J Sched (2015) 18:107–118
DOI 10.1007/s10951-014-0371-3

# Scheduling to minimize energy and flow time in broadcast scheduling

**Benjamin Moseley**

Received: 17 December 2012 / Accepted: 13 February 2014 / Published online: 5 March 2014


**Abstract** In this paper, we initiate the study of minimizing power consumption in the broadcast scheduling model. In this setting, there is a wireless transmitter. Over time requests arrive at the transmitter for pages of information. Multiple requests may be for the same page. When a page is transmitted, all requests for that page receive the transmission simultaneously. The speed the transmitter sends data at can be dynamically scaled to conserve energy. We consider the problem of minimizing flow time plus energy, the most popular scheduling metric considered in the standard online scheduling model when the scheduler is energy aware. We will assume that the power consumed is modeled by an *arbitrary* convex function. For this problem, there is an $\Omega(n)$ lower bound on the competitive ratio. Due to the lower bound, we consider using resource augmentation and give a *scalable* algorithm.

**Keywords** Scheduling · Broadcast · Online · Resource augmentation · Energy · Speed scalable

## 1 Introduction

Wireless transmitters can typically be utilized in a variety of ways. This presents the designer with tradeoffs between power, signal range, bandwidth, cost, etc. In this paper, we consider the trade off between power and performance. Reducing the energy consumption is of importance in many systems. For example, in ad hoc wireless networks, a typical transmitter is battery operated, and therefore, energy conservation is critical. There are two modes of a wireless transmitter where power can be saved: (1) During idle times, (2) During transmit times. Work on the first mode focuses on putting the system to sleep when not in use (Chen et al. 2002; Xu et al. 2000). By appropriately putting the system to sleep, energy consumption can be drastically reduced. Another way the transmitter can reduce power is scaling the speed of the wireless transmission (Irani et al. 2007). By using more power, a signal can be sent at a faster rate or, to save power, the signal can be sent at a slower rate. We focus on reducing energy in the second mode. Reducing energy by changing the transmission speed is naturally related to the well studied model of speed scaling in scheduling theory. However, now that we are interested in wireless networks, this motivates a generalization of the standard speed scaling model.

Companies such as IBM and AMD are producing processors whose speed can be dynamically scaled by the operating system. A typical operating systems can control the power consumed in the system by scaling the processor speed. To model this in scheduling theory, it is assumed that there is a power function $P$ where $P(s)$ is the power used when running the processor at speed $s$. A scheduler not only chooses the job to schedule, but also the speed used to process the job. In other words, a scheduling algorithm consists of a job selection policy and an energy policy. This is known as the speed scaling scheduling model.

The standard speed scaling model is similar to our setting, except that we are interested in speed scaling in wireless networks. Like the speed scaling setting, we assume that there is a power function $P$ where $P(s)$ is the power used when transmitting at speed $s$. To model the wireless network, we will assume a well-studied model known as the broadcast model. In a broadcast network, there is a single server, and $n$ pages of information are stored at the server. Over time clients send requests for pages to the server. Multiple clients could be interested in the same page of information. When the

B. Moseley (✉)
Toyota Technological Institute, Chicago, IL 60637, USA
e-mail: moseley@ttic.edu

sever broadcasts a page $p$, all outstanding requests for that page are simultaneously satisfied. This is how the broadcast model differs from the standard scheduling model. In the standard model, each request (job) is for a unique page and broadcasting (processing) a page satisfies exactly one request. In the *online* setting, the server is not aware of a request until it arrives to the system. The broadcast model is motivated by applications in multicast systems and in wireless and LAN networks (Wong 1988; Acharya et al. 1995; Aksoy and Franklin 1999; Hall and Täubig 2003). Along with practical interest, the broadcast model has been studied extensively in the scheduling literature (Bar-Noy et al. 2002; Aksoy and Franklin 1999; Acharya et al. 1995; Bartal and Muthukrishnan 2000; Hall and Täubig 2003; Gandhi et al. 2006). Similar models have been considered in queuing theory and in the stochastic scheduling literature (Deb 1984; Deb and Serfozo 1973; Weiss 1979; Weiss and Pliska 1982).

The goal of the scheduler is to satisfy the requests in an order which optimizes a quality of service metric. Naturally, this metric depends on the needs of the system. When speed scaling is allowed, the server has a dual objective. One is to minimize the power usage, and the other is to optimize the quality of service the clients receive. Perhaps the most popular metric in the speed scaling literature is minimizing a linear combination of flow time and total energy (Albers and Fujiwara 2007; Bansal et al. 2009; Chan et al. 2009; Gupta et al. 2010; Bansal et al. 2009). The flow time[1] of a request is the amount of time the server takes to satisfy the request. Let $F$ denote the total flow time of a schedule over all requests and let $E$ denote the energy consumption of the schedule. The scheduler focuses on minimizing $G = F + E$. This has a natural interpretation. Say the system is willing to spend one unit of energy to reduce $\rho$ units of flow time. For example, a system designer may be able to justify spending 1 erg of energy to decrease the flow time by $\rho$ =10 μs . The system designer would then desire a schedule that minimizes $F + 10E$. By scaling the units of energy and flow time, we can assume that $\rho = 1$. The main contribution of this work is to initiate the study of energy in the broadcast model. We focus on the problem of minimizing flow time plus energy in the broadcast setting. Besides practical interest in the model, we believe this problem is of theoretical interest as it is a natural extension of previous work.

*Results* In this paper, we give an algorithm for minimizing total flow time plus energy in the online broadcast setting where the power function is an *arbitrary* convex function. There is an $\Omega(n)$ lower bound on the problem of minimizing flow time in broadcast scheduling when all broadcasts are sent at a fixed rate and, energy is not included in the objective (Kalyanasundaram et al. 2000). Since the power function is arbitrary, this lower bound also extends to the problem of minimizing flow time plus energy. Thus, we will resort to resource augmentation (Kalyanasundaram and Pruhs 2000). The resource augmentation model we consider is the same as that introduced in Gupta et al. (2010). Here, if the algorithm is given $1+\epsilon$ resource augmentation over the adversary, then our algorithm uses power $P(\gamma)$ when broadcasting at a rate of $(1+\epsilon)\gamma$. The rest of the paper will be devoted to proving the following theorem.

**Theorem 1.1** *There exists an algorithm that with* $1+\epsilon$ *resource augmentation is* $O\left(\frac{1}{\epsilon^3}\right)$*-competitive for minimizing total flow time plus energy in broadcast scheduling with an arbitrary power function where* $0 < \epsilon \leq 1$.

Our algorithm is called *scalable* since the minimum amount of resource augmentation is used to be $O(1)$-competitive. This is the best result that can be shown in the worst case setting up to a constant in the competitive ratio because there is a strong lower bound on an algorithm's competitive ratio when the algorithm is not given resource augmentation.

*Relation to previous work* In the standard scheduling setting, speed scaling has traditionally used power functions of the form $P(s) = s^\alpha$ for some $\alpha > 0$. These power functions were motivated by the fact that the power used by CMOS-based processors is approximately $s^3$. Recently, modeling the power function as an arbitrary convex function was suggested to capture the power consumption of more complex systems (Bansal et al. 2010). As mentioned, in this work, we adopt this general model.

There is a large amount of the literature on scheduling in the broadcast model. The most popular scheduling metric considered is minimizing average flow time. Recently, it was shown that there exists an online scheduler that is a scalable algorithm for minimizing the total flow time of a schedule when broadcasts are sent at a fixed rate, and energy is not considered in the objective (Im and Moseley 2012; Bansal et al. 2010). This work builds on the ideas and techniques given in Bansal et al. (2010). In particular, this paper uses the algorithm of Bansal et al. (2010) with the additional feature of being power aware. Since the power function considered in this paper is an arbitrary convex function, our work generalizes these results.

