Parallel Scheduling Problems in Next Generation Wireless Networks

L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and A. Vitaletti
Università di Roma, "La Sapienza," Rome, Italy

S. Diggavi
Laboratoire des systèmes d'information et de communication, Lausanne, Switzerland

S. Muthukrishnan
Rutgers University, New Jersey

T. Nandagopal
Bell Labs, Holmdel, New Jersey

Next-generation 3G/4G wireless data networks allow multiple codes (or channels) to be allocated to a single user, where each code can support multiple data rates. Providing fine-grained QoS to users in such networks poses the two-dimensional challenge of assigning both power (rate) and codes to every user. This gives rise to a new class of parallel scheduling problems. We abstract general downlink scheduling problems suitable for proposed next-generation wireless data systems. Our contribution includes a communication-theoretic model for multirate wireless channels. In addition, while conventional focus has been on throughput maximization, we attempt to optimize the maximum response time of jobs, which is more suitable for streams of user requests. We present provable results on the algorithmic complexity of these scheduling problems. In particular, we are able to provide very simple, on-line algorithms for approximating the optimal maximum response time. We also perform an experimental study with realistic data of channel conditions and user requests that strengthens our theoretical results. © 2004 Wiley Periodicals, Inc. NETWORKS, Vol. 45(1), 9–22 2005

1. INTRODUCTION

There is tremendous momentum in the wireless industry towards next-generation (3G and beyond)[1] systems. These systems will not only migrate the existing voice traffic to a higher bandwidth platform, but are also expected to jump-start large-scale data traffic. These emerging wireless systems[2] such as CDMA, wideband OFDM, and multislot TDMA allow multiple codes (parallel channels) to be allocated to users, in each of which traffic can flow in one of multiple rates. This provides them more flexibility than is available in current systems to manage and modulate the traffic. This also gives rise to novel parallel scheduling problems that we study in this article. We use the terms code and channel interchangeably, because the terminology varies between CDMA, OFDM, and TDMA systems. So, for instance, a code (channel) will denote a code when referring to CDMA, a frequency tone when referring to OFDM, and a time slot for TDMA.

In a packet wireless cellular architecture each cell has a base station and is connected by a high-speed backbone to the Internet. Each base station handles all requests to and from mobile users within the cell, that is, it handles both uplink (from mobile users) and downlink (to mobile users) requests.

Our focus in this article is on the downlink channel

---

[1] G: Generation. Current systems are 2G or 2.5G, where rudimentary data services are being deployed.

[2] CDMA: Code Division Multiple Access; OFDM: Orthogonal Frequency Division Multiplexing; TDMA: Time-Division Multiple Access.

performance, which is likely to be a major focus in emerging systems because data traffic is expected to dominate over time and data traffic typically tends to have asymmetrically large downlink demand. Existing wireless scheduling and resource allocation algorithms cannot be directly applied to manage the downlink because wireless networks have unique characteristics, such as location-dependent channel errors. Users in different regions of a cell experience different channel conditions, resulting in location-dependent data and packet error rates. Unlike traditional scheduling scenarios, in a wireless environment, the scheduler must consider channel state to provide reasonable Quality of Service (QoS), and wireless systems have a variety of built-in capabilities to gather channel condition information to this purpose.

1.1. Contributions

Our contributions are threefold.

1. We abstract a general downlink scheduling problem in next generation wireless data networks. This problem has many novelties. For example, we embody channel characteristics guided by communication theoretic considerations, and the properties of these channels get exploited in our scheduling algorithms. Second, we study QoS parameters related to per-request behavior; in particular, we focus on optimizing response time per request. In contrast, prior work in wireless systems scheduling has typically focused on rate optimization (maximization) metrics (see, for instance, [17, 21, 22]).

2. We show that the problems we propose are NP-complete, and that they are also hard to approximate. However, we use resource-augmented competitive analysis, to derive simple, on-line algorithms that provably have good performance in approximating the optimal maximum response time of a job. We also present results for other QoS metrics, such as average and weighted response times.

3. We present a detailed experimental study of our algorithms. Using real Web server request logs and realistic 3G/4G system parameters, we show experimentally that our on-line algorithms are practical, and perform significantly better than our worst-case analyses indicate. We also examine which resource is most important to augment in a wireless setting.

1.2. Related Work

There has been a significant amount of work on scheduling problems over wireless channels. We study the downlink scheduling problem. The uplink scheduling problem is a complementary problem where the fundamental issues are quite different. See [5] and references therein for more details.

Typically, resource allocation problems study per-user rate throughput (see e.g. [4]). The rate optimization problem has been extensively studied for various wireless systems with focus varying from maximizing overall throughput to providing a minimum throughput guarantee for all users. A good discussion of related work about throughput optimization and fairness in wireless data networks can found in [17, 21].

Job scheduling is very popular in the context of processor scheduling, and various algorithms have been proposed for different QoS metrics such as completion time, maximum response time, and weighted response time [15]. In the parallel scheduling literature, there has been significant work on scheduling of *malleable* tasks; see, for example, [12, 16, 18, 25, 26]. Our multidimensional malleable scheduling problem has not been studied previously.

In wireless networks, job scheduling has been addressed in the context of downlink *broadcast* scheduling [1]. There a single transmission may satisfy multiple users or requests, which is a model that is applicable in many special purpose systems. In contrast, we focus on unicast scheduling where each transmission is destined to a unique request that applies to general purpose wireless data networking. In a recent work, downlink unicast scheduling in CDMA systems was studied [13]; this is close to our work in spirit. However, they assume a linear rate model for the physical layer, which is not accurate. Also, they do not have any upper bound on the number of available codes; hence, they study the problem of allocating power only. We have studied the nuances of allocating both power and codes, which is more suitable. Finally, we have provided a thorough competitive analysis of the on-line algorithms, in particular, using the resource augmented analysis; this is the first provable result known for any of the on-line scheduling problems, including the ones in [13].

1.2.1. Roadmap. We present the communication channel model, and abstract our scheduling problem in Section 2. In Section 3, we present a theoretical study showing the structure and the complexity of these problems. In Section 4, we present our main algorithmic results, namely, simple on-line algorithms and present our augmented-resource based analyses. In Section 5, we present our experimental results. We have obtained a number of algorithmic results for optimizing other per-request QoS metrics such as (weighted) average response times, which we have included in Section 4.2.

2. PROBLEM FORMULATION

In this section, we describe the effects of the wireless communication medium on the transmission rates for each user, and we also formulate parallel scheduling problems that arise in next-generation wireless networks.

2.1. Communication Channel Model

In wireless systems, channels have variable attenuation depending on the geographic location of the users. This is mainly due to multipath impairments and radio propagation losses. Say the base station (BS) is communicating with n mobile users. The physical channel (code) attenuations of

the users are denoted by $\bar{g}_1, \bar{g}_2, \ldots, \bar{g}_n$, respectively; each $\bar{g}_i$ is a scalar parameter called the *physical* gain. If the BS transmits power p_i to a user i, the signal-to-interference-plus-noise ratio (SINR) is given by SINR $= \bar{g}_i p_i / \sigma^2$, where σ^2 is the total noise power (including interference) [23]. SINR determines the rate of transmission of packets to the user. In particular, the rate $r_{bps}(\cdot)$ as a function of the SINR can be suitably approximated by a concave logarithmic function [9].

$$r_{bps}(\text{SINR}) = \bar{W} \log_2\left(1 + \frac{\text{SINR}}{\Gamma}\right) \qquad (1)$$

where $r_{bps}(\text{SINR})$ is the rate in bits per second, $\bar{W}$ is the spectral bandwidth used, and Γ is dependent on the coding gain from the physical layer error-correcting code [9]. Both Γ and $\bar{W}$ are system parameters, which for our purposes will be constants. Therefore, for a particular user i that is allocated any single channel, the number of bits received over a period of time τ and over the assigned channel, obtained as a function of the power p allocated to the user on that channel (code), is given by

$$r_i(p) = \bar{W}\tau \log_2\left(1 + \frac{\bar{g}_i p}{\sigma^2 \Gamma}\right) \qquad (2)$$

The rate versus SINR curves for next-generation wireless systems closely approximate the convex function described by this equation (see, e.g., [7]). This rate function already embodies the effect of variable rate error-correcting coding schemes in the physical layer, as is typical in next-generation wireless systems [7, 9, 20]. Therefore, we will use this equation for rate calculations in our scheduling problems. For notational convenience we will denote $g_i = \bar{g}_i / \Gamma \sigma^2$ as the *channel gain,* and we will set $W = \tau \bar{W}$ yielding $r_i(p) = W \log_2(1 + pg_i)$. Observe that SINR is implicitly contained in the term pg_i.

