let $S_i(t)$:=`undefined` for $i = 0, \ldots, N-1$.
2 for $i = 0, 1, 2, \ldots, N-2$
3 if $C(r_i(t), r_j(t)) > 0$ for all $j > i$
let $S_j(t) := e_i$ for $j \geq i$
exit Part I
4 Let $S_{N-1}(t) := e_{N-1}$.

Part II (performed at the end of window ℓ)
5 Compute a feasible fractional schedule S_{-1} for the adversary
6 Initialize temporary counters, let $c(\cdot, \cdot) := 0$
7 for $t = \ell w, \ell w + 1, \ldots, (\ell+1)w - 1$
/* Define $S_i(t)$ if it is `undefined` and update counters */
8 for $i = 0, 1, 2, \ldots, N-2$
9 Case 1: if $S_i(t) = e_i$
for all $j > i$, decrement $c(r_i(t), r_j(t))$ by $s_{i-1,j}(t)$
10 Case 2: if $S_i(t) =$ `undefined`
there exists some $j > i$ s.t. $C(r_i(t), r_j(t)) \leq 0$
let $s_{i,j}(t) := s_{i-1,j}(t) + s_{i-1,i}(t)$, increment $c(r_i(t), r_j(t))$ by $s_{i-1,i}(t)$
let $s_{i,i}(t) := 0$
for $k \neq i$ and $k \neq j$ let $s_{i,k}(t) = s_{i-1,k}(t)$.
11 Update permanent counters, $C(r_i(t), r_j(t)) := C(r_i(t), r_j(t)) + c(r_i(t), r_j(t))$.

Fig. 6. Tracking algorithm against a fractional adversary.

choice $S_{-1}(t)$. Moreover, $S_{N-1}(t) = e_i$ for some i. Hence S_{N-1} is an integral schedule.

Fact 3.3 remains true and can be restated as follows.

FACT 3.9. *If $j < i$, then $s_{i,j}(t) = s_{i-1,j}(t)$.*

PROOF. If $s_{j,j} = 1$, then Part I of the algorithm defines $s_{i,j} = 1$ for all $i \geq j$. Suppose $s_{j,j} = 0$. If Part I defines S_{j+1}, then $S_{j+1} = e_k$ for some $k \neq j$ which implies $s_{j+1,j} = 0$. If Part II defines S_{j+1}, then $s_{j+1,j}$ is defined to be $s_{j,j}$. This argument holds inductively for all $i \geq j+1$. □

The counters still have the same meaning except that the number of times a user is served can be fractional. The following lemma is almost identical to Lemma 3.6.

LEMMA 3.10. *For $a \in \mathcal{R}$, $b \in \mathcal{R}$ and $0 \leq i < j < N$ the expression $C_{i,j}(a,b) + c_{i,j}(a,b)$,*

(1) *remains in the range of $[-w, w]$;*
(2) *counts* fractionally *the number of times that S_i serves j minus the number of times that S_{i-1} serves j during the time steps for which $r_i(t) = a$ and $r_j(t) = b$.*

PROOF. To prove item 1, we assume the statement holds inductively at the beginning of a window. During the window, a temporary counter is decremented

only when the corresponding permanent counter is positive. It is decremented at most w times and by at most 1 every time. (See Case 1.) A temporary counter is incremented only when the corresponding permanent counter is non-positive. It is incremented at most w times and by at most 1 every time. (See Case 2.)

To see item 2, assume it holds inductively at all times before t. We examine how $S_i(t)$ differs from $S_{i-1}(t)$. Suppose $S_i(t)$ is defined in Part I. If $S_i(t) = e_j$ for some $j < i$ then $S_{i-1}(t)$ must be e_j as well. No counter is updated. If $S_i(t) = e_i$ then $s_{i,j}(t) = 0$ for $j > i$. Hence the counters $c(r_i(t), r_j(t)$ for $j > i$ are decremented by $s_{i-1,j}(t)$ in Case 1 of Part II. If $S_i(t)$ is defined in Part II then the difference between $S_i(t)$ and $S_{i-1}(t)$ is that $s_{i,i}(t)$ is set to 0 and $s_{i,j}(t)$ is incremented by $s_{i-1,i}(t)$ for some $j > i$. For this particular j, the counter $c(r_i(t), r_j(t))$ is incremented accordingly. □