*Previous work on flow time in broadcast scheduling* Minimizing flow time (without energy) in broadcast scheduling where broadcasts are sent at a fixed speed has a rich history beginning with the seminal work of Kalyanasundaram et al. (2000); Bartal and Muthukrishnan (2000). Kalyanasundaram et al. (2000), showed that no online deterministic algorithm can be $\Omega(n)$-competitive. This has been extended to show that no randomized online algorithm can be $\Omega(\sqrt{n})$-competitive (Bansal et al. 2005). Due to these strong

[1] Flow time is also known as waiting time or response time.

lower bounds, most previous work has focused on analyzing algorithms in a resource augmentation model (Kalyanasundaram and Pruhs 2000). In this resource augmentation analysis, the online algorithm is given $s$-speed and is compared to a 1-speed adversary. An algorithm is said to be $s$-speed $c$-competitive for some objective function if the algorithm's objective when given $s$ speed is within a $c$ factor of the optimal solution's objective given 1-speed for every request sequence. An algorithm that is $(1+\epsilon)$-speed $O(1)$-competitive is called scalable where $0 < \epsilon \leq 1$ since it is $O(1)$-competitive when given the minimum advantage over the adversary.

In the offline setting, the problem was shown to be NP-Hard (Erlebach and Hall 2004; Chang et al. 2008). The first $O(1)$-speed $O(1)$-approximation was found in Kalyanasundaram et al. (2000). The best positive result in the offline setting using resource augmentation is a $(1+\epsilon)$-speed $O(1)$-approximation (Bansal et al. 2005). Without resource augmentation, (Bansal et al. 2005) gave an $O(\sqrt{n})$-approximation. This has since been improved to an $O(\log^2 n/\log\log n)$ approximation (Bansal et al. 2008). It is long standing open question whether or not this problem admits an $O(1)$-approximation.

It has been difficult to find competitive online algorithms for broadcast scheduling. The first online algorithm was given by Edmonds and Pruhs in Edmonds and Pruhs (2003). This algorithm was shown to be $(4+\epsilon)$-speed $O(1)$-competitive algorithm via a reduction to a different scheduling problem known as arbitrary speed up curves. This reduction takes a $s$-speed $c$-competitive algorithm for the speed up curves setting and converts it into a $(2s)$-speed $c$-competitive algorithm for the broadcast setting. In Edmonds and Pruhs (2005), Edmonds and Pruhs showed that a natural algorithm Longest-Wait-First (LWF) is 6-speed $O(1)$-competitive via a global charging argument. The analysis of LWF has been improved to show that the algorithm is $(3.4+\epsilon)$-speed $O(1)$-competitive (Chekuri et al. 2009a).

Later, Edmonds and Pruhs (2012) gave an algorithm LAPS that is scalable in the arbitrary speed up curves setting. Using the reduction in Edmonds and Pruhs (2003), this gives a $(2+\epsilon)$-speed $O(1)$-competitive algorithm for the broadcast setting. It was a long standing problem whether or not there existed a scalable online algorithm in the broadcast model. This question was resolved by Im and Moseley (2012); Bansal et al. (2010) by finding scalable algorithms. The analysis and algorithm of Bansal et al. (2010) has since been extended to show an algorithm that is $(2+\epsilon)$-speed $O(1)$-competitive for the $\ell_2$-norm of flow time (Gupta et al. 2010). Recently, a scalable algorithm was found for the $\ell_k$-norms of flow time for all $k \geq 1$ (Edmonds et al. 2010).

*Previous work on speed-scaling* All of the previous work on broadcast scheduling has assumed that the scheduler broadcasts at some fixed speed. Although speed scaling has not been considered in broadcast scheduling, it has been considered extensively in the standard scheduling model. As mentioned, the standard scheduling model can be interpreted as each request being for a unique page. Recall that in the speed scaling setting, a power function $P$ is given where $P(s)$ is the power used when running the processor at speed $s$. We first consider the traditional model where $P(s) = s^\alpha$ and $\alpha > 1$. In Pruhs et al. (2008), an efficient algorithm was given for the problem of minimizing flow time offline subject to a bound on the amount of power consumed. This algorithm can be extended to find a schedule that minimizes the flow time plus energy in the offline setting. The problem of minimizing flow time plus energy online was first studied by Albers and Fujiwara (2007). Bansal et al. (2009) showed that the algorithm that runs jobs with power proportional to the number of outstanding requests is 4-competitive for unit work jobs. They also showed the Highest-Density-First algorithm coupled with this power strategy is $O\left(\frac{\alpha^2}{\log^2 \alpha}\right)$ competitive for weighted flow plus energy. This is since been improved in Lam et al. (2008) for unweighted flow plus energy to give an $O\left(\frac{\alpha}{\log \alpha}\right)$-competitive algorithm. In Chan et al. (2009) an $O(\alpha^3)$-competitive *non-clairvoyant* algorithm was given.

Recently, Bansal et al. (2009) introduced the problem of minimizing flow time plus energy with an arbitrary convex power function. Surprisingly, they were able to give an algorithm that is 3-competitive in this general setting for flow plus energy. This resolved a central question on whether an algorithm could be $O(1)$-competitive where the competitive ratio does not depend on $\alpha$. Gupta et al. (2010) gave a scalable algorithm for minimizing weighted flow time plus energy in the case where there are $m$ machines, each machine may have a different power function, and each power function is an arbitrary convex function. In Gupta et al. (2012) several different objective functions are considered when the power function is an arbitrary convex function. As mentioned, in this paper, we adopt the assumption that the power function is an arbitrary convex function.

## 2 Preliminaries

We begin by introducing some notation. There are $n$ pages stored at the server. Each page has a size $\sigma_p$. A request for page $p$ is satisfied if it receives $\sigma_p$ pieces of page $p$ in sequential order. That is, a request receives the transmission of page $p$ from start to finish in that order. This will be further elaborated on later. Notice that by this definition, multiple requests can be satisfied by a single broadcast. Let $J_{p,i}$ denote the $i$th request for page $p$. Let $a_{p,i}$ be the time this request arrives to the server. In the online setting, this is the first time the server is aware of the request. Let $f_{p,i}$ be the time the server satisfies request $J_{p,i}$, the total flow time of a schedule is

$F = \sum_p \sum_i (f_{p,i} - a_{p,i})$. The following definitions will be useful.

**Definition 2.1 (convex function)** A real-valued function $f$ is convex if and only if for any $0 \leq \alpha \leq 1$ and any real valued $x$ and $y$ it is the case that $f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y)$

**Definition 2.2 (concave function)** A real-valued function $f$ is concave if and only if for any $0 \leq \alpha \leq 1$ and any real valued $x$ and $y$ it is the case that $f(\alpha x + (1-\alpha)y) \geq \alpha f(x) + (1-\alpha) f(y)$

Using the definition of a concave function, we can easily show the following proposition.

**Proposition 2.3** *For any real-valued concave function $f$ where $f(0) = 0$, the following holds*

- *For any positive real number $x$, $\frac{x-1}{x} \leq \frac{f(x-1)}{f(x)}$*
- *For any positive real number $x$ and any $\alpha \leq 1$, $\alpha f(x) \leq f(\alpha x)$*

We will be given a power function $P : \mathbb{R} \to \mathbb{R}$. The value of $P(s)$ is the power used when the server broadcasts with speed $s$. In Bansal et al. (2009); Gupta et al. (2010) with a small loss in the competitive ratio, it was assumed that power function $P$ satisfies the following conditions,

1. At all speeds, $P$ is defined, continuous, and differentiable
2. $P(0) = 0$
3. $P$ is strictly increasing
4. $P$ is strictly convex
5. $P$ is unbounded

In this paper we adopt these assumptions on $P$. Let the function $Q = P^{-1}$. That is, $Q(y)$ is the maximum speed of the processor with a limit of $y$ on the power used. Notice that $Q(0) = 0$ and that $Q$ is concave. Let $s(t)$ denote the speed used at time $t$. Let $E = \int_{t=0}^{\infty} P(s(t))\mathrm{d}t$ be the total energy used. The goal of the scheduler is to minimize $G = F + E$.

### 2.1 Fractional broadcast scheduling

In *integral* broadcast scheduling (the standard model), a request $J_{p,i}$ for page $p$ is satisfied if it receives each of $\sigma_p$ pieces of page $p$ in sequential order. That is, a page is divided into pieces. At any time the scheduler decides which piece to broadcast. A request is satisfied if it receives each of the pieces from start to finish, in that order. We note that in the previous literature, a slot model was considered where each page consists of unit sized pieces and in each time slot one unit-sized piece is chosen to be broadcasted. Since the speed of the broadcasts can be dynamically scaled over time, a slot model does not make sense for our setting. We will consider a continuos model where any portion of a page can be broadcasted and time is continuous and not slotted. Again, we require that a request is satisfied only if it receives all pieces of page $p$ in sequential order.