2.2. Abstract Scheduling Problem

The base station has a total power P to transmit. Time is assumed to be partitioned into equal width windows called *time slots*[3] (or frames), whose width is τ. We also assume that there are a total of C codes that can be assigned to users in each time slot. Requests[4] arrive in the system over time at the beginning of time slots; requests/jobs are for nonreal-time traffic such as browsing, downloads, etc., that is, file transfers. In the sequel, we sometimes speak of a request's gain, to mean the gain of the user that placed the request. The size s_i (in bits) and the channel gain g_i of the user who made the ith request are known when the request arrives at time a_i. The arrival time is also known as *release time.* We

will assume that the channel conditions of the users are constant over the scheduling period. Although this is a simplification, it holds in realistic cases.[5] The scheduling problem is to determine an assignment of power and codes to each user in each time slot, so as to optimize the required QoS criterion. Observe that, if power p is allocated to the ith user on any code j over some time slow, the user receives data at a rate $r_i(p)$ in the time slot over code j. If a user is assigned more codes over a same time slot, the corresponding rates sum up. So, for instance, if user i is assigned powers p_1 and p_2 over two different codes in some time slot, it receives data at a rate $r_i(p_1) + r_i(p_2)$ in the time slot considered. Observe that the above implies that gain only depends on the user and is the same for all codes assigned to the user. An obvious constraint is that the same code cannot be used by two different users in a same time slot.

Formally, jobs arrive on-line and each is fully described by its size and arrival time, while the scheduling algorithm produces the following output.

2.2.1. Output. For every user/job i, $\mathcal{C}_i(t)$, the set of codes assigned to i at time t, and $p_i^j(t)$, the power assigned to i at time t in each code $j \in \mathcal{C}_i(t)$. In the sequel, we sometimes omit t when clear from context.

The assignment must satisfy the following conditions.

1. *Total Code and Power Constraints.* Obviously, for any time slot t, $\sum_i |\mathcal{C}_i(t)| \leq C$ and $\sum_i \sum_{j \in \mathcal{C}_i(t)} p_i^j(t) \leq P$.
2. *Discrete Rate Set.* Only a discrete set of rates (equivalently, minimum power per discrete rate) is allowed. These rates denoted $R(1), R(2), \ldots$, have the property[6] that $R(h)/(R(h-1)) \leq 2$.
3. *Request Completion.* All requests get the requested data size, that is, if s_i and $R_i(t)$ denote the size of request i and the overall rate provided to it in time slot t, then we need

$$s_i = \sum_t R_i(t) = \sum_t \sum_{j \in \mathcal{C}_i(t)} r_i(p_i^j(t)) \qquad (3)$$

where $r_i(p_i^j(t))$ is calculated using (2),[7] subject to the discrete rate set constraint above.

Finally, as remarked above, no two users can be allocated the same code in the same time slot.

Our goal is to *minimize the maximum response time* or *max-flow* [6], where response time for request i is $c_i - a_i$,

---

[3] We will use time and time slot interchangeably when no confusion arises.
[4] The terms: requests, jobs, and users, will be used interchangeably.

[5] If time scale, of the scheduler is several seconds, and if the users do not have very high mobility, then the channel conditions will be static over this time scale [23].
[6] This relationship holds for existing next-generation wireless data system proposals like cdma2000 and HDR, which have rate set of {38.4, 76.8, 102.6, 153.6, 204.8, 307.2, 614.4, 921.6, 1228.8, 1843.2, 2457.6} kilobits per second (kbps). The factor 2 is not sacrosanct. If the discrete rates are more spread out, but bounded by some constant, all our results will apply with minor changes in the claimed bounds.
[7] For a given user the gain is constant across channels.

if request i is completed at time c_i.[8] Sometimes, we may write $c_i^A(\mathcal{I})$, to mean the completion time of the ith request of instance $\mathcal{I}$, when submitted to algorithm A. We will focus on this goal for most of the article. We also prove results for certain related metrics such as total weighted response time. These results appear in Section 4.2.

We assume requests may be served over several time slots with different sets of codes at each time slot. In standard scheduling terminology [6], this corresponds to requests being *preempted* (i.e., stop processing a request, process other requests, and resume the original request) and *migrated* (i.e., sets of codes assigned to a user may differ from one time slot to another).[9]

There are two basic variants of our problems, namely *off-line* or *on-line*. In the off-line version, all request arrivals are known ahead of time. The off-line case is of theoretical interest and is mainly useful to quantify the benefit to be accrued from scheduling. In the on-line case, requests arrive over time and the scheduling algorithms have to take their decisions without knowledge of future requests. The performance of the on-line algorithms is measured in comparison to the off-line case as in standard competitive analysis.

A *malleable resource* problem is one in which giving more of a resource (say, more processors) improves performance. Therefore, our problem is a version of a multidimensional malleable resource problem. Prior work involves malleable scheduling of parallelizable jobs [12, 26] or scheduling a mixture of malleable and nonmalleable jobs [8, 18], but none addresses our problem. For example, in our case, the processing time of a job depends on two variables, the number of codes, and the assigned power per slot; both these resources are malleable, and they have a strongly (nonlinearly) coupled effect on data rate through Equations (2)–(3). Our work is different from [24], because our focus is on maximum response time, with preemption and on-line job arrivals.

3. UNDERSTANDING THE SCHEDULING PROBLEMS

3.1. Some Structural Observations

Here, we state and prove properties of the communication channel. They will be invoked later in proving our main results.

3.1.1. Continuous Power (Rate) Cases.
First, we consider the case where rates are not discrete, and they depend on power according to a monotonic function, as in Equation (2). Concavity of the rate with respect to p in (2) implies that if we assign $c \geq 1$ codes to a user u, then *it is optimal*

to divide the total power p allocated to that user equally among the codes assigned as summarized below.

Lemma 3.1 (Equipartition of power). *Given c codes and overall power p to a user i in time slot t, the overall rate $R_i(t)$ in time slot t is maximized for $p_i^j = p/c$, $j = 1, \ldots, c$.*

Using Lemma 3.1, we can write the rate obtainable by the ith user when it is given c codes and total power p as

$$R_i(t) = R_i(p, c) = Wc \log\left(1 + \frac{g_i p}{c}\right). \tag{4}$$

3.1.2. Discrete Power (Rate) Case.
When we have a discrete rate set, the actual rate obtained on each code is given by the highest discrete rate,[10] which is below $W \log(1 + (gp/c))$. Therefore, the difference between the continuous rate and the discrete rate case depends on the discrete rate set available. The proof of the following fact is obvious.

Fact 3.1. *If $R(h)/(R(h-1)) \leq 2$ for the discrete rate set $\{R(h)\}$, then for any continuous power allocation p there exists a discrete rate $R(l)$ such that $R(l) \geq \frac{1}{2} r_i(p)$, where $r_i(p)$ is given by* (2).

3.2. Computational Hardness

To understand the challenge of the problem further, let us consider the off-line complexity of the scheduling problems. If power (rate) values are required to be drawn from a discrete set, the problem in its simplest instance is the bin packing problem, and hence, it is NP-complete [11]. We focus on the more challenging case when the number of codes is discrete, as usual, but the power (and, hence, the rate) is allowed to take any continuous value. To prove the hardness of this problem, we will consider the version of the problem in which the ith request has arrival (release) time a_i, deadline d_i and size s_i in bytes. Using this, we can state the following theorem.

Theorem 3.1. *If the number of codes assigned to each user is integral, then it is NP-complete to compute a feasible schedule for the problem of meeting deadlines even if all users have the same channel gain, a common release time, a common deadline and the power assigned to each code is not restricted to a discrete set of values.*

Proof. We reduce the NP-complete problem *3-partition* defined as follows [11]: *INSTANCE:* A set A of $3m$

---

[8] This is also sometimes called flow time in literature.

[9] A more detailed model may distinguish some codes to be more preferable than the others from one time slot to another to ensure intercell interference avoidance, an issue we do not consider in this article.

[10] Note that we make a regularity assumption that the user gains are such that the lowest discrete rate $R(1) \leq W \log(1 + gP)$, that is, by allocating all the resources to the user there exists a feasible discrete rate. In practice, error-correcting codes can be used over a group of codes to increase the dynamic range of user gains that fall into the feasible region.

elements, $a_1, a_2, \ldots, a_{3m}$, a bound B and a size $s(a_j)$, $a_j \in A$, such that $\sum_{j=1}^{3m} s(a_j) = mB$.

QUESTION: Can A be partitioned into m disjoint sets $A_1, A_2, \ldots, A_m$ such that for $1 \leq i \leq m$, A_i has three elements and $\sum_{a_j \in A_i} s(a_j) = B$?