Using Fact 3.9 and Lemma 3.10 we can show that the statement of Theorem 3.7 still holds. We obtain,

THEOREM 3.11. *If the rate set $\mathcal{R}$ is finite, the Tracking algorithm is stable for any (w, ε)-admissible system. This includes critical systems in which $\varepsilon = 0$.*

3.4 Infinite Number of Rates

For the case of an infinite rate set $\mathcal{R}$, the stability of the Tracking algorithm can be preserved as long as $R^{\inf}$ is positive and ε is positive. We define,

$$\gamma_k = R^{\sup}(1 - \varepsilon/2)^k, \qquad \text{for } 0 \le k \le \left\lceil \log_{1-\varepsilon/2} \frac{R^{\sup}}{R^{\inf}} \right\rceil,$$

and new rates $\tilde{r}_i(t)$, by,

$$\tilde{r}_i(t) = \begin{cases} 0 & \text{if } r_i(t) = 0 \\ \gamma_k & \text{if } \gamma_k \le r_i(t) < \gamma_{k+1}. \end{cases}$$

Since $\tilde{r}_i(t) \ge r_i(t)(1 - \varepsilon/2)$, any (w, ε)-admissible arrival process with respect to rates $r_i(t)$ is $(w, \varepsilon/2)$-admissible with respect to the new rates $\tilde{r}_i(t)$. We apply the Tracking algorithm to these new rates. We conclude,

THEOREM 3.12. *If the rate set $\mathcal{R}$ is infinite, the Tracking algorithm is stable for any (w, ε)-admissible system as long as $R^{\inf}$ and ε are both positive.*

4. STABILITY OF MAX WEIGHT

In this section we consider the MAX WEIGHT algorithm. Recall that at each time step t, MAX WEIGHT serves the user i that maximizes the product $q_i(t)r_i(t)$ where $q_i(t)$ is the queue length of user i at time t. In the following we show that the MAX WEIGHT algorithm is stable for many (w, ε)-admissible systems. However, it is unknown whether MAX WEIGHT is stable when $0 \in \mathcal{R}$ or when $\mathcal{R}$ is finite and $\varepsilon = 0$. This distinguishes the Tracking algorithm from MAX WEIGHT since the Tracking algorithm is known to be stable in these two cases. In particular, in the wireless context it is important to be able to handle cases when $0 \in \mathcal{R}$. Another distinction is that for MAX WEIGHT the queue lengths are dependent on $1/\varepsilon$.

THEOREM 4.1. MAX WEIGHT *is stable for any (w, ε)-admissible system if $\varepsilon > 0$, $R^{\inf} > 0$ and $0 \notin \mathcal{R}$.*

PROOF. Our analysis of MAX WEIGHT is standard. Consider the potential function $L(t) = \sum_i (q_i(t))^2$. We show that if the queues are sufficiently large then this potential function has negative drift.

Let $m_i(t) \in \{0, 1\}$ indicate whether or not the max weight algorithm serves user i at time t. For simplicity, in the following analysis we sometimes drop the bounds on summation. If $q_i(t_0) \geq \sum_{t=t_0}^{t_0+w-1} m_i(t) r_i(t)$ then

$$q_i(t_0+w)^2 = \left(q_i(t_0) + \sum_{t=t_0}^{t_0+w-1} a_i(t) - \sum_{t=t_0}^{t+w-1} m_i(t) r_i(t) \right)^2,$$

and if $q_i(t_0) < \sum_{t=t_0}^{t_0+w-1} m_i(t) r_i(t)$ then

$$\begin{aligned} q_i(t_0+w)^2 &\leq \left(q_i(t_0) + \sum_{t=t_0}^{t_0+w-1} a_i(t) \right)^2 \\ &\leq \left(\sum_{t=t_0}^{t+w-1} m_i(t) r_i(t) + \sum_{t=t_0}^{t_0+w-1} a_i(t) \right)^2. \end{aligned}$$