We now define a different version of the broadcast scheduling problem called *fractional* broadcast scheduling. In this setting, the $\sigma_p$ unit-sized pieces of page $p$ are indistinguishable. Now a request $J_{p,i}$ is satisfied once the server devotes a total of $\sigma_p$ time units to page $p$ after time $a_{p,i}$. In Bansal et al. (2010), it was shown that an online algorithm running at a fixed speed that satisfies a request $J_{p,i}$ by time $t$ in a fractional schedule can be converted into a different online algorithm that completes $J_{p,i}$ by time $t + \frac{3}{\epsilon'}(t - a_{p,i}) + \frac{5}{\epsilon'}$ in an integral broadcast schedule. Here, the new algorithm is given an additional $\epsilon'$ resource augmentation.

It is not obvious that this generalizes when the server can use varying speeds over time. In Sect. 4, we extend their result to the speed scaling setting. We prove the following theorem. This theorem may be of independent interest, since it can be used to reduce integral broadcast scheduling to the fractional setting for a variety of objective functions where speeds can vary over time.

**Theorem 2.4** *Let $S$ be a fractional broadcast schedule where the server can vary its speed over time and let $0 \leq \epsilon' \leq 1$. The schedule $\mathcal{S}$ can be converted into a schedule $\mathcal{S}'$ using $\epsilon'$ resource augmentation such that the schedule $\mathcal{S}'$ satisfies the following properties*

- *The power used by $\mathcal{S}'$ is at most twice the power used by $\mathcal{S}$.*
- *If a request $J_{p,i}$ is fractionally satisfied at time $f_{p,i}$ under $S$ then this request is integrally satisfied at time $f_{p,i} + \frac{5}{\epsilon'}(f_{p,i} - a_{p,i})$ under the schedule $S'$.*
- *If the algorithm that generates $S$ is online then so is the algorithm that generates $S'$.*

This theorem implies the following corollary.

**Corollary 2.5** *An algorithm with $1+\epsilon$ resource augmentation that is $c$-competitive for flow time plus energy in fractional broadcast scheduling can be converted into an algorithm that is $\left(\frac{5c}{\epsilon'} + 1\right)$-competitive for integral broadcast scheduling and uses $(1+\epsilon)(1+\epsilon')$ resource augmentation where $0 < \epsilon' \leq 1$.*

Due to the previous corollary, we will focus on fractional broadcast scheduling for the rest of this paper.

### 2.2 The algorithm

Let $N_a(t)$ contain the unsatisfied requests under our algorithm's schedule at time $t$. Our algorithm broadcasts at speed $Q(|N_a(t)|)$ at time $t$. Intuitively, since the flow time of the schedule at time $t$ increases by $|N_a(t)|$, the scheduler can afford to use this much power at time $t$. Now we define how our algorithm distributes its processing power. Here the algorithm BLAPS is used. This algorithm was introduced in Bansal et al. (2010); Edmonds and Pruhs (2012). We will be assuming that our algorithm is given $(1+6\epsilon)$ resource augmentation, and this implies the total speed our algorithm uses at time $t$ is $(1+6\epsilon)Q(|N_a(t)|)$ where $\epsilon \leq \frac{1}{6}$. The algorithm BLAPS takes a parameter $\beta \leq 1$. We will fix $\beta = \epsilon$ later. Let $N'_a(t)$ be the $\lceil \beta |N_a(t)| \rceil$ requests in $N_a(t)$ with the latest arrival times. For a moment, assume that $\beta|N_a(t)|$ is always integral. At any time $t$, the algorithm equally distributes its processing power amongst the requests in $N'_a(t)$. That is, for a request $J_{p,i} \in N'_a(t)$ page $p$ is broadcasted at a rate of $x_{p,i}(t) = \frac{(1+6\epsilon)Q(|N_a(t)|)}{\beta|N_a(t)|}$. Notice that there could be multiple requests for page $p$ in $N'_a(t)$. Let $S_p(t)$ be the set of requests for page $p$ in $N'_a(t)$. A page $p$ is broadcasted at a rate of $\sum_{J_{p,i} \in S_p(t)} x_{p,i}(t)$ at time $t$. We call $x_{p,i}(t)$ the amount $J_{p,i}$ contributes to how much page $p$ is broadcasted. Notice that at any time, the algorithm uses total speed $(1+6\epsilon)Q(|N_a(t))$.

Now we describe the algorithm in the case that $\beta|N_a(t)|$ is not integral. For the $\lfloor \beta|N_a(t)| \rfloor$ requests with latest arrival times in $N'_a(t)$, BLAPS keeps the value of $x$ the same. Let $J_{p,i}$ be the only other request in $N'_a(t)$. For this request, we set $x_{p,i} = (\beta|N_a(t)| - \lfloor \beta|N_a(t)| \rfloor)\frac{(1+\epsilon)Q(|N_a(t)|)}{\beta|N_a(t)|}$. That is, $J_{p,i}$ is given processing power proportional to $(\beta|N_a(t)| - \lfloor \beta|N_a(t)| \rfloor)$, the amount $J_{p,i}$ "overlaps" the $\beta|N_a(t)|$ latest arriving requests.

## 3 Analysis

We will be using a potential function argument. See Im et al. (2011) for a tutorial on potential function analysis in scheduling theory. Let $\frac{\mathrm{d}}{\mathrm{d}t}G(t)$ be the increase in our algorithm's objective at time $t$. Likewise, let $\frac{\mathrm{d}}{\mathrm{d}t}G^*(t)$ be the increase in OPT's objective at time $t$. Notice that $\frac{\mathrm{d}}{\mathrm{d}t}G(t) = 2|N_a(t)|$ because there are $|N_a(t)|$ outstanding requests at time $t$, which increases the flow time of the schedule by $|N_a(t)|$ and the scheduler uses power $|N_a(t)|$ at time $t$. Let $G = \int_0^\infty \frac{\mathrm{d}}{\mathrm{d}t}G(t)\mathrm{d}t$ denote our algorithm's total cost and let $G^* = \int_0^\infty \frac{\mathrm{d}}{\mathrm{d}t}G^*(t)\mathrm{d}t$ denote the optimal solution's total cost. We will define a potential function $\Phi(t)$ that will satisfy the following conditions:

1. *Boundary Condition* $\Phi$ is 0 before any request arrives and 0 after all requests complete
2. *Arrival/Completion Condition* $\Phi$ does not increase when a request is completed by BLAPS or OPT or when a request arrives.
3. *Running Condition* At all times $t$ it is the case that $\frac{\mathrm{d}}{\mathrm{d}t}G(t) + \frac{\mathrm{d}}{\mathrm{d}t}\Phi \leq c\frac{\mathrm{d}}{\mathrm{d}t}G^*(t)$ where $c$ is some positive constant.

By integrating the running condition over time, this is sufficient to show that our algorithm is $c$-competitive. This can be seen as follows,

$$G = \int_0^\infty \frac{\mathrm{d}}{\mathrm{d}t}G(t)\mathrm{d}t = \int_0^\infty \left(\frac{\mathrm{d}}{\mathrm{d}t}G(t) + \frac{\mathrm{d}}{\mathrm{d}t}\Phi(t)\right)\mathrm{d}t$$
$$\leq \int_0^\infty c\frac{\mathrm{d}}{\mathrm{d}t}G^*(t)\mathrm{d}t \leq cG^*.$$

The second equality holds due to $\Phi(0) = \Phi(\infty) = 0$.