The reduction is as follows: given an instance $\mathscr{I}$ of 3-partition we define an instance $\mathscr{J}$ of the combinatorial problem of meeting deadlines with $3m$ users. The request of user j, $1 \leq j \leq 3m$, has size $s_j = s(a_j)$; all requests are released at time 0 and have a common deadline $D = m$. All users have the same channel gain g and, therefore, the power that users need to get desired rates depends only on the number of codes they are assigned and on the size of the request. Let p_j denote the power assigned to user j if only one code is assigned to j over all frames. We assume that there are three codes per frame and that the maximum power of the base station is P where $\sum_{j=1}^{3m} p_j = mP$.

We now show that there is an assignment of users to frames that meets the common deadline D if and only if $\mathscr{I}$ has a feasible solution.

Assume that $A_1, A_2, \ldots, A_m$ is a feasible solution of $\mathscr{I}$. We define a solution of $\mathscr{J}$ as follows: if element a_j, $j = 1, 2, \ldots, 3m$, is assigned to set A_i then user j is assigned to frame i with the minimum power that is needed to satisfy the user's request in one code in any one frame. It is easy to see that this is a feasible solution because $\sum_{a_j \in A_i} s(a_j) = B$ implies that P is the total power required by users assigned to frame i.

Similarly, it is possible to show that given a feasible solution of $\mathscr{J}$ that satisfy all users' request within m frames, it is possible to obtain a feasible solution of $\mathscr{I}$. Namely, it is sufficient to assign to set A_i, $i = 1, 2, \ldots, m$, the elements that correspond to users assigned to frame i; because the total power assigned to frame i is P, it follows that $\sum_{a_j \in A_i} s(a_j) = B$. That completes the proof. ∎

The result above, in fact, shows the problem to be NP-complete in the strong sense (see [11] for definition and significance). It is easy to see that this result immediately extends to the problem of minimizing the maximum response time.

We can also show that this problem is hard to approximate. This is summarized in the following theorem.

Theorem 3.2. *For every $c > 0$, it is NP-hard to compute a c-approximation of the maximum response time.*

Proof. We apply the gap technique for proving inapproximability results ([2]) showing a reduction from Three-Dimensional Matching to the problem of minimizing the maximum response time in the discrete case. Three-Dimensional Matching is defined as follows:

INSTANCE: a set A of $3n$ rationals x_i, $0 < x_i < 1$, $i = 1, 2, \ldots, 3n$ such that $\sum_{i=1}^{3n} x_i = n$.

QUESTION: Can A be partitioned into n sets F_j, $j = 1, 2, \ldots, n$, such that for each j, $j = 1, 2, \ldots, n$, $|F_j| = 3$ and $\sum_{\forall x_i \in F_j} x_i = 1$?

Given c and an instance $\mathscr{I} = \{x_1, x_2, \ldots, x_{3n}\}$ of Three-Dimensional Matching we define an instance $\mathscr{I}'$ of the scheduling problem of minimizing the max response time, and we show that if $\mathscr{I}$ has a solution, then the max response time of $\mathscr{I}'$ is n; otherwise, the max response time of $\mathscr{I}'$ is bigger than cn. Therefore, by the gap technique, if there exists a polynomial time algorithm for solving the problem of minimizing the maximum response time within a factor c approximation of the optimum, then it is possible to solve Three-Dimensional Matching in polynomial time. Because Three-Dimensional Matching is NP-complete the theorem follows.

Given c and $\mathscr{I} = \{x_1, x_2, \ldots, x_{3n}\}$, we define an instance of the scheduling problem with $(cn + 1)n(c + 3)$ jobs, and we assume that each slot has total power equal to 10, and that there are three codes per slot.

The jobs are released in $cn + 1$ phases; in each phase $n(c + 3)$ jobs are released. Phase $k = 0, \ldots, cn$ is formed by two stages as follows:

STAGE 1. At time $t = kn(c + 1)$, $3n$ requests $J_1, J_2, \ldots, J_{3n}$ arrive. For each k and i, the gain function of request J_i is such that by assigning power greater or equal to $3 + x_i$ one code is sufficient to complete the job in a slot, and $3 + x_i$ is the minimum power requirement to process request J_i. Jobs released in Stage 1 of a phase are called **J**-jobs in the following.

STAGE 2. At time $t = kn(c + 1) + n + j$, $j = 0, 1, \ldots, nc - 1$ a job A_j is released. For each k and j, the gain function of this job is such that by assigning power equal or greater to 8, one code is sufficient to complete it in a slot, and 8 is the minimum power requirement to process the job. Jobs released in Stage 2 of a phase are called A-jobs in the following.

The easy proof of the following Fact is omitted.

Fact 3.2. *All requests released in stage 1 of a phase can be completed using n slots if and only if $\mathscr{I}$ has a solution; otherwise, there are at least $n + 1$ slots containing requests released in stage 1 of the phase.*

We first prove that if instance $\mathscr{I}$ of Three-Dimensional Matching has a solution then the maximum response time is equal to n. This follows, because the $3n$ jobs released at time $kn(c + 1)$ can be scheduled in n slots, and therefore completed by time $kn(c + 1) + n$. Every job of stage 2, released at time $t = kn(c + 1) + n + j$, $j = 0, 1, \ldots, nc - 1$, is scheduled at time t itself.

We are left to prove that if instance $\mathscr{I}$ of the three-dimensional matching problem has not a solution then the maximum response time is bigger than cn. This follows from the following induction on the number k of phases:

Fact 3.3. *Assume that instance $\mathscr{I}$ has not a solution. For every $k = 1, 2, \ldots, cn$, if no J-job released in a phase $j < k$*

TABLE 1. Off-line scheduling programs.

Time indexed program (integral)	Interval indexed program (fractional)
Maximize 1	Maximize 1
Subject to	Subject to
$\sum_{i=1}^{n} c(i, t) \leq C, \; \forall t \quad \sum_{i=1}^{n} p(i, t) \leq P, \; \forall t$	$\sum_{i=1}^{n} c(i, k) \leq C, \; \forall k \quad \sum_{i=1}^{n} p(i, k) \leq P, \; \forall k$
$\sum_{t=a_i}^{d_i-1} Wc(i, t)\log\left(1 + \dfrac{p(i, t)g_i}{c(i, t)}\right) \geq s_i, \; \forall i$	$\sum_{t=a_i}^{d_i-1} W\tau_k c(i, k)\log\left(1 + \dfrac{p(i, k)g_i}{c(i, k)}\right) \geq s_i, \; \forall i$
$\sum_{t<a_i, t\geq d_i} c(i, t) = 0, \; \forall i$	$\sum_{k<a_i^{-1}, k>d_i^{-1}} c(i, k) = 0, \; \forall i$
$\sum_{t<a_i, t\geq d_i} p(i, t) = 0, \; \forall i$	$\sum_{k<a_i^{-1}, k>d_i^{-1}} p(i, k) = 0, \; \forall i$
$c(i, t), p(i, t)$ discrete values	$c(i, k), p(i, k) \geq 0, \; \forall i, k$

has response time bigger that cn then $k - 1$ A-jobs released in phases previous to k are not scheduled by time $kn(c + 1)$.

Proof. We prove the claim by induction. For the basis of the induction we prove the claim for $k = 1$. If instance $\mathscr{I}$ has not a solution then, by Fact 3.2, at least one J-job is not completed by slot $n - 1$. If the response time of this job is less than $n(c + 1)$, then it is scheduled in some time slot in $[n, n(c + 1) - 1]$, and therefore, one A-job will be scheduled after time $n(c + 1)$. Assume the claim is true until phase $k - 1$. We therefore have $k - 1$ A-jobs released before time $(k \; 1)$ not completed by time slot $kn(c + 1)$. All J-jobs released in stage 1 of phase k are scheduled in at least $n + 1$ different slots before time $(k + 1)n(c + 1)$. Because a time slot cannot accommodate an A-job together with a J-job, it then follows that at least k of the A-jobs released by time $(k + 1)n(c + 1)$ will be scheduled no earlier than $(k + 1)n(c + 1)$. ∎

We therefore have $cn + 1$ A-jobs released by time $(cn + 1)n(c + 1)$ scheduled after this time. The max response time of one A-job is therefore at least $cn + 1$.. ∎

3.3. Off-Line Scheduling Problem

We study the off-line version, that is, when all arrival times are known *a priori*. Using this, we will get lower bounds on the optimal values of certain QoS metrics, which will be a benchmark to compare against on-line algorithms.

3.3.1. Deadlines Scheduling Problem. Here, each request i has an arrival time a_i as well as a deadline d_i. As before, for each request i, at time a_i we know its size s_i and the channel gain g_i. The goal is to merely test feasibility, that is, determine if there is a valid schedule that meets all deadlines. This problem is the technical core of many other scheduling problems.

We can write the solution to the deadlines scheduling problem as a combinatorial optimization program as shown

in Table 1. Here, $c(i, t)$ denotes the number of codes assigned to user i in time slot t, while $p(i, t)$ is the total power assigned to user i in time slot t over all the codes. This is called the *time indexed program* in the table.