Therefore in both cases we have,

$$\begin{aligned} & L(t_0+w) - L(t_0) \\ = & \sum_i q_i(t_0+w)^2 - \sum_i q_i(t_0)^2 \\ \leq & \sum_i \left(q_i(t_0) + \sum_t a_i(t) - \sum_t m_i(t) r_i(t) \right)^2 \\ & \qquad + \sum_i \left(\sum_t m_i(t) r_i(t) + \sum_t a_i(t) \right)^2 - \sum_i q_i(t_0)^2 \\ \leq & \; 2 \sum_i \left(\sum_t a_i(t) \right)^2 + 2 \sum_i \left(\sum_t m_i(t) r_i(t) \right)^2 \\ & \qquad + 2 \sum_i q_i(t_0) \left(\sum_t a_i(t) - \sum_t m_i(t) r_i(t) \right) \end{aligned} \tag{3}$$

Since the arrival process is (w, ε)-admissible, there exists a set of $x_i(t) \in \{0, 1\}$ that satisfies,

$$\sum_{t=t_0}^{t_0+w-1} a_i(t) \leq (1-\varepsilon) \sum_{t=t_0}^{t_0+w-1} x_i(t) r_i(t) \quad \forall i \quad \text{and} \quad \sum_i x_i(t) = 1 \quad \forall t.$$

Hence, the first two terms of (3) can be upper bounded by a function of $R^{\sup}$, w and N, the number of users. Let c_1 denote this upper bound. To bound the third term of (3) we note that,

$$q_i(t) - wR^{\sup} \leq q_i(t_0) \leq q_i(t) + wR^{\sup} \quad \text{for} \quad t_0 \leq t \leq t_0 + w.$$

Hence,

$$\begin{aligned} & L(t_0+w) - L(t_0) \\ \leq\ & c_1 + 2wR^{\sup} \sum_i \sum_t (a_i(t) + m_i(t)r_i(t)) \\ & \quad + 2\sum_t \sum_i (q_i(t)(1-\varepsilon)x_i(t)r_i(t) - q_i(t)m_i(t)r_i(t)) \end{aligned}$$

Again the second term of the above expression can be upper bounded by a function of $R^{\sup}$, w and N. Let c_2 denote this upper bound. To bound the third term, we note that by the definition of MAX WEIGHT, $\sum_i q_i(t)r_i(t)m_i(t) \geq \sum_i q_i(t)r_i(t)x_i(t)$. If $q_{\max}(t) = \max_i q_i(t)$ and $k = \text{argmax}_i q_i(t)$ then $\sum_i q_i(t)r_i(t)m_i(t) \geq q_k(t)r_k(t) \geq q_{\max}(t)R^{\inf}$. We have

$$\begin{aligned} & L(t_0+w) - L(t_0) \\ \leq\ & c_1 + c_2 - 2\sum_t \sum_i (\varepsilon q_i(t)m_i(t)r_i(t)) \\ \leq\ & c_1 + c_2 - 2\varepsilon R^{\inf} \sum_t q_{\max}(t). \end{aligned}$$

Hence when the queues are sufficiently large, i.e. $\sum_{t=t_0}^{t_0+w} q_{\max}(t) > (c_1+c_2)/(2\varepsilon R^{\inf})$, the potential function decreases. This implies stability. We emphasize that this argument only works for $\varepsilon > 0$, $R^{\inf} > 0$ and $0 \notin \mathcal{R}$. □

5. OFFLINE ALGORITHM

We now consider the offline case. While this is an interesting theoretical problem in its own right, it could also be applicable to a situation in which we know the mobility and traffic patterns in advance and we wish to precompute a schedule.