Now we define our potential function. We assume without loss of generality that all requests arrive at distinct times, which will simplify the definition of the potential function. For a request $J_{p,i} \in N_a(t)$ let $\mathbf{rank}(J_{p,i}) = \sum_{J_{q,j} \in N_a(t), a_{q,j} \leq a_{p,i}} 1$ at time $t$ be the number of requests that have arrived during $[0, a_{p,i}]$ that are unsatisfied by the algorithm. Recall that $x_{p,i}(t)$ is the amount page $p$ is broadcasted by our algorithm at time $t$ due to $J_{p,i}$ and $\sigma_p$ is the amount of page $p$ that must be broadcasted after $a_{p,i}$ to satisfy $J_{p,i}$. Let $\mathtt{On}(p,i,t_1,t_2) = \int_{t_1}^{t_2} x_{p,i}(t)dt$. Let $y^*_p(t)$ be the amount of page $p$ that is broadcasted at time $t$ by OPT and let $\mathtt{Opt}(p,t_1,t_2) = \int_{t_1}^{t_2} y^*_p(t)dt$. We define the variable $z_{p,i}(t)$ for a request $J_{p,i}$ to be $\frac{\mathtt{On}(p,i,t,\infty)\mathtt{Opt}(p,a_{p,i},t)}{\sigma_p}$. Our potential function is,

$$\Phi(t) := \frac{1}{\epsilon}\sum_{J_{p,i} \in N_a(t)} \mathbf{rank}(J_{p,i})\left(\frac{z_{p,i}(t)}{Q(\mathbf{rank}(J_{p,i}))}\right).$$

Our potential function combines the potential functions of Bansal et al. (2010) and Gupta et al. (2010). The potential function of Bansal et al. (2010) was used for broadcast scheduling without energy, which needs to somehow have the power function reflected in the potential if it is to work in our setting. To incorporate the power function we use some ideas given in Gupta et al. (2010).

As is usually the case, our potential is designed to approximate the algorithm future cost after subtracting the optimal solution's future cost. Our algorithm gives higher priority to requests that have arrived recently. The potential function is designed to capture the remaining cost of the algorithm if it satisfies requests in the opposite order of

arrival. Fix a request $J_{p,i}$. If the optimal solution satisfied request $J_{p,i}$ then the value of $z_{p,i}(t)$ should be thought of as the remaining volume of page $p$ that must be broadcasted to satisfy request $J_{p,i}$. Assume $J_{p,i}$ has the highest rank in $N_a(t)$. Then $Q(\mathbf{rank}(J_{p,i}))$ is the speed that will be used at time $t$. The value of $\mathbf{rank}(J_{p,i})$ is the number of requests waiting for request $J_{p,i}$ to be satisfied. Thus, if requests are satisfied in opposite order of arrival, $\mathbf{rank}(J_{p,i})\left(\frac{z_{p,i}}{Q(\mathbf{rank}(J_{p,i}))}\right)$ is an estimate on the flow time that will be accumulated while the algorithm satisfies $J_{p,i}$ because it will take at least $\frac{z_{p,i}}{Q(\mathbf{rank}(J_{p,i}))}$ time units to satisfy request $J_{p,i}$.

### 3.1 Change of the potential function

It is easy to see that the potential function is not affected when the optimal solution completes a request. Also the potential function does not increase when a request $J_{p,i}$ arrives since $z_{p,i} = 0$. Further, the potential is 0 before all requests arrive, and after all requests are completed. We now show that $\Phi$ does not increase when a request is satisfied by the algorithm. This lemma, combined with the previous arguments shows that $\Phi$ satisfies the boundary and arrival/completion conditions.

**Lemma 3.1** *$\Phi$ does not increase when the algorithm completes a request.*

*Proof* Consider a time $t$ where that algorithm completes a request $J_{p,i}$. We can assume that $\mathbf{rank}(J_{p,i}) < |N_a(t)|$; otherwise, trivially there is no increase in $\Phi$. The change in $\Phi$ is,

$$\Delta\Phi(t) = \frac{1}{\epsilon}\sum_{J_{p',j}\in N_a(t), a_{p',j}>a_{p,i}}\left(\frac{(\mathbf{rank}(J_{p',j}) - 1)z_i}{Q(\mathbf{rank}(J_{p',j}) - 1)} - \frac{\mathbf{rank}(J_{p',j})z_i}{Q(\mathbf{rank}(J_{p',j}))}\right).$$

To show that $\Delta\Phi(t) \le 0$, we need to show that

$$\begin{aligned}
&\sum_{J_{p',j}\in N_a(t), a_{p',j}>a_{p,i}} \frac{(\mathbf{rank}(J_{p',j}) - 1)z_i}{Q(\mathbf{rank}(J_{p',j}) - 1)} \\
&\le \sum_{J_{p',j}\in N_a(t), a_{p',j}>a_{p,i}} \frac{\mathbf{rank}(J_{p',j})z_i}{Q(\mathbf{rank}(J_{p',j})} \\
&\Rightarrow \sum_{J_{p',j}\in N_a(t), a_{p',j}>a_{p,i}} \frac{(\mathbf{rank}(J_{p',j}) - 1)}{\mathbf{rank}(J_{p',j})} \\
&\le \sum_{J_{p',j}\in N_a(t), a_{p',j}>a_{p,i}} \frac{Q(\mathbf{rank}(J_{p',j}) - 1)}{Q(\mathbf{rank}(J_{p',j}))}.
\end{aligned}$$

This is true by definition of $Q$ and Proposition 2.3 □

Now we concentrate on the running condition, the final property of $\Phi$ that needs to be shown. Fix a time $t$. Let $N_o(t)$ be the set of requests unsatisfied by OPT at time $t$. First lets consider the change in $\Phi(t)$ due to the algorithm's processing. Recall that the algorithm broadcasts page $p$ at a rate of $x_{p,i}(t)$ due to a request $J_{p,i} \in N'_a(t)$. Further, notice that $\mathtt{Opt}(p, a_{p,i}, t) \ge \sigma_p$ for any request $J_{p,i} \in N_a(t) \setminus N_o(t)$, since OPT must broadcast $\sigma_p$ units of page $p$ after $a_{p,i}$ to satisfy page $p$. Therefore, for any $J_{p,i} \in N'_a(t)\setminus N_o(t)$ we have $\frac{\mathrm{d}}{\mathrm{d}t}z_{p,i}(t) \ge \frac{\mathrm{d}}{\mathrm{d}t}\mathtt{On}(p, i, t, \infty) = -x_{p,i}(t)$. Notice that the **rank** of each request the algorithm is currently working on is at least $(1-\beta)|N_a(t)|$. This is because $N'_a(t)$ consists of the $\lceil\beta|N_a(t)|\rceil$ requests with latest arrival times. Using this, we can determine a upper bound on the change in $\Phi(t)$ due to the algorithm's processing,

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\Phi &\le -\frac{1}{\epsilon}\sum_{J_{p,i}\in N'_a(t)\setminus N_o(t)} \mathbf{rank}(J_{p,i})x_{p,i}(t) \\
&\le -\frac{(1-\beta)|N_a(t)|}{\epsilon}\sum_{J_{p,i}\in N'_a(t)} x_{p,i}(t). \qquad (1)
\end{aligned}$$

Next, we consider the change in $\Phi$ due to the adversary's processing. Let $p^*$ be the page which the adversary broadcasts. Let $s_o(t)$ be the speed the optimal solution processes page $p^*$ at. Let $P^*(t) = P(s_o(t))$ be the power OPT uses at time $t$. The adversary can affect $z_{p^*,i}$ for any request $J_{p^*,i} \in N_a(t)$. We first observe the following,

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}t}\sum_{J_{p^*,i}\in N_a(t)} z_{p^*,i}(t) \\
&\le \sum_{J_{p^*,i}\in N_a(t)} \frac{\mathtt{On}(p^*, i, t, \infty)}{\sigma_{p^*}} \cdot \left(\frac{\mathrm{d}}{\mathrm{d}t}\mathtt{Opt}(p^*, a_{p^*,i}, t)\right) \\
&\le \frac{\mathrm{d}}{\mathrm{d}t}\mathtt{Opt}(p^*, a_{p^*,i}, t) = s_o(t)\mathrm{d}t.
\end{aligned}$$

The last inequality holds since $\sum_{i, J_{p^*,i}\in N_a(t)} \mathtt{On}(p^*, i, \infty) \le \sigma_{p^*}$. This is because the algorithm needs to only broadcast page $p^*$ for a total of $\sigma_{p^*}$ amount of time to satisfy all outstanding requests for page $p^*$. Let $J_{p^*,k}$ be the request in $N_a(t)$ such that $\frac{\mathbf{rank}(J_{p^*,k})}{Q(\mathbf{rank}(J_{p^*,k}))}$ is maximized. We can upper bound the increase in $\Phi$ due to OPT' processing as,

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\Phi &= \frac{1}{\epsilon}\left(\frac{\mathbf{rank}(J_{p^*,k})}{Q(\mathbf{rank}(J_{p^*,k}))}\right)\sum_{J_{p^*,i}\in N_a(t)} z_{p^*,i}(t) \\
&\le \frac{\mathbf{rank}(J_{p^*,k})s_o(t)}{\epsilon Q(\mathbf{rank}(J_{p^*,k}))}.
\end{aligned}$$

Our goal is to show that $\frac{\mathrm{d}}{\mathrm{d}t}G(t) + \frac{\mathrm{d}}{\mathrm{d}t}\Phi(t) \le \frac{2}{\epsilon^2}\frac{\mathrm{d}}{\mathrm{d}t}G^*(t)$. First we consider the case when the adversary uses power at least $|N_a(t)|$. In this case, the increase in the algorithm's

objective can be charged directly to the optimal solution, along with any increase in the potential function.