Because $c(\;)$ and $p(\;)$ take on only discrete values, even to check feasibility is an NP-complete problem as proved earlier. Hence, we relax the variables to be continuous by allowing $c(i, t)$ and $p(i, t)$ to be fractional, resulting in the *Fractional Time Indexed Program.*

Theorem 3.3. *There exists a pseudopolynomial time algorithm to solve the Fractional Time Indexed Program.*

Proof. We first show that the constraint set is convex. The Hessian **H** for $R_i(p, c) = Wc \log(1 + (gp/c))$ is given by

$$\mathbf{H} = -\frac{Wg^2}{(c + gp)^2 c}\begin{bmatrix} c \\ -p \end{bmatrix}[c \quad -p]. \tag{5}$$

H is negative semidefinite, and therefore, the function $R_i(p, c)$ is concave in (p, c), (albeit not *strictly* concave) [19]. The other constraints are linear, and hence, the program is convex. There exist polynomial time solutions for convex programming problems that test feasibility [19]. For this program, the running time is polynomial not in input size (n, $\log s_i$, etc.), but rather is polynomial in the number of variables, which in turn, is bounded by the size of the numbers in the input (i.e., s_i). ∎

3.3.2. Interval Indexed Program. The relaxation of the integer programming problem to the pseudopolynomial time algorithm, still results in a problem size of $O(nT)$ variables, where n is the number of requests and T is the total length of the schedule. Next we consider decreasing the number of variables used in the convex program. We will define a new program below called the *interval indexed program.*

An *event* is either the arrival or the deadline of a request in the system. Consider the sorted list of the events $t_1, \ldots, t_K$. We divide the time scale into *intervals* where an interval is the time period between any two consecutive events, that is interval I_k contains $[t_k, t_{k+1})$. For n requests, the total number of intervals is at most $2n$. We will look for *sliver* solutions, that is, ones in which for each interval I, each user i gets power $p(i, t)$ and $c(i, t)$ for $t \in I$ that remains constant for all $t \in I$, that is, $p(i, t_1) = p(i, t_2)$ for $t_1, t_2 \in I$, and likewise, for $c(\ )$. Let $c(i, k)$ be the fractional number of codes and $p(i, k)$ be the fractional power assigned to job i in interval k *per time slot*. Let a_i^{-1} denote the interval at the beginning of which job i arrives in the system, and d_i^{-1} be the interval at the end of which its deadline lies.

The formulation of the scheduling problem with slivers as a fractional program is given in Table 1.

Theorem 3.4. *The time indexed program has a feasible solution if and only if the interval indexed program has a feasible solution. It can be solved in time polynomial in n, C using convex programming.*

Proof. We will show that if the time indexed convex program has a feasible solution, so does the interval indexed convex program; the other direction is trivial.

Say $\overline{p(i, t)}$ and $\overline{c(i, t)}$ be the power and code assignments, respectively, at time frame t to user i in the time indexed convex program. Therefore, these values satisfy all the constraints of the time indexed convex program. We now claim that $p(i, k) = (\sum_{t \in k} \overline{p(i, t)})/\tau_k$ and $c(i, k) = (\sum_{t \in k} \overline{c(i, t)})/\tau_k$ are feasible values in the interval indexed convex program for user i in interval k, for all users and intervals. That is, these values would satisfy the constraints of the interval indexed convex program. Clearly, we can bound $\sum_i p(i, k)$ as

$$\sum_i \frac{\sum_{t \in k} \overline{p(i, t)}}{\tau_k} = \sum_{t \in k} \frac{\sum_i \overline{p(i, t)}}{\tau_k} \le \sum_{t \in k} \frac{P}{\tau_k} \le P.$$

So the power constraint is satisfied; similarly, the code constraint is satisfied also. We have

$$\frac{1}{\tau_k} \sum_{t \in k} \overline{c(i, t)} \log\left(1 + \frac{g_i \overline{p(i, t)}}{\overline{c(i, t)}}\right) \le c(i, k) \log\left(1 + \frac{g_i p(i, k)}{c(i, k)}\right)$$

due to Jensen's inequality for multidimensional functions, which states that $\mathbb{E}[f(\mathbf{X})] \le f(\mathbb{E}[\mathbf{X}])$ for a concave function $f(\cdot)$. Therefore, the demand constraint is satisfied, which completes the proof. ∎

The result above exposes an interesting structural property of the interval indexed convex program, that is, the structure that sliver assignment of power and code to requests is optimal in the fractional case.

3.3.3. Using the Deadlines Scheduling Problem. Given that the feasibility of the (relaxed) time-indexed program can be solved efficiently, we can use it to solve the off-line maximum response time problem nearly optimally. We do this by guessing a target response time F, and checking the feasibility of a deadline scheduling problem with deadlines $a_i + F$. By doing a binary search on the target response time value, we get an efficient (polynomial time) algorithm to optimize the maximum response time. Indeed, the same approach works for optimizing quality of service criteria such as $\max_i f(c_i - a_i)$ for any monotonic increasing function f.

4. ON-LINE HEURISTICS

In this section we present our main algorithmic results, namely, a set of on-line algorithms for optimizing the metrics of our interest. As is standard, we measure the performance of an on-line algorithm using the ratio of the value of the objective function (here, max-*flow*) achieved by the on-line algorithm and the optimal value, which can be computed off-line.

In our analysis we also use *resource augmentation* [14]. That is, we compare the optimum (i.e., the solution found by adversary) with the value of the solution found by the on-line algorithm when it is provided with more resources than an adversary who can serve it optimally.

Formally, we say that algorithm A for our scheduling problem is an $(\alpha, \beta, \gamma, \delta)$ approximation if it provides an α approximation of the optimum when the sizes of user requests are scaled down by a factor β and the number of codes (the power) used by the algorithm is at most γ (δ, respectively) times the number of codes (the power, respectively) used by the optimum solution to serve the original input sequence.

4.1. Minimizing the Maximum Response Time

We first develop our analysis for the "continuous" rate case (Theorem 4.2) and then analyze the case when, for each code, only a discrete rate set is allowed (Theorem 4.4). It is well known in processor scheduling literature [6] that the on-line algorithm Earliest Release Time (ERT) or First-In First-Out (FIFO) is optimal for minimizing the maximum response time on a single machine and is a 3-approximation algorithm on parallel machines. In our case, FIFO allocates all the codes and power to one user at a time until the user completes the job. Using the "Equipartition of power" lemma (Lemma 3.1), this translates into giving the user a power per code of P/C in consecutive time slots that the user completely occupies. In the sequel, for the sake of brevity, we speak of a *P/C allocation* to mean such an allocation. All heuristics we propose in the following use a *P/C allocation*.

We study the on-line scheme where each user is given a power P/C per code and is served by an FIFO scheduling discipline. We call this scheduling discipline FIFO(P/C). In

the sequel, given an instance $\mathscr{I}$ of continuous job arrivals and any algorithm A, we denote the maximum response time achieved by A on instance $\mathscr{I}$ by $f^A(\mathscr{I})$. In the case of FIFO(P/C) we simply write $f^{\text{FIFO}}(\mathscr{I})$. We also denote by $p_i^{\text{OPT}}(\mathscr{I})$ and $k_i^{\text{OPT}}(\mathscr{I})$ the overall power and codes assigned to the ith job of instance $\mathscr{I}$ by the optimum. We also denote by $k_i(\mathscr{I})$ the overall number of codes assigned to job i, when it is served by allocating equal power P/C to each of the codes assigned to i. We first show a negative result, which demonstrates that FIFO(P/C) could be arbitrarily worse than the optimum.

Theorem 4.1. *For any $M > 0$ there is an instance $\mathscr{I}$ such that*

$$f^{\text{FIFO}}(\mathscr{I})/f^{\text{OPT}}(\mathscr{I}) > M.$$

Proof. Assume $C - 1$ jobs arrive in a batch every time slot so that they need $P + \delta/C$ power each and one code to complete and such that $0 < \delta \leq (P/(C - 1))$, which implies that the jobs can be scheduled in one time slot.[11] Such a sequence would be scheduled in two time slots by FIFO(P/C), because it assigns two codes for each job, and therefore, would need $2C - 2$ codes for every $C - 1$ jobs. Hence, the job batch that arrives at the Mth time slot is scheduled in the same time slot by the optimum, and therefore has a response time of 1, whereas the on-line FIFO(P/C) serves this job set only after $2(M - 1)$ time slots have elapsed. Hence, this job set has a response time $2 + 2(M - 1) - M = M$, for any M. ∎

Despite the negative results above, we can show that FIFO(P/C) is able to achieve the optimum if every request is reduced to 50% of its original size. In the sequel, given an instance $\mathscr{I}$ of the problem, we denote by $\mathscr{I}'$ the instance in which each job size s_i in $\mathscr{I}$ has been reduced to $s_i/2$. So, the ith request becomes $(a_i', s_i') = (a_i, s_i/2)$. To prove our result we need the following lemma.