We assume a set of users that each has a queue of data at time 0. Our objective is to service all the queues within a given set of time steps whenever possible. This is equivalent to finding a feasible solution to the following set of constraints. Let the integer variable $x_i(t) = \{0, 1\}$ indicate whether or not user i is serviced at time t. Let q_i be the queue size of user i at time 0. Ideally, we wish to find values for $x_i(t)$ that satisfy,

$$\sum_t r_i(t)x_i(t) \geq q_i \quad \forall i \ \text{ and } \ \sum_i x_i(t) = 1 \quad \forall t$$

It is easy to see that this offline problem is NP hard. For two users, suppose $r_0(t) = r_1(t)$ for all t and $q_0 = q_1 = \frac{1}{2}\sum_t r_0(t)$. Finding a feasible solution to service both queues completely is equivalent to solving the Partition problem, which is known to be NP-hard [10].

By linearizing the integer constraint $x_i(t) = \{0, 1\}$ to $0 \leq x_i(t) \leq 1$, we can find a feasible fractional solution in polynomial time. We propose two algorithms for rounding the fractional solution and prove that they satisfy the properties in Theorems 5.1 and 5.2. The construction of our rounding algorithms is motivated by the approach of Shmoys and Tardos [18] for minimizing makespan when scheduling on unrelated machines.

THEOREM 5.1. *If there is a feasible fractional solution, then we can guarantee that a* $1-\alpha$ *fraction of each queue is served as long as* $r_i(t) \leq \alpha q_i$ *for each time* t *and user* i*,*

PROOF. We construct a bipartite graph (U, T, E) with node sets U and T and an edge set E connecting nodes in U and nodes in T. Each time step t corresponds to one node in T. Each user i corresponds to c_i nodes in U, where $c_i = \lceil \sum_t x_i(t) \rceil$. Let these c_i nodes be called i^k where $1 \leq k \leq c_i$.

To describe the edge set, let us consider a particular user i. If in the fractional solution $x_i(t) = 0$ for time t then edges (t, i^k) do not exist for $k = 1 \ldots c_i$. Suppose $\{t^j : 1 \leq j \leq m\}$ is the set of time steps for which $x_i(t^j) > 0$. We now describe the subgraph induced by the node sets $\{t^j\}$ and $\{i^k\}$. We assign a value y to each edge in the edge set to ensure the following properties. For $(t^j, i^k) \in E$ we want,

$$\sum_{k=1}^{c_i} y(t^j, i^k) = x_i(t^j) \quad \text{for all } i, \tag{4}$$

$$\sum_{j} y(t^j, i^k) = 1 \quad \text{for } k < c_i, \tag{5}$$

$$\sum_{j} y(t^j, i^k) \leq 1 \quad \text{for } k = c_i. \tag{6}$$

For notational simplicity let us temporarily rename the time steps $t^1, \ldots, t^m$ by $1, \ldots, m$. We also assume without loss of generality that the rates are ordered, $r_i(1) \geq r_i(2) \geq \ldots \geq r_i(m)$. We find the minimum m_1 such that $\sum_{t=1}^{m_1} x_i(t) \geq 1$. We add edges (t, i^1) for $t = 1 \ldots m_1$ and assign the following y-values to ensure (4)-(6).

$$y(t, i^1) = \begin{cases} x_i(t) & \text{for } t < m_1, \\ 1 - \sum_{t=1}^{m_1-1} x_i(t) & \text{for } t = m_1. \end{cases}$$

We now move on to node i^2. If $y(m_1, i^1) < x_i(m_1)$ then part of $x_i(m_1)$ is currently unassigned. Hence we add an edge (m_1, i^2) and assign $y(m_1, i^2) = x_i(m_1) - y(m_1, i^1)$. In a similar manner we find the minimum m_2 such that $\sum_{t=m_1+1}^{m_2} x_i(t) + y(m_1, i^2) \geq 1$ and add edges (t, i^2) for $t = m_1 + 1, \ldots, m_2$. To ensure (4)-(6), we assign,

$$y(t, i^2) = \begin{cases} x_i(t) & \text{for } m_1 < t \leq m_2, \\ 1 - \sum_{t=m_1+1}^{m_2-1} x_i(t) - y(m_1, i^2) & \text{for } t = m_2. \end{cases}$$

We continue this process until we reach the last node i^{c_i}.