**Lemma 3.2** *If $Q(|N_a(t)|) \leq s_o(t)$ (equivalently, $P^*(t) \geq |N_a(t)|$) then $\frac{\mathrm{d}}{\mathrm{d}t}G(t) + \frac{\mathrm{d}}{\mathrm{d}t}\Phi(t) \leq \frac{2}{\epsilon}\frac{\mathrm{d}}{\mathrm{d}t}G^*(t)$.*

*Proof* First we bound the increase in $\Phi(t)$ due to OPT's processing. Intuitively, the increase in OPT's objective, due to using a lot of power, is large enough to absorb the increase in $\Phi$ and the increase in algorithm's objective. By increasing $\Phi$ now and charging it to OPT, we can later use the decrease in $\Phi$ to pay for increases in the algorithm's objective. Recall that $J_{p^*,k}$ is the request in $N_a(t)$ for page $p^*$ that maximizes $\frac{\mathbf{rank}(J_{p^*,k})}{Q(\mathbf{rank}(J_{p^*,k}))}$. Let $\alpha|N_a(t)| = \mathbf{rank}(J_{p^*,k})$ and let $\gamma = \frac{s_o(t)}{Q(|N_a(t)|)}$. Notice that $\alpha \leq 1$ and $\gamma \geq 1$. The function $Q$ is concave. Therefore, $Q(\mathbf{rank}(J_{p^*,k})) \geq \alpha Q(|N_a(t)|)$ by Proposition 2.3. The increase in $\Phi(t)$ due to OPT's processing can be bounded as follows,

$$\begin{aligned}
&\left(\frac{\mathbf{rank}(J_{p^*,k})}{\epsilon}\right)\frac{s_o(t)}{Q(\mathbf{rank}(J_{p^*,k}))}\\
&= \frac{1}{\epsilon}\alpha|N_a(t)|\gamma\frac{Q(|N_a(t))|}{Q(\mathbf{rank}(J_{p^*,k}))}\\
&\leq \frac{1}{\epsilon}\alpha|N_a(t)|\gamma\frac{Q(|N_a(t))|}{\alpha Q(|N_a(t)|)} \leq \frac{\gamma}{\epsilon}|N_a(t)|.
\end{aligned}$$

The total power used by OPT is at least $P^* \geq \gamma|N_a(t)|$ since $P$ is convex, and the speed OPT uses is $\gamma Q(|N_a(t)|)$. Hence, $\frac{\mathrm{d}}{\mathrm{d}t}G^*(t) \geq \gamma|N_a(t)|$. Recall that $\frac{\mathrm{d}}{\mathrm{d}t}G(t) = 2|N_a(t)|$. Knowing that $\epsilon \leq \frac{1}{6}$, we have that,

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}t}G(t) + \frac{\mathrm{d}}{\mathrm{d}t}\Phi(t) \leq 2|N_a(t)| + \frac{\gamma}{\epsilon}|N_a(t)| \leq \frac{2\gamma}{\epsilon}|N_a(t)|\\
&\leq \frac{2}{\epsilon}\frac{\mathrm{d}}{\mathrm{d}t}G^*(t).
\end{aligned}$$

□

Due to the previous lemma, for the rest of this paper we can concentrate on the case where $Q(|N_a(t)|) > s_o(t)$. We begin by bounding the increase in $\Phi(t)$ due to OPT's processing using this assumption,

$$\begin{aligned}
&\left(\frac{\mathbf{rank}(J_{p^*,k})}{\epsilon}\right)\frac{s_o(t)}{Q(\mathbf{rank}(J_{p^*,k}))}\\
&\leq \left(\frac{\mathbf{rank}(J_{p^*,k})}{\epsilon}\right)\frac{Q(|N_a(t)|)}{Q(\mathbf{rank}(J_{p^*,k}))}\\
&\leq \left(\frac{\mathbf{rank}(J_{p^*,k})}{\epsilon}\right)\frac{(|N_a(t)|/\mathbf{rank}(J_{p^*,k}))Q(\mathbf{rank}(J_{p^*,k}))}{Q(\mathbf{rank}(J_{p^*,k}))}\\
&\leq \frac{1}{\epsilon}|N_a(t)|. \qquad (2)
\end{aligned}$$

The first inequality holds since we assumed that $Q(|N_a(t)|) > s_o(t)$. The second inequality follows from definition of $Q$ and Proposition 2.3. We can now prove the final case of the running condition.

**Lemma 3.3** *If $s_o(t) < Q(|N_a(t)|)$ and $\beta = \epsilon$ then $\frac{\mathrm{d}}{\mathrm{d}t}G(t) + \frac{\mathrm{d}}{\mathrm{d}t}\Phi(t) \leq \frac{2}{\epsilon^2}\frac{\mathrm{d}}{\mathrm{d}t}G^*(t)$.*

*Proof* We know that $\frac{\mathrm{d}}{\mathrm{d}t}G(t) = 2|N_a(t)|$. The increase in $\Phi(t)$ due to OPT's processing is at most $\frac{1}{\epsilon}|N_a(t)|$, and the change in $\Phi(t)$ due to the algorithm's processing is at most $-\frac{(1-\beta)|N_a(t)|}{\epsilon}\sum_{J_{p,i}\in N'_a(t)\setminus N_o(t)} x_{p,i}(t)$. Combining these, we have the following.

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}t}G(t) + \frac{\mathrm{d}}{\mathrm{d}t}\Phi(t) \leq 2|N_a(t)| + \frac{1}{\epsilon}|N_a(t)|\\
&\quad -\frac{(1-\beta)|N_a(t)|}{\epsilon}\sum_{J_{p,i}\in N'_a(t)\setminus N_o(t)} x_{p,i}(t)\\
&\leq 2|N_a(t)| + \frac{1}{\epsilon}|N_a(t)|\\
&\quad -\frac{(1-\beta)|N_a(t)|}{\epsilon}\left(\sum_{J_{p,i}\in N'_a(t)} x_{p,i}(t) - \sum_{J_{p,i}\in N_o(t)\cap N'_a(t)} x_{p,i}(t)\right)\\
&\leq 2|N_a(t)| + \frac{1}{\epsilon}|N_a(t)|\\
&\quad -\left(\frac{(1+6\epsilon)(1-\beta)}{\epsilon}|N_a(t)| - \sum_{J_{p,i}\in N_o(t)\cap N'_a(t)} x_{p,i}(t)\right)\\
&\quad \left[\sum_{J_{p,i}\in N'_a(t)} x_{p,i}(t) = (1+6\epsilon)\right]\\
&\leq 2|N_a(t)| + \frac{1}{\epsilon}|N_a(t)|\\
&\quad -\left(\frac{(1+6\epsilon)(1-\beta)}{\epsilon}|N_a(t)| - \frac{(1+6\epsilon)(1-\beta)}{\epsilon\beta}|N_o(t)|\right)\\
&\quad \left[x_{p,i}(t) \leq \frac{(1+6\epsilon)Q(|N_a(t)|)}{\beta|N_a(t)|} \quad \text{for all } J_{p,i} \in N'_a(t)\right]\\
&\leq \frac{2}{\epsilon^2}|N_o(t)| \leq \frac{2}{\epsilon^2}\frac{\mathrm{d}}{\mathrm{d}t}G^*(t)
\end{aligned}$$

The second to last inequality follows from $\beta = \epsilon \leq \frac{1}{6}$. The last inequality follows since the optimal solution's flow time increases by $|N_o(t)|$ at time $t$. □

Combing Lemmas 3.2 and 3.3 along with setting $\beta$ to be $\epsilon$ gives the running condition, namely $\frac{\mathrm{d}}{\mathrm{d}t}G(t) + \frac{\mathrm{d}}{\mathrm{d}t}\Phi(t) \leq \frac{2}{\epsilon^2}\frac{\mathrm{d}}{\mathrm{d}t}G^*(t)$. Thus, we have shown all the properties of $\Phi$, which gives the following theorem.