Lemma 4.1.

$$\frac{k_i(\mathscr{I}')}{C} \leq \max\left\{\frac{p_i^{\text{OPT}}(\mathscr{I})}{P}, \frac{k_i^{\text{OPT}}(\mathscr{I})}{C}\right\} \qquad (6)$$

Proof. For each job i of size s_i on the original instance $\mathscr{I}$, let the pair p_i^{GS}, k_i^{GS} be such that

$$(p_i^{\text{GS}}, k_i^{\text{GS}}) = \arg \min_{(p,k):s_i \leq Wk\,\log(1+g_i p/k)} \left[\frac{p}{P} + \frac{k}{C}\right] \qquad (7)$$

where p_i^{GS}, k_i^{GS} are the total power and codes assigned to user i. This solution allocates a power per code $p_i^s =$

$p_i^{\text{GS}}/j_i^{\text{GS}}$ to the ith job/user. Of course, it is possible that $k_i^{\text{GS}} > C$. To have a feasible allocation, for each code that uses power p_i^s we allocate $r_i^s = \lceil p_i^s/(P/C) \rceil$ codes with power P/C in each code. Because $r_i^s \geq p_i^s/(P/C)$,

$$s_i \overset{(a)}{\leq} Wk_i^{\text{GS}}\log(1 + p_i^s g_i) \quad \leq Wk_i^{\text{GS}}\log\left(1 + r_i^s\frac{P}{C}g_i\right)$$

$$\leq Wk_i^{\text{GS}}\tau_i^s\log\left(1 + \frac{P}{C}g_i\right) \qquad (8)$$

where (a) is due to (7). Hence, using this allocation the demand s_i of each user i is satisfied. As a result, we can give $k_i(\mathscr{I}) = k_i^{\text{GS}}\lceil (p_i^{\text{GS}}/k_i^{\text{GS}})/(P/C) \rceil$ codes with power P/C each and still complete the job. Therefore, for the P/C allocation, if we use $k_i(\mathscr{I})$ codes for job i, we obtain,

$$\frac{k_i(\mathscr{I})}{C} = \frac{k_i^{\text{GS}}}{C}\left\lceil\frac{p_i^{\text{GS}}/k_i^{\text{GS}}}{P/C}\right\rceil \leq \left(\frac{p_i^{\text{GS}}}{P} + \frac{k_i^{\text{GS}}}{C}\right) \qquad (9)$$

We now relate this to the optimal solution on the instance $\mathscr{I}$. Recall that the size of each job is reduced by 1/2 in $\mathscr{I}'$, with respect to $\mathscr{I}$. Therefore,[12] considering the P/C allocation done for jobs in $\mathscr{I}'$ we have,

$$\frac{k_i(\mathscr{I}')}{C} \leq \frac{1}{2}\frac{k_i(\mathscr{I})}{C} \quad \leq \frac{1}{2}\left(\frac{p_i^{\text{GS}}(\mathscr{I})}{P} + \frac{k_i^{\text{GS}}(\mathscr{I})}{C}\right)$$

$$\leq \frac{1}{2}\left(\frac{p_i^{\text{OPT}}(\mathscr{I})}{P} + \frac{k_i^{\text{OPT}}(\mathscr{I})}{C}\right)$$

$$\leq \max\left\{\frac{p_i^{\text{OPT}}(\mathscr{I})}{P}, \frac{k_i^{\text{OPT}}(\mathscr{I})}{C}\right\} \qquad (10)$$

∎

Observe that $k_i(\mathscr{I}')/C$ is the number of time slots needed to satisfy the ith request in the reduced instance, using a P/C allocation.

Algorithm FIFO(P/C) schedules the jobs in order of release time, assigning a number of codes $k_i(\mathscr{I}')$ with power allocation P/C until 50% of the original demand is met. Its operation can be summarized as follows:

---

[11] Recall that the maximum power and codes available in a slot are P and C, respectively.

[12] Note that the optimal assignment need not necessarily assign equal power per code across the time slots where it schedules the jth user. However, due to the joint concavity of the rate in terms of power and code assignment (see in proof of Theorem 3.3) the rate for a given user can only increase by giving equal power assignment per code across time slots, provided the power constraint is satisfied. Therefore, the optimal solution has a tighter constraint than the minimization in (7), and hence, the third inequality in (10) is satisfied.

1. When a request i is presented, find the number $k_i(\mathcal{I}')$ of codes needed to complete the ith demand, reduced by 50% with P/C power per code.
2. When a job is completed select the pending user request, if any, with earliest release time.

Observe that, as remarked above, a job may be served using more than C codes, over more than one time slot.

Theorem 4.2.

$$f^{\text{FIFO}}(\mathcal{I}') \leq f^{\text{OPT}}(\mathcal{I}) + 2, \ \forall \ \mathcal{I}. \tag{11}$$

Proof. Consider the request r achieving the maximum flow time for the scheduling discipline FIFO(P/C) applied to instance $\mathcal{I}'$. W log, we assume that request r is the last request presented to the algorithm. Request r has been released in slot t_r and completed in slot $C_r(\mathcal{I}')$ in the algorithm's solution. Denote by t (the renewal time) the last slot in which all requests that have been presented before time t have been completed by slot t. We can restrict our attention to the subset of user requests, denoted by $\mathcal{J} = \{i \in \mathcal{I} | a_i \geq t\}$, that have been presented at or after slot t, because these are the only requests that contribute to the flow time of request r. The completion time for request r using the FIFO(P/C) discipline on instance $\mathcal{I}'$ is at most $c_r^{\text{FIFO}}(\mathcal{I}') \leq t + \lceil (\sum_i k_i(\mathcal{I}'))/C \rceil$. Denote by s, the user request completed last in the solution of the optimum on instance $\mathcal{I}$. Request s has been released at some time $t_s \leq t_r$ and therefore, $c_s^{\text{OPT}}(\mathcal{I}) - t_s \geq t - t_s + \Psi$, where

$$\Psi = \max\left\{ \left\lceil \frac{\sum_i p_i^{\text{OPT}}(\mathcal{I})}{P} \right\rceil, \left\lceil \frac{\sum_i k_i^{\text{OPT}}(\mathcal{I})}{C} \right\rceil \right\}.$$

Now, we have

$$f^{\text{FIFO}}(\mathcal{I}') = c_r^{\text{FIFO}}(\mathcal{I}') - t_r$$

$$\leq t - t_r + \left\lceil \frac{\sum_i k_i(\mathcal{I}')}{C} \right\rceil \overset{(a)}{\leq} t - t_s + \Psi \tag{12}$$

$$\leq c_s^{\text{OPT}}(\mathcal{I}) - t_s + 2 \leq f^{\text{OPT}}(\mathcal{I}) + 2,$$

where (a) follows from Lemma 4.1 giving us the result. ∎

This result shows that by reducing the demand, we can prove a positive result on FIFO(P/C). Next, we explore the benefits of resource augmentation. For $P' \geq P$ and $C' \geq C$, we denote by FIFO(P'/C') the FIFO heuristics, as described above, when the on-line is provided with power P' and a number C' of codes per time slot. As proved by the following theorem, a positive result can be shown for FIFO(P'/C'). In both the statement and proof of Theorem 4.3 we use FIFO to mean FIFO(P'/C').

Theorem 4.3. $\exists P' \leq 2P, C' \leq 2C$ such that,

$$f^{\text{FIFO}}(\mathcal{I}) \leq f^{\text{OPT}}(\mathcal{I}), \ \forall \ \mathcal{I}. \tag{13}$$

Proof. Denote by $p_i^{j,\text{OPT}}(t)$ the power assigned by OPT to the ith request on the jth code during time slot t. Correspondingly, we assign $r_i^j(t) = \lceil p_i^{j,\text{OPT}}(t)/(P/C) \rceil$ codes with power P/C per code. Due to the power constraint, we have for the tth time slot: $\sum_i \sum_{j=1}^{C} p_i^{j,\text{OPT}}(t) \leq P$. For every slot t, we can now easily give an upper bound to the total number of codes needed with the P/C allocation:

$$\sum_i \sum_{j=1}^{C} r_i^j(t) \leq \sum_i \sum_{j=1}^{C} \left\lceil \frac{p_i^{j,\text{OPT}}(t)}{P/C} \right\rceil + C \leq 2C, \tag{14}$$

where the second inequality is due to the power constraint. Therefore, there exists a P/C allocation that achieves the same schedule as the optimal but using power $P' \leq 2P$ and codes $C' \leq 2C$. The problem of scheduling jobs with P/C power per code is like a single processor scheduling problem [6], and FIFO is optimal for this problem with respect to maximum response time, proving the result. ∎

We now show how to transform our algorithms for the continuous case into algorithms for the discrete case. Recall that in the continuous case we assign power P/C to every code, but this may correspond to a nonfeasible transmission rate at the receiver for some specific user. To move from the continuous to the discrete case, we need to round the power assignment to a value that sustains one of the allowed discrete transmission rates.[13]

We propose a rounding scheme that allows us to turn a solution for the continuous case into a solution for the discrete case. We consider two possible kinds of rounding of a power z assigned to a code:

1. *Round up:* If there exists a power $\bar{z}_1 \in (z, 2z]$ corresponding to a discrete rate, then assign power $\bar{z}_1$ to the code.
2. *Round down:* If there exists a power $\bar{z}_2 \leq z$ corresponding to a discrete rate, then assign power $\bar{z}_2$ per code.