We now assume that in fact $\sum_j y(t^j, i^{c_i}) = 1$. (If this is not the case then we define a dummy time slot $\hat{t}_i$ and set $r_i(\hat{t}_i) = 0$, $y(\hat{t}_i, i^{c_i}) = 1 - \sum_j y(t^j, i^{c_i})$. We observe that y defines a *fractional matching* from nodes in T to U that matches every node i^k *exactly*. By standard matching theory this implies there is an *integral* matching that exactly matches every node i^k. We denote this integral matching by z. We have,

$$\sum_{0<t\leq m_1} r_i(t) z(t, i^1) + \sum_{m_1<t\leq m_2} r_i(t) z(t, i^2) + \ldots + \sum_{m_{c_i-1}<t\leq m_{c_i}} r_i(t) z(t, i^{c_i}) \tag{7}$$

$$\geq \sum_{m_1 < t \leq m_2} r_i(t) y(t, i^2) + \sum_{m_2 < t \leq m_3} r_i(t) y(t, i^3) + \ldots + \sum_{m_{c_i - 1} < t \leq m_{c_i}} r_i(t) z(t, i^{c_i}) \quad (8)$$

$$\geq \sum_{t=1}^{m} r_i(t) \sum_k y(t, i^k) - \sum_{t=1}^{m_1} r_i(t) y(t, i^1)$$

$$= \sum_{t=1}^{m} r_i(t) x_i(t) - \sum_{t=1}^{m_1} r_i(t) y(t, i^1)$$

$$= q_i - \sum_{t=1}^{m_1} r_i(t) y(t, i^1). \quad (9)$$

To see that the first inequality holds, recall $r_i(1) \geq r_i(2) \geq ....$ Every term in (7) and (8) is a convex combination of rates. The rates in (7) are larger than those in (8). If the rates satisfy the condition $r_i(t) \leq \alpha q_i$ for each time t and user i, the second term in (9) is upper bounded by αq_i. □

THEOREM 5.2. *If there is a feasible fractional solution, then for any rates we can guarantee that at least half of the users have at least half of their queue served.*

PROOF. We first note that integral feasibility is unchanged if we take each rate $r_i(t)$ to be $\min(r_i(t), q_i)$. Therefore, let us assume without loss of generality that $r_i(t) \leq q_i$ for all t and i. For each time slot t we create a pair of time slots t^A and t^B. We define new rates by,

$$r_i(t^A) = \begin{cases} q_i/2 & \text{if } r_i(t) \geq q_i/2 \\ 0 & \text{otherwise} \end{cases}$$

$$r_i(t^B) = r_i(t) - r_i(t^A).$$

We set up the constraints using the new time slots $\{t^A, t^B\}$ and the new channel rates. We then apply the above rounding algorithm to a feasible fractional solution. However, the rounded integral solution may not define a valid solution if t^A and t^B are assigned to serve two different users. We apply the following procedure to create a valid solution.

Initially all users are *unremoved*. Consider time slots t^A and t^B. Suppose t^A serves i and t^B serves j. If $r_i(t^A) = q_i/2$, then in our final solution t^A serves i and j is not served at all. Note that in this way, at least half of i's queue is served. Now both i and j are considered *removed*. We move on to the next pair of time slots and see which unremoved user(s) they serve. The procedure terminates when either all users are removed or all time slots are considered. Among all removed users exactly half of them have exactly half of their queue served. Among all unremoved users, no two can be served by time slots t^A and t^B for the same t (unless $r_i(t^A) = 0$). Hence, the rounded integral solution induced on the unremoved users is valid. □

We also present asymptotically matching integrality gaps to demonstrate that no rounding algorithm could be significantly better. Let n be the number of users.