**Theorem 3.4** *The algorithm with $\beta = \epsilon \leq \frac{1}{6}$ is $(1+6\epsilon)$-speed $\frac{2}{\epsilon^2}$-competitive for flow plus energy in fractional broadcast scheduling.*

By using the reduction from integral to fractional broadcast scheduling in Corollary 2.5 and scaling $\epsilon$, we have proven Theorem 1.1.

## 4 Reduction of integral to fractional broadcast scheduling

In this section we prove Theorem 2.4. Consider any sequence of requests and let $\mathcal{S}$ denote a valid fractional broadcast schedule. We now define an algorithm to construct an integral schedule $\mathcal{S}'$. In the integral schedule we will use $(1+\epsilon)$ resource augmentation over the schedule $\mathcal{S}$ where $0 < \epsilon \leq 1$. Let $P(s)$ be the power that $\mathcal{S}$ uses if it runs at speed $s$ for a time step. Note that since $\mathcal{S}'$ has $(1+\epsilon)$ resource augmentation over $\mathcal{S}$ it is the case that the power consumed by $\mathcal{S}'$ when running at speed $(1+\epsilon)s$ is $P(s)$. Recall that in an integral schedule, a request $J_{p,i}$ must receive $\sigma_p$ unit sized pieces of page $p$ in *sequential* order. We will assume that the fractional schedule works on at most one page during a unit time slot and at exactly one speed. Further, requests are only completed at the end of a unit time slot. We can assume this without loss of generality because we can set a unit time slot to be arbitrarily small, since we are assuming preemption and a continuous model. Let $f_{p,i}$ denote the time request $J_{p,i}$ is fractionally satisfied in $\mathcal{S}$. Let $f^I_{p,i}$ denote the time that $J_{p,i}$ is integrally satisfied in $\mathcal{S}'$. Our algorithm and analysis build on the reduction from fractional to integral broadcast scheduling given in Bansal et al. (2010) where broadcasts are always sent at a fixed speed. Since the processor speeds can vary in our setting, in our analysis we will have to be careful about how speed is distributed, and how power is accounted for. Accounting for the speed will make our algorithm and analysis more involved than that in Bansal et al. (2010).

Our algorithm will keep track of a queue $\mathcal{Q}$ of tuples. A tuple will be of the form $\langle p, w, b, k, L\rangle$. Here $p$ corresponds to a page. The value of $k \in \mathbb{R}$ will correspond to the part of page $p$ that will be broadcasted. Here $k$ can be any value between 1 and $\sigma_p$. The variable $w \in \mathbb{R}$ will be called the *width* and $b \in \mathbb{R}$ will be the *start time*. Each tuple $\tau = \langle p, w, b, k\rangle$ will correspond to some request $J_{p,i}$, and the width will be $w = f_{p,i} - a_{p,i}$. The width of a tuple and the request it corresponds to can be updated over time. Finally, $L$ will be a list of time slots corresponding to a subset of unit times the fractional schedule broadcasted page $p$. Initially $L$ will be set to $\emptyset$. When the algorithm broadcasts $p$ because of this tuple, it will choose a speed to broadcast $p$ at based on some time slot where the fractional schedule broadcasted $p$. To ensure that the schedule $\mathcal{S}'$ does not use too much energy, we need to ensure that we do not broadcast using a large speed too many times when the fractional schedule does not use this fast speed often. To ensure this, the list $L$ will keep track of all unit time slots that this tuple has previously chosen to base its speed on in the fractional schedule. When choosing a time slot, no slot already in $L$ will be chosen. We note that two different tuples could possibly choose the same unit time slot, but by the way we will define which time slots a tuple can be used, we will be able to ensure no slot is chosen more than twice. This way, we will be able to bound the energy used by $\mathcal{S}'$ by twice that used by $\mathcal{S}$.

Our algorithm will perform a broadcast corresponding to a single tuple each $1/(1+\epsilon)$ time steps. The algorithm will always choose the tuple $\tau = \langle p, w, b, k\rangle$ in $\mathcal{Q}$ such that $w$ is minimized. Then it will broadcast page $p$ starting from the point $k$. It only remains to determine the speed that $\mathcal{S}'$ uses. Say that the current time is $t$. Let $t^*$ be the latest possible time such that exactly a total volume of $\sigma_p$ units of page $p$ is broadcasted in $\mathcal{S}$ during $[t^*, b]$. Let $T(b,t)$ denote the set of unit times that $\mathcal{S}$ broadcasts $p$ during $[t^*, t]$. Let $s$ be the fastest speed used by $\mathcal{S}'$ to broadcast page $p$ during a unit time slot in $T(b,t) \setminus L$. The speed $\mathcal{S}'$ uses at time $t$ is $(1+\epsilon)s$. Note that since $\mathcal{S}'$ has $1+\epsilon$ resource augmentation over $\mathcal{S}$, the total volume of page $p$ broadcasted is $s$, and the power used is $P(s)$. See algorithm $\mathcal{S}'(t)$ for a formal definition.

It can be observed that the algorithm $\mathcal{S}'$ is online if $\mathcal{S}$ is online. The analysis will proceed as follows. First, we show the algorithm can always schedule a tuple using a valid speed at each time. That is, $T(b,t) \setminus L$ is never empty. Then we will show that for any unique time slot in $\mathcal{S}$ at most two broadcasts in $\mathcal{S}'$ will use the speed corresponding to this time slot. This will allow us to bound the energy cost of $\mathcal{S}'$ by that of $\mathcal{S}$. Then we will focus on bounding the flow time of each request $J_{p,i}$ in $\mathcal{S}'$ by the flow time of $J_{p,i}$ in $\mathcal{S}$.

**Lemma 4.1** *At any time $t$ where the tuple $\tau = \langle p, w, b, k, L\rangle$ is scheduled by $\mathcal{S}'$ it is the case that $T(b,t) \setminus L \neq \emptyset$*

*Proof* Consider any time $t$ and any tuple $\tau = \langle p, w, b, k, L\rangle$ scheduled at time $t$. We know that during the set of time slots in $T(b,t)$ the schedule $\mathcal{S}$ broadcasts a total volume of at least $\sigma_p$ units of page $p$ by definition of $T(b,t)$. We also know that any tuple with start time $b$ or less will be removed after $\sigma_p$ units of page $p$ are broadcasted by $\mathcal{S}'$ after time $b$. This implies that there will always be some available speed for $\mathcal{S}'$ to use when broadcasting a tuple with start time $b$. □

The previous lemma establishes that there is always a valid speed for the schedule $\mathcal{S}'$ to use at any point in time. Now our goal is to bound the energy used by $\mathcal{S}'$ by the energy used by $\mathcal{S}$. For page $p$, let $S_p$ denote the set of times that $\mathcal{S}'$ schedules page $p$ starting from the first piece of page $p$. We assume that these times are indexed as $\rho_1 < \rho_2, < \cdots < \rho_{|S_p|}$. We begin by showing that a total volume of at least $\sigma_p$ of page $p$ is broadcasted between any two times in $S_p$ in the schedule $\mathcal{S}$.