For each user we choose the rounding that would give the higher rate, provided the user were given all the resources, that is, all the power and codes. We denote by FIFO$_{\text{disc}}(P/C)$, the scheduling discipline obtained by taking the discipline FIFO(P/C) and applying the above rounding procedure. It is easy to see that the following result holds.

---

[13] For the P/C allocation we impose a further regularity condition that the lowest discrete rate $R(1) \leq W \log(1 + (gP/C))$, that is, there is a feasible discrete rate below this power allocation. Although this is perhaps a little more stringent than required, it makes the analysis simpler. As before, this restriction can be removed in practice by using error-correcting codes on a group of codes, so that the combined rate is a feasible discrete rate.

Lemma 4.2. *The allocation scheme for the discrete case satisfies all users demands.*

Proof. User i is allocated with power per code x. For every code allocated with exactly x, rounding up will increase the transmission rate achieved on a code by a user. Rounding down will result, by Fact 3.1, in a transmission rate that is at least half of the transmission rate in the continuous case. Therefore, by assigning two codes, the transmission rate for the user does not decrease. Hence, the demand is satisfied by the rounding scheme. ∎

Now, we show that the approximations shown for the continuous case can be translated to the case of a discrete rate set by additional resource augmentation. The idea of the proof is to show that the additional power and codes needed in the rounding procedure above, for the discrete schedule to be as good as the continuous case, are properly bounded. In both the statements and proofs of Theorems 4.4 and 4.5 we use FIFO to mean FIFO(P'/C').

Theorem 4.4. $\exists P' \leq 2P$, $C' \leq 2C$ *such that,*

$$f^{\text{FIFO}_{\text{disc}}}(\mathcal{I}') \leq f^{\text{OPT}}(\mathcal{I}) + 2, \ \forall \ \mathcal{I}. \tag{15}$$

Proof. Let $\mathcal{K}_1$ ($\mathcal{K}_2$) denote the sets of codes whose associated power was rounded up (respectively, down) in a particular time slot t. We are interested in the power assigned to a particular code j in time slot t after rounding, regardless of the particular request it serves. For this reason, we denote by p_j this power and it is understood that we refer to code j and time slot t. Clearly, $|\mathcal{K}_1| + |\mathcal{K}_2| \leq C$. Furthermore, $p_j \leq 2P/C$, $j \in \mathcal{K}_1$ and $p_j \leq P/C$, $j \in \mathcal{K}_2$. Now, suppose in each time slot, for every code $j \in \mathcal{K}_2$ we assign two codes with power p_j, and for each code $j \in \mathcal{K}_1$ (whose power was rounded up) we assign one code. Clearly, this allocation will meet the same demand as the continuous rate FIFO(P/C) schedule; hence, the two schedules are equivalent. Using this and Theorem 4.2, (15) can be obtained. The only question that remains is how much resource augmentation was done to obtain this. In the new allocation and in the considered time slot, we have used

$$P' = \sum_{j \in \mathcal{K}_1} p_j + 2 \sum_{j \in \mathcal{K}_2} p_j$$

total power and $C' = |\mathcal{K}_1| + 2|\mathcal{K}_2|$ total codes. But we have,

$$P' \leq 2 \frac{P}{C} |\mathcal{K}_1| + \frac{P}{C} 2|\mathcal{K}_2| \leq 2P$$
$$C' = |\mathcal{K}_1| + 2|\mathcal{K}_2| \leq 2C.$$
$$\tag{16}$$

Hence, the new allocation uses at most a power $P' \leq 2P$ and a number of codes $C' \leq 2C$ in each time slot. ∎

We can also extend the result in Theorem 4.3 to the discrete rate with more resource augmentation. This is given in Theorem 4.5 below, whose proof proceeds the same as Theorem 4.4, and is therefore omitted.

Theorem 4.5. $\exists P' \leq 4P$, $C' \leq 4C$ *such that,*

$$f^{\text{FIFO}_{\text{disc}}}(\mathcal{I}) \leq f^{\text{OPT}}(\mathcal{I}), \ \forall \ \mathcal{I}. \tag{17}$$

4.2. Other QoS Criteria

Although we focused on minimizing the maximum response time in this article, we can extend our results to other optimization criteria such as *minimizing total weighted response time,* $\Sigma_i w_i(c_i - a_i)$, where arbitrary weights w_i are specified for each request i. We also consider the special case of average flow time, where $w_i = 1/n$, for n jobs. For average flow time, we analyze the Shortest Remaining Processing Time (SRPT) heuristic, which is optimal for the problem of minimizing the average flow time on a single machine [3]. For weighted response time, we analyze Highest Density first (HDF), which at any time t schedules the pending request with maximum ratio w_i/s_i, which is defined as the *density*. More in detail, for each instance $\mathcal{I}$ and job $i \in \mathcal{I}$, the number of codes needed to serve i using a P/C allocation is first calculated. This also allows to calculate the number of time frames needed to complete i using a P/C allocation. Jobs are then scheduled with this power/code allocation using the SRPT (respectively, HDF) heuristic. In the case of SRPT, this means that at any time the job with the smallest remaining number of frames to complete is scheduled. We have the following results.

Theorem 4.6. *Algorithm SRPT achieves the optimum average flow time if every user demand is reduced by a factor* $1/2$.

Proof. Consider user i and denote by f_i^{red} and by f_i^{OPT}, respectively, the number of frames used for user i by the algorithm working on instance $\mathcal{I}'$ (demands reduced by 50% in size) and a lower bound on the number of frames used for the same request by the optimum working on the original instance $\mathcal{I}$.

From Lemma 4.1 it follows

$$f_i^{\text{red}} \leq \max\left\{ \frac{p_i^{\text{OPT}}}{P}, \frac{k_i^{\text{OPT}}}{C} \right\} \leq f_i^{\text{OPT}}$$

We know that SRPT on allocation $\{f_i^{\text{OPT}}\}$ gives an optimal solution. Consider the schedule produced by SRPT on $\{f_i^{\text{OPT}}\}$ and stop processing request i when it has been allocated f_i^{red} time slots. This new schedule has an average flow time certainly smaller than SRPT on $\{f_i^{\text{OPT}}\}$, from which the theorem follows. ∎

Theorem 4.7. *For any $\epsilon > 0$, Highest Density first is a $1 + \epsilon/\epsilon$ approximation for minimizing the weighted flow time if every request is guaranteed for a fraction $1/(2(1 + \epsilon))$ of the original demand.*

Proof. For every job i, we denote by f_i^{red} and by f_i^{OPT}, respectively, the maximum number of frames used by the algorithm working with reduced demands and a lower bound on the number of frames used by the optimum to serve the ith request. Observe that, if we denote by f_i' the number of time slots needed to serve the ith request when each demand is reduced by 50% in size and a P/C allocation is used, we have $f_i^{\text{red}} \le (1/(1 + \epsilon))f_i'$. Also, from Lemma 4.1, $f_i' \le f_i^{\text{OPT}}$, for which $f_i^{\text{red}} \le (1/(1 + \epsilon))f_i^{\text{OPT}}$. The optimum has to allocate at least f_i^{OPT} frames to request i to meet its demand.

For the sake of analysis, we compare HDF with a lower bound on the optimum given by a fractional version of HDF, denoted in the following by FHDF. HDF and FHDF work the same way, but FHDF is able to reduce the weight of a job fractionally, as the job is processed. In particular, if an amount s of job i has already been scheduled by frame t, then the weight of job i is reduced to $w_i^{\text{FHDF}}(t) = w_i(s_i - s)/s_i$. The size of job i at time t is analogously reduced to $s_i(t) = s_i - s$. Observe $w_i(t)/s_i(t) = w_i/s_i$; hence, the density of a job remains constant along the execution of FHDF. This implies that FHDF schedules jobs in the same order as HDF.