THEOREM 5.3. *There exists an example with a feasible fractional solution such that in any integral solution at least one user has at most a* $1 - \alpha + \alpha/n$ *fraction of its queue served.*

PROOF. Suppose for simplicity that $1/\alpha$ is an integer. (The proof is similar if this is not the case.) We have $n(1+1/\alpha)$ time slots and $n+1$ users each of which has queue size 1. We define the following rates.

$$r_i(t) = \begin{cases} \alpha & \text{for } 0 \le t < n, i = 0 \\ \alpha & \text{for } 0 \le t < n(1/\alpha), i-1 = t \bmod n \\ \alpha/n & \text{for } n(1/\alpha \le t < n(1+1/\alpha), i-1 = t \bmod n \\ 0 & \text{otherwise} \end{cases}$$

Note that $r_i(t) \le \alpha$ for all i, t. A fractional solution that serves all users completely is given by,

$$x_i(t) = \begin{cases} 1/n & \text{for } 0 \le t < n, i = 0 \\ 1-1/n & \text{for } 0 \le t < n, i-1 = t \\ 1 & \text{for } n \le t < n(1+1/\alpha), i-1 = t \bmod n \\ 0 & \text{otherwise} \end{cases}$$

For the integral solution suppose that all users $1 \le i \le n$ receive service higher than $1-\alpha+\alpha/n$. Then all time slots $0 \le t < n$ must be assigned to user $i = t+1$. Therefore user 0 receives no service. □

THEOREM 5.4. *There exists an example with a feasible fractional solution such that in any integral solution at least $n/2-1$ users have less than half of their queues served.*

PROOF. We have $3n+1$ time steps and $2n$ users each of which has queue size 1. Let $\delta = 1/4n$. We define the following rates.

$$r_i(t) = \begin{cases} 2n\delta & \text{for } t = 0, 1 \le i \le 2n \\ 1 & \text{for } 0 < t \le n, i = t \\ 1 & \text{for } 0 < t \le n, i = t+n \\ 1/2-\delta & \text{for } n < t \le 2n, i = t \\ 1/2-\delta & \text{for } 2n < t \le 3n, i = t-2n \\ 0 & \text{otherwise} \end{cases}$$

The following is a fractional solution that serves all users completely. During every time step the server splits its service equally among all the users that have positive rates. The fractional solution x is given by,

$$x_i(t) = \begin{cases} 1/2n & \text{for } t = 0, 1 \le i \le 2n \\ 1/2 & \text{for } 0 < t \le n, i = t \\ 1/2 & \text{for } 0 < t \le n, i = t+n \\ 1 & \text{for } n < t \le 2n, i = t \\ 1 & \text{for } 2n < t \le 3n, i = t-2n \\ 0 & \text{otherwise} \end{cases}$$

For the integral solution, note that the rates for times $n+1, \ldots, 3n$ are less than $1/2$. Therefore, if a queue is more than half served then it must receive service during the first $n+1$ steps. In an integral solution, at most $n+1$ users (out of $2n$ users) can receive service in $n+1$ steps. □

A. THE QUALCOMM HIGH DATA RATE SYSTEM

In this section we describe the Qualcomm High Data Rate (HDR) system for high speed data transmission in wireless networks. This provides further motivation for our model and demonstrates its relation to wireless systems in reality.

HDR. In HDR the length of the time slot is 1.67ms, i.e. 600 slots per second. In each slot the basestation can transmit to at most one user. The user rate $r_i(t)$ is calculated as follows. Each mobile monitors the quality of a pilot signal transmitted by the basestation and estimates the channel quality. From this estimation mobile i decides on the rate, $r_i(t)$ at which it can receive data in the next time slot. The better the channel quality, the larger $r_i(t)$ is. The mobile then transmits this information to the basestation in a *Data Rate Control* message (DRC). From these messages the basestation has the complete rate vector. We note that many papers use the notation $DRC_i(t)$ instead of $r_i(t)$.

In order for decoding to be feasible, HDR uses a finite set of rates, namely (in kbits per second) $\mathcal{R} = \{0, 38.4, 76.8,\ 153.6, 307.2,\ 614.4, 921.6,\ 1228.8, 1843.2, 2457.6\}$. Our model simplifies two features of HDR. First, if the basestation decides to transmit at a low rate (e.g. 38.4kbps), HDR forces multiple time slots (e.g. 16 slots) to be assigned to the same user. Otherwise, the amount of data transmitted would be too small, which would lead to implementation problems. Second, if the basestation decides to serve user i, the actual data transmission rate might be slightly different from $r_i(t)$, due to the coding schemes that are used to compensate for noisy wireless channels.