**Lemma 4.2** *For any two times $\rho_i, \rho_j \in S_p$ it is the case that the total volume of page $p$ broadcasted by $\mathcal{S}$ during $[\rho_i, \rho_j)$ is at least $\sigma_p$ for $i < j$.*

**Algorithm**: $\mathcal{S}'(t)$

All requests arrive unmarked
Simulate the schedule $\mathcal{S}(t)$
**for** Any unmarked request $J_{p,i}$ completed by $\mathcal{S}$ at time $t$ **do**
  **if** There is a tuple $\tau = \langle p, w, b, k, L\rangle \in \mathcal{Q}$ for page $p$ where $b \geq a_{p,i}$ **then**
    Update the width of $\tau$ to be $w = \min\{w, f_{p,i} - a_{p,i}\}$
  **else**
    Insert the tuple $\langle p, (f_{p,i} - a_{p,i}), \infty, 1, \emptyset\rangle$ into $\mathcal{Q}$
  **end if**
**end for**
**if** $t$ is a multiple of $1/(1+\epsilon)$ **then**
  Dequeue the tuple $\tau = \langle p, w, b, k, L\rangle$ with minimum width, breaking ties in favor the tuple with the largest $b$ value
  **if** $b = \infty$ **then**
    Update $b = t$
  **end if**
  Let $s$ be the fastest speed used by $\mathcal{S}$ for a unit time slot in $T(b, t) \setminus L$ and $t'$ be the beginning of the corresponding unit time slot.
  Broadcast $p$ from time $t$ to time $t + 1/(1+\epsilon)$ using speed $(1+\epsilon)s$. The pieces of page $p$ broadcasted are those between $[k, k+s)$
  Update $k = k + s$
  Update $L = L \cup \{t'\}$
**end if**
**if** $k + s = \sigma_p$ **then**
  Mark all requests for page $p$ that arrived before time $b$
  Dequeue all tuples $\tau' = \langle p, w', b', k', L'\rangle$ for page $p$ where $b' \leq b$
**end if**

*Proof* Consider any two times $\rho_i, \rho_j \in S_p$ where $i < j$. Let $\tau = \langle p, w, b, k, L\rangle$ be the tuple that caused $\mathcal{S}'$ to schedule page $p$ at time $\rho_j$, and let $J_{p,i}$ be the request that corresponds to this tuple at time $\rho_j$. By definition of the algorithm $\mathcal{S}'$, it is the case that $a_{p,i} > \rho_i$. This is because a tuple corresponding to $J_{p,i}$ would not start broadcasting page $p$ from the beginning at time $\rho_j$ if $a_{p,i} \leq \rho_i$ by definition of $\mathcal{S}'$.

We also know that no request can correspond to a tuple $\mathcal{S}'$ schedules until it is completed in $\mathcal{S}$. Thus, a volume of $\sigma_p$ units of page $p$ must be broadcasted by $\mathcal{S}$ during $[a_{p,i}, \rho_j)$ which implies that $\sigma_p$ units of page $p$ are broadcasted by $\mathcal{S}$ during $[\rho_i, \rho_j)$ since $a_{p,i} > \rho_i$. □

**Lemma 4.3** *For any two times $\rho_i, \rho_{i+1} \in S_p$ for some page $p$, the only tuples scheduled for page $p$ during $[\rho_i, \rho_{i+1})$ have start time $\rho_i$*

*Proof* To prove the lemma, we show that for every tuple $\tau = \langle p, w, b, k, L\rangle$ scheduled by $\mathcal{S}'$ at a time $t$ after time $s_j$ it is the case that $b \geq \rho_j$ for any $\rho_j \in S_p$. For the sake of contradiction say that there is a tuple $\tau = \langle p, w, b, k, L\rangle$ scheduled by $\mathcal{S}'$ at a time $t$ after $\rho_j$ where $b < \rho_j$. Let $\tau' = \langle p, w', \rho_j, k', L'\rangle$ be the tuple scheduled by $\mathcal{S}'$ for page $p$ at time $\rho_j$. Knowing that $b < \rho_j$ at time $\rho_j$ there must have been a tuple in $\mathcal{Q}$ for page $p$ with start time $b$. Let $\tau'' = \langle p, w'', b, k'', L''\rangle$ be this tuple. Since $\tau'$ was scheduled at time $\rho_j$ it must be the case that $w' \leq w''$. The only way a tuple $\tau$ could be scheduled at time $t$ is if $w < w'$. However, knowing that $b < \rho_j$ any time the width of a tuple with start time $b$ for page $p$ is updated, the width of the tuple with start time $\rho_j$ is updated to have the same width. Since the algorithm always breaks ties in favor of tuples with larger $b$ values it cannot be the case that $\tau$ was scheduled. □

**Lemma 4.4** *For any unit time interval beginning at $t$, at most two broadcasts made by $\mathcal{S}'$ at any time will set their speed according to the speed used by $\mathcal{S}$ during $[t, t+1]$.*

*Proof* Consider any time $t$ where $\mathcal{S}$ schedules some fixed page $p$. Let $\rho_i$ be the latest time in $S_p$ such that $\rho_i \leq t$. Note that $\rho_{i+1}$ is the earliest time in $S_p$ such that $\rho_{i+1} > t$. First we show that after time $\rho_{i+2}$ no tuple scheduled by $\mathcal{S}'$ will set its speed based on the speed $\mathcal{S}$ uses at time $t$. Indeed, by Lemma 4.2 there is a total volume of $\sigma_p$ units of page $p$ broadcasted by $\mathcal{S}$ during $[\rho_{i+1}, \rho_i]$. This implies that $T(\rho_j, t'')$ will not include any time slot in $[\rho_i, \rho_{i+1})$ by definition of $T(\rho_j, t'')$ for any $j \geq i+2$ and $t'' \geq s_j$.

The above fact and Lemma 4.3 imply that if $\mathcal{S}'$ ever sets the speed based on the speed used by $\mathcal{S}$ at time $t$ then it is the case that the tuple chosen by $\mathcal{S}'$ has a start time $b \in \{\rho_i, \rho_{i+1}\}$. Knowing that if a tuple $\tau = \langle p, w, b, k, L\rangle$ with start time $\rho_i$ or $\rho_{i+1}$ is scheduled by $\mathcal{S}'$ then $t$ will be appended to the list $L$ if $\mathcal{S}'$ sets the speed to the speed used by $\mathcal{S}$ at time $[t, t+1]$, it must be the case that at most two broadcasts made by $\mathcal{S}'$ at any time will set their speed according to the speed use by $\mathcal{S}$ during $[t, t+1]$ □

The previous lemma immediately gives rise to the following corollary. This is because we can map the power used by $\mathcal{S}'$ for any tuple scheduled to a time slot where $\mathcal{S}$ used this speed. Further, no slot in $\mathcal{S}$ will be mapped to more than twice.

**Corollary 4.5** *The total energy used by $\mathcal{S}'$ is at most twice that used by $\mathcal{S}$.*

Now that we have bounded the energy consumption used by $\mathcal{S}'$, we will now focus on bounding the total flow time of this schedule. To do this, we will show that every request has at most a constant factor larger flow time in $\mathcal{S}'$ as compared to $\mathcal{S}$. That is, we show the second part of Theorem 2.4. Recall that $f_{p,i} - a_{p,i}$ is the flow time of request $J_{p,i}$ in the schedule $\mathcal{S}$ and $f^I_{p,i} - a_{p,i}$ is the flow time of $J_{p,i}$ in $\mathcal{S}'$. We will show that $f^I_{p,i} - a_{p,i} \leq \frac{5}{\epsilon}(f_{p,i} - a_{p,i})$ for all requests. For the sake of contradiction say that this is not the case and let $t_2$ be the first time there exists an unsatisfied request $J_{q,x}$ in $\mathcal{S}'$ such that $t - a_{q,x} \leq \frac{5}{\epsilon}(f_{q,x} - a_{q,x})$. Let $w^* = (f_{q,x} - a_{q,x})$. Let $t_1$ be the earliest time before time $t_2$ such that during the interval $[t_1, t_2]$ every tuple scheduled by $\mathcal{S}'$ has width at most $w^*$.

Now we are going to say that for a given tuple $\tau = \langle p, w, b, k, L\rangle$ if $J_{p,i}$ is the request that corresponds to this tuple currently then $J_{p,i}$ gave this tuple its width. We begin by showing for every tuple scheduled by $\mathcal{S}'$ at some time $t$ during $[t_1, t_2]$ that the request which gave the tuple its width at this time arrived at the earliest $t_1 - w^*$.