The maximum ratio between the weights not completed by HDF and FHDF at any time t is an upper bound on the approximation ratio of the two algorithms. Because $f_i^{\text{red}} \le (1/(1 + \epsilon))f_i^{\text{OPT}}$ and both FHDF and HDF schedule jobs in the same order, the number of frames allocated by FHDF to any request i by time t is never larger than the number of frames allocated to request i by HDF. It follows that the jobs that are completed by FHDF at any time t are also completed by HDF within the same time. As a consequence, the ratio between the uncompleted weight of the two algorithms is given by the maximum over all jobs i of $w_i/(w_i^{\text{FHDF}}(t))$. Assume the worst case, in which both HDF and FHDF started job i at the same time. When HDF is about to complete request i, FHDF is still left with at least $(\epsilon/(1 + \epsilon))f_i^{\text{OPT}}$ frames to be allocated to request i; therefore, with a fraction at least $\epsilon/(1 + \epsilon)$ of the original weight of request i. The ratio between the remaining weight of HDF and FHDF is then bounded by

$$\frac{w_i}{w_i^{\text{FHDF}}(t)} \le \frac{1 + \epsilon}{\epsilon},$$

for which the claim of the theorem holds. ∎

The Highest Density first heuristic, when specialized to average stretch and average response time, becomes the Shortest Processing Time first heuristic (SPT), that at any time schedules the pending request that has shortest pro-cessing time. (Note that the selection policy still applies to the SPT algorithm.)

Corollary 4.1. *For any $\epsilon > 0$, Shortest Processing Time first is a $(1 + \epsilon)/\epsilon$ approximation for minimizing the average stretch (and response time) if every request is guaranteed for a fraction $1/(2(1 + \epsilon))$ of the original demand.*

5. SIMULATION STUDY

In this section, we study the performance of our on-line and off-line algorithms experimentally.

5.1. On-Line Algorithms

The FIFO(P/C) algorithm was described in Section 4. We call this algorithm FIFO-*continuous*. Essentially, this algorithm allots P/C power to each code, and job requests are then scheduled in the order of their arrival.

The rounding procedure for covering the continuous power (rate) algorithm to a discrete power (rate) algorithm was described in Section 4.1. Because rounding the rates might result in some power and/or codes to be unused in a slot, we design discrete-rate on-line algorithms to minimize this potential waste of resources to reduce the maximum response time. We have developed three on-line discrete-rate algorithms, which we call FIFO, 2D-FIFO, and 2D-PIKI. Given a job, the power per code corresponding to the discrete bit rate is the same for all of these algorithms. They differ only in the way the jobs are selected for receiving service.

5.1.1. FIFO. This is the traditional FIFO algorithm. The request i currently in the system that has the earliest release time a_i is always selected.

5.1.2. 2D-FIFO. The request i currently in the system that has the earliest release time a_i has higher priority over other job requests. However, if job i leaves power/codes unused in that time slot, other jobs h in the system are considered in the nondecreasing order of their release times a_h.

5.1.3. 2D-PIKI. The request i currently in the system that has the highest value of power per code p_i is selected. If this job leaves power/codes unused in that time slot, other jobs i in the system are considered in the nonincreasing order of the power per code p_i. This scheme aims to achieve a better packing in each time slot, to reduce the completion time.

Due to discrete nature of the rate set, in certain slots the scheduler may have some codes k^{extra} and some power p^{extra} that cannot be assigned to any job in the system, because the power per code $p_i > p^{\text{extra}}$ for all jobs i. In such a situation, the scheduler will choose the first job that received service in the slot and give it the best possible discrete rate with the remaining power and codes. This is applicable to all the three algorithms described above.

| Trace | OPT (in slots) | Continuous | Discrete | | |
		FIFO	FIFO	2D-FIFO	2D-PIKI
config1	109257	109891	120682	114054	114587
config2	50249	50637	55281	52263	59467
config3	36460	36725	40325	38540	46432
config4	16224	16254	17280	17210	24711

Note that no algorithm guarantees that all the power and codes will be used in every slot. Therefore, we expect to see differences between the FIFO-*continuous* algorithm and the three discrete-rate algorithms. In the remainder of this section, we will quantify the differences through simulations.

5.2. Channel Specifications

We adopt the channel specifications similar to 3G system proposals [7, 20] for our channel model.[14]

We perform experiments on a single cell and abstract the effect of out-of-cell interferers into a decrease in SINR values. The peak power available at the base station was chosen to be $P = 40W$, while the maximum number of channels was chosen as $C = 16$. The power attenuation factor $\bar{g}_u$ for user u is modeled with two components: (a) shadow loss component S, which is a log-normal shadowing variable, and (b) path loss components $P = 1/d^{\alpha}$, where d is the distance between the base station and the user and α is the distance loss exponent. We chose $\alpha = 3$, giving $\bar{g}_u \propto S/d_u^3$.

The parameters to be used for the rate calculation given in Equation (2) were chosen as follows: $\tau = 1.67$ milliseconds, $\bar{W} = 76.8$ KHz and $\Gamma = 4.7$ dB. We operated over an SINR range from -15 dB/Hz to 15 dB/Hz. The discrete rate set used is a set of 15 rates: {2.4, 4.8, 9.6, 19.2, 38.4, 76.8, 102.6, 153.6, 204.8, 307.2, 614.4, 921.6, 1228.8, 1843.2, 2457.6} Kbps. Under these restrictions, the maximum data rate for a mobile user in the cell will range from 10 Kbps to 2 Mbps.

5.3. Data Sets and Simulation Tools

The traces used in the experiments are derived from a single Web-proxy server. In comparing the on-line algorithms with the off-line optimum, we used traces that consist of up to 100 jobs that arrive over a period of 1 minute.[15] To evaluate the performance of various on-line algorithms under heavy demand, we used traces consisting of 4000 jobs arriving over a period of 35–40 minutes, where the requests

are generated by 100 users in the cell. In all of the traces used, the minimum request size was 40 bytes, the maximum request size was 500 kilobytes, with mean request sizes ranging from 20–34 kilobytes. The average interarrival time of requests in the traces is 500–600 milliseconds.

To compute the optimum flow in the off-line case, we used an optimization tool called LOQO (LOQO Optimization Toolkit: http://www.orfc.princeton.edu/logo, 2000). Our on-line algorithms were evaluated using a custom-built simulator.

5.4. Experiments

We performed two kinds of experiments to evaluate our algorithms. The first set of experiments validate our theoretical results and demonstrate some interesting properties of the on-line algorithms, while the second experiment was designed to measure the average-case performance of our algorithms.

5.4.1. On-Line Heuristics. In this section, we evaluate the different on-line algorithms and compare their performance against the off-line optimal algorithm. We used small Web traces, with 100 jobs arriving over a period of 1 minute, for computing the convex programming lower bound for max-flow (see Section 3.3), which we denote by OPT. The job requests are for users who are distributed uniformly in the cell. We present the max-flow results for four such traces along with the results for the on-line heuristics in Table 2: all max-flow values are in terms of slots.

It can be seen that the on-line algorithms perform very close to the optimal, on the average. From the table, we also see that 2D-FIFO performs the best among the three discrete-rate algorithms. In addition, the discrete algorithms always appear to perform worse than the continuous version.

These inferences continue to hold true in most instances, as we will show in the subsequent examples. We simulated the three algorithms FIFO-*continuous,* 2D-FIFO, and 2D-PIKI on 36 traces of 4000 requests each generated by 100 users distributed over the cell. Figure 1 shows the values of the max-flow computed for all the 36 traces by the three algorithms. One can still observe that 2D-FIFO is very close to FIFO-*continuous* on all traces, while 2D-PIKI performs the worst.

Although these results hold on the average, we give a

---

[14] We would like to emphasize that our algorithms are applicable to all systems that support multiple channels and multiple rates. Such systems include the various next-generation wireless data networks.

[15] The small size of these traces was primarily due to the computational restrictions on finding the optimal flow.
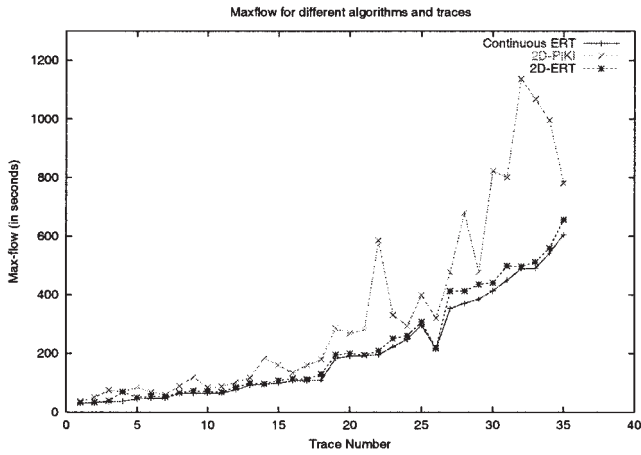
FIG. 1.   On-line heuristics: performance.