As we mentioned earlier, the scheduler in our model assigns slots one at a time. In addition, if user i is served at time t the actual transmission rate is exactly $r_i(t)$. Our model is more general in that we allow a continuum of rates.

Proportional Fair. The standard scheduling algorithm used by HDR is known as Proportional Fair. This algorithm always chooses the user i that maximizes $r_i(t)/\mu_i(t+1)$ where $\mu_i(t+1)$ is an exponentially smoothed average of the service rate to user i up to time t. More precisely,

$$\mu_i(t+1) = (1 - 1/\tau)\mu_i(t) + s_i(t)/\tau,$$

where $s_i(t)$ is the amount of data scheduled to user i in time slot t and τ is a "time constant" for the averaging. (In the HDR system the value of τ is typically 1024 and so the averaging is taken over roughly 1.7 seconds.) The rationale for Proportional Fair is that if all users have large backlogs then the scheme maximizes $\sum_i \log \mu_i(t)$ [12; 19]. This implies that increasing $\mu_i(t)$ by some factor has the same effect on the objective as increasing $\mu_j(t)$ by the same factor.

There are in fact three versions of Proportional Fair depending on how users with small queues are scheduled. In [2] it is shown that all three versions can be unstable. This is even true against a benign adversary that only injects data at constant arrival rates and only uses 2 rate vectors that appear in a periodic fashion. More precisely, consider a 2-user system with the following arrival rates and service rates,

$$a_0(t) = 94 \quad \forall t, \quad a_1(t) = 49 \quad \forall t$$

$$r_0(t) = 100 \quad \forall t, \quad r_1(t) = \begin{cases} 1000 & \text{for } t \bmod 10 = 0 \\ 100 & \text{otherwise} \end{cases}$$

The system is stable if we assign time slot t to user 1 if $t \bmod 20 = 0$ and to user 0 otherwise. However, Proportional Fair is not "intelligent" to enough assign time slots in this way. It is shown in [2] that at most 9 out of every 10 time slots are assigned to user 0. Therefore the queue for user 0 is unstable.

Acknowledgements

The authors would like to thank Sasha Stolyar and Sem Borst for many helpful discussions about wireless scheduling. The authors would also like to thank two anonymous referees for their suggestions on improving the presentation of the paper.

REFERENCES

W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosen. Adaptive packet routing for bursty adversarial traffic. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 359 – 368, Dallas, TX, May 1998.

M. Andrews. Instability of the proportional fair scheduling algorithm for HDR. Submitted.

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.

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.

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.

E. Anshelevich, D. Kempe, and J. Kleinberg. Stability of load balancing algorithms in dynamic adversarial systems. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, Montreal, Canada, May 2002.

B. Awerbuch, P. Berenbrink, A. Brinkmann, and C. Scheideler. Simple routing strategies for adversarial systems. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, pages 158 – 167, Las Vegas, NV, October 2001.

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.

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

M. R. Garey and D. S. Johnson. *Computers and intractability - A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

A. Jalali, R. Padovani, and R. Pankaj. Data throughput 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.

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

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.

J. Rhee, T. Kim, and D. Kim. Wireless fair scheduling algorithm for 1xEV-DO system. In *Proceedings of the IEEE Semiannual Vehicular Technology Conference, VTC2001-Fall*, Atlantic City, NJ, October 2001.

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.

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.

S. Shakkottai and A. Stolyar. Scheduling for multiple flows sharing a time-varying channel: The exponential rule. *Analytic Methods in Applied Probability*, 2002. To appear.

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.

A. Stolyar. Multiuser throughput dynamics under proportional fair and other gradient based scheduling algorithms. Submitted.

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.

D. Tse. Forward-link multiuser diversity through rate adaptation and scheduling. In preparation.

Received ; revised ; accepted