**Lemma 4.6** *Consider any tuple $\tau = \langle p, w, b, k, L\rangle$ that was scheduled by $\mathcal{S}'$ at a time $t \in [t_1, t_2]$. If $J_{p,i}$ gave $\tau$ its width then $a_{p,i} \geq t_1 - w^*$.*

*Proof* For the sake of contradiction say that there is a fixed tuple $\tau = \langle p, w, b, k, L\rangle$ scheduled by $\mathcal{S}'$ at a time $t$ during $[t_1, t_2]$ where $J_{p,i}$ is the request that gave $\tau$ its width and $a_{p,i} < t_1 - w^*$. We know that $w \leq w^*$ by definition of $[t_1, t_2]$. This implies that during $[a_{p,i} + w^*, t_1]$, there is always a tuple in $\mathcal{Q}$ with width smaller than $w^*$ by definition of $\mathcal{S}'$. Knowing that $\mathcal{S}'$ always schedules the tuple with the smallest width at each time, this contradicts the definition of $t_1$. □

At this point, we are going to map each tuple scheduled by $\mathcal{S}'$ during $[t_1, t_2]$ to a unit time slot during $[t - w^*, t_2]$ where $\mathcal{S}$ performs a broadcast. Then we will show that this mapping implies that $\mathcal{S}$ requires more time slots in $[t - w^*, t_2]$ than actually exist. This will give us a contradiction and complete the proof. This argument is fairly technical because we do not know how to relate the speed used by $\mathcal{S}'$ to the speeds used by $\mathcal{S}$. Due to this, we cannot use a standard volume argument to draw a contradiction. Rather, we focus mapping tuples to unit time slots.

Let $S^*_p$ be the set of times in $S_p$ which occur during $[t_1, t_2)$. Let $\rho^*_1, \rho^*_2, \ldots \rho^*_{|S^*_p|}$ be the set of times in $S^*_p$ indexed from smallest to largest.

**Lemma 4.7** *For any $\rho^*_i, \rho^*_{i+1} \in S^*_p$ where $i \geq 2$, the total number of tuples scheduled by $\mathcal{S}'$ during $[\rho^*_i, \rho^*_{i+1})$ for page $p$ is at most the total number of unit time slots $\mathcal{S}$ devotes to page $p$ during $[\rho^*_{i-1}, \rho^*_i)$. Further, the total number of tuples scheduled by $\mathcal{S}'$ during $[\rho^*_{|S^*_p|}, t_2)$ for page $p$ is at most the total number of unit time slots $\mathcal{S}$ devotes to page $p$ during $[\rho^*_{|S^*_{p-1}|}, \rho^*_{|S^*_p|})$ and the total number of tuples scheduled by $\mathcal{S}'$ during $[\rho^*_1, \rho^*_2)$ for page $p$ is at most the total number of unit time slots $\mathcal{S}$ devotes to page $p$ during $[t_1 - w^*, \rho_1)$.*

*Proof* First we show that, for any $\rho^*_i, \rho^*_{i+1} \in S^*_p$ where $i \geq 2$, the total number of tuples scheduled by $\mathcal{S}'$ during $[\rho^*_i, \rho^*_{i+1})$ is at most the total number of unit time slots $\mathcal{S}$ devotes to page $p$ during $[\rho^*_{i-1}, \rho^*_i)$. Consider $T(\rho^*_i, \rho^*_i)$. By definition of $T(\rho^*_i, \rho^*_i)$ and Lemma 4.2, it is the case that $T(\rho^*_i, \rho^*_i)$ is a subset of unit times that page $p$ is scheduled by $\mathcal{S}$ during $[\rho^*_{i-1}, \rho^*_i)$. Further, by Lemma 4.3 any tuple scheduled for page $p$ by $\mathcal{S}'$ during $[\rho^*_i, \rho^*_{i+1})$ has start time $\rho^*_i$. Thus at any time $t$ during $[\rho^*_i, \rho^*_{i+1})$ a tuple $\tau = \langle p, w, \rho^*_i, k, L\rangle$ scheduled by $\mathcal{S}'$ for page $p$ can use a speed according to the time slots in $T(\rho^*_i, t) \setminus L \supseteq T(\rho^*_i, \rho^*_i) \setminus L$. Knowing that the algorithm always chooses the fastest speed possible, it must be the case that the number of tuples scheduled by $\mathcal{S}'$ is at most the total number of unit time slots $\mathcal{S}$ devotes to page $p$ during $[\rho^*_{i-1}, \rho^*_i)$.

The remaining parts of the lemma can be proved similarly, as follows. We now focus on showing that the total number of tuples scheduled by $\mathcal{S}'$ during $[\rho^*_{|S^*_p|}, t_2)$ for page $p$ is at most the total number of unit time slots $\mathcal{S}$ devotes to page $p$ during $[\rho^*_{|S^*_p|-1}, \rho^*_{|S^*_p|})$. Consider $T(\rho^*_{|S^*_p|}, \rho^*_{|S^*_p|})$. By definition of $T(\rho^*_{|S^*_p|}, \rho^*_{|S^*_p|})$ and Lemma 4.2 it is the case that $T(\rho^*_{|S^*_p|}, \rho^*_{|S^*_p|})$ is a subset of unit times that page $p$ is scheduled by $\mathcal{S}$ during $[\rho^*_{|S^*_p|-1}, \rho^*_{|S^*_p|})$. Further, by Lemma 4.3 any tuple scheduled for page $p$ by $\mathcal{S}'$ during $[\rho^*_{|S^*_p|}, t_2)$ has start time $\rho^*_{|S^*_p|}$. Thus at any time $t$ during $[\rho^*_{|S^*_p|}, t_2)$ a tuple $\tau = \langle p, w, \rho^*_{|S^*_p|}, k, L\rangle$ scheduled by $\mathcal{S}'$ for page $p$ can use a speed according to the time slots in $T(\rho^*_{|S^*_p|}, \rho^*_{|S^*_p|}) \setminus L$. Knowing that the algorithm always chooses the fastest speed possible, it must be the case that the number of tuples scheduled by $\mathcal{S}'$ is at most the total number of unit times slots $\mathcal{S}$ devotes to page $p$ during $[\rho^*_{|S^*_p|-1}, \rho^*_{|S^*_p|})$.

Finally, we show that he total number of tuples scheduled by $\mathcal{S}'$ during $[\rho^*_1, \rho^*_2)$ for page $p$ is at most the total number of unit time slots $\mathcal{S}$ devotes to page $p$ during $[t_1 - w^*, \rho^*_1)$. Consider $T(\rho^*_1, \rho^*_1)$ and the tuple $\tau = \langle p, w, \rho^*_1, k, L\rangle$ that $\mathcal{S}'$ chooses at time $\rho^*_1$. Let $J_{p,i}$ be the request associated with this tuple. By Lemma 4.6 it is the case that $a_{p,i} \geq t_1 - w^*$. Since no request can have a tuple associated with it until it is completed in $\mathcal{S}$ it must be the case that $\mathcal{S}$ broadcasts a total volume of $\sigma_p$ units of page $p$ during $[t_1 - w^*, \rho^*_1)$. This and the definition of $T(\rho^*_1, \rho^*_1)$ imply that $T(\rho^*_1, \rho^*_1)$ is a subset of unit times that page $p$ is scheduled by $\mathcal{S}$ during $[t_1 - w^*, \rho^*_1)$. Further, by Lemma 4.3 any tuple scheduled for page $p$ by $\mathcal{S}'$ during $[\rho^*_1, \rho^*_2)$ have start time $\rho^*_1$. Thus at any

time $t$ during $[\rho_1^*, \rho_2^*)$ a tuple $\tau = \langle p, w, \rho_1^*, k, L\rangle$ scheduled by $\mathcal{S}'$ for page $p$ can use a speed according to the time slots in $T(\rho_1^*, \rho_1^*) \setminus L$. Knowing that the algorithm always chooses the fastest speed possible, it must be the case that the number of tuples scheduled by $\mathcal{S}'$ is at most the total number of unit times slots $\mathcal{S}$ devotes to page $p$ during $[t_1 - w^*, \rho_1^*)$. □

**Lemma 4.8** *The total number of tuples scheduled by $\mathcal{S}'$ during $[t_1, \rho_1^*]$ for page $p$ is at most the total number of unit time slots $\mathcal{S}$ devotes to page $p$ during $[t_1 - w^*, t_1 + w^