FIG. 3.   On-line heuristics: reduced demand and resource augmentation.

cautionary note that there can be examples where this average behavior is violated. We designed synthetic traces where 2D-PIKI performs as well as the optimal algorithm and better than the 2D-FIFO algorithm.

5.4.2. Resource Augmentation.   In this set of experiments, we will examine the amount of resource augmentation needed for a discrete-rate algorithm to achieve the sme max-flow as FIFO-*continuous* and compare it to theoretical bounds given in Theorems 4.4 and 4.5.

Using the same set of 36 Web traces described in Section 5.4.1., the scheduling algorithms were provided with augmented power in steps {P, $1.25P$, $1.5P$, $1.75P$, $2P$} and augmented number of codes in steps {C, $1.5C$, $2C$, $3C$}.

In the first experiment we measure the worst-case ratio between the max flow-time of 2D-FIFO and FIFO-*continuous*. We tested the 2D-FIFO algorithm, because it outperforms the other algorithms in the average case. The results of the first experiment are summarized in the 3D graphic of Figure 2. We observe on all traces that 2D-FIFO obtains a
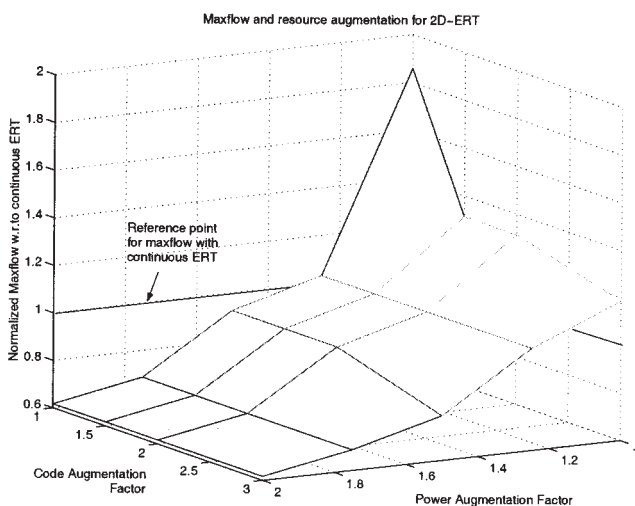
max-flow time lower than FIFO-*continuous* with power augmentation factor 1.5. We also found that a code augmentation factor of 4 is needed (without power augmentation) to obtain max-flow time lower than FIFO-*continuous*. Therefore, augmenting codes is not as efficient as augmenting power. Another way to observe this is by examining (4) where the rate is $R(p, c) = Wc \log(1 + gp/c)$. Here, one can see that $R(p, c) \leq Wgp$, $\forall c$; hence, even with a large number of codes, the rate is bounded above. In contrast, the rate is an unbounded function of p.

In a second experiment, for each combination of augmented power and codes we reduce the demand in steps {s_i, $0.95s_i$, $0.9s_i$, $0.85s_i$, $0.8s_i$, $0.75s_i$, $0.7s_i$, $0.6s_i$} until we find the max-flow for the reduced demand in the discrete case to be lesser or equal than the max-flow computed by FIFO-*continuous* with power P and codes C at 100% of the demand. The lower hull over all traces of the reduced demand for each combination of augmented code and power was taken. This represents the maximum demand reduction for a given combination of augmented resources, as shown in Figure 3.

Two observations can be made here: (a) *The average case is better than the worst-case:* in Figure 3, we observe that if power is augmented $1.25 P$, then the max-flow in the discrete case with 95% of the original demand equals that of the continuous case with power P. (b) *Code augmentation and power augmentation are not the same.* As can be seen from Figures 2 and 3, the asymmetric nature of the graphs tells us that overprovisioning codes is not very efficient compared to overprovisioning of power.

In conclusion, our experimental results demonstrate that the discrete-rate algorithms proposed in this section perform closer to the continuous-rate case than the worst-case analyses indicate. We also show that resource augmentation, in particular, power augmentation, will considerably enhance the performance of discrete algorithms.

FIG. 2.   On-line heuristics: max-flow with resource augmentation.

REFERENCES

[1] S. Acharya and S. Muthukrishnan, Scheduling on-demand broadcasts for heterogenous: New metrics and algorithms, ACM MobiCom (1998), 43–54.

[2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, Complexity and approximation: Combinatorial optimization problems and their approximability properties, Springer-Verlag, Berlin, 1999.

[3] K.R. Baker, Introduction to sequencing and scheduling, Wiley, New York, 1974.

[4] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz, A comparison of mechanisms for improving TCP performance over wireless links, IEEE/ACM Trans Networking 5 (1997), 756–769.

[5] N. Bambos, Toward power-sensitive network architectures in wireless communications: Concepts, issues, and design aspects, IEEE Pers Commun 5 (1998), 50–59.

[6] M. Bender, S. Chakrabarti, and S. Muthukrishnan, Flow and stretch metrics for scheduling continuous job streams, Proc of Annual Symposium on Discrete Algorithms (SODA '98), 1998, pp. 270–279.

[7] P. Bender, P. Black, M. Grob, R. Padovani, N. Sindhushayana, and A. Viterbi, CDMA/HDR; A bandwidth-efficient high-speed wireless data service for nomadic users, IEEE Commun Magazine 38 (2000), 70–77.

[8] S. Chakrabarti and S. Muthukrishnan, Resource scheduling for parallel database and scientific applications, Proc of the ACM Symposium on Parallel Algorithms and Architectures, 1996, pp. 329–335.

[9] J.M. Cioffi, G.P. Dudevoir, M.V. Eyuboglu, and G.D. Forney, MMSE decision-feedback equalizers and coding: Parts I & II, IEEE Trans Comm 43 (1995), 2582–2604.

[10] T.M. Cover and J.A. Thomas, Elements of information theory, John Wiley & Sons, New York, 1991.

[11] M.R. Garey and D.S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, W. H. Freeman, San Francisco, 1979.

[12] K. Jansen and L. Porkolab, Linear-time approximation schemes for scheduling malleable parallel tasks, Proc of the 10th ACM SIAM Symposium on Discrete Algorithms, 1999, pp. 490–498.

[13] N. Joshi, S.R. Kadaba, S. Patel, and G. Sundaram, Down-link scheduling in CDMA data networks, ACM MobiCom (2000), 179–190.

[14] B. Kalyanasundaram and K. Pruhs, Speed is a powerful as clairvoyance, IEEE Symp on Foundations of Computer Sci 1995, pp. 214–221.

[15] D. Karger, C. Stein, and J. Wein, Scheduling algorithms, Algorithms and theory of computation handbook, CRC Press, Boca Raton, FL, 1999.

[16] R. Lepère, D. Trystram, and G. Woeginger, Approximation algorithms for scheduling malleable tasks under precedence constraints, Proc of the 9th Annual European Symposium on Algorithms, LNCS 2161, 2001, pp. 146–157.

[17] Y. Lu and R.W. Brodersen, Integrating power control, error correction coding, and scheduling for a CDMA downlink system, IEEE Selected Areas Commun 17 (1999), 978–989.

[18] W. Ludwig and P. Tiwari, Scheduling malleable and non-malleable parallel tasks, Proc of the 5th ACM SIAM Symposium on Discrete Algorithms and Architectures, 1994, pp. 167–176.

[19] D.G. Luenberger, Linear and non-linear programming, 2nd ed., Addison-Wesley, Reading, MA, 1984.

[20] S. Nanda, K. Balachandran, and S. Kumar, Adaptation techniques in wireless packet data services, IEEE Commun Magazine 38 (2000), 54–65.

[21] T. Nandagopal, S. Lu, and V. Bharghavan, A unified architecture for the design and evaluation of wireless fair queueing algorithms, ACM Mobicom (1999), 132–142.

[22] S. Ramakrishna and J.M. Holtzman, A scheme for throughput maximization in a dual-class CDMA system, IEEE Select Areas Commun 16 (1998), 830–844.

[23] T.S. Rappaport, Wireless communications, principles, & practice, Prentice Hall, Inc., Piscataway, NJ, 1996.

[24] H. Shachnai and J.J. Turek, Multiresource malleable task scheduling to minimize response time. Informat Process Lett 70 (1999), 47–52.

[25] J. Turek, W. Ludwig, J.L. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schweigelshohn, and P.S. Yu, Scheduling parallelizable tasks to minimize average response time, Proc of the 6th Annual Symposium on Parallel Algorithms and Architectures, 1994, pp. 200–209.

[26] J. Turek, J. Wolf, and P. Yu, Approximate algorithms for scheduling parallel tasks, Proc of the 4th Annual Symposium on Parallel Algorithms and Architectures, 1992, pp. 323–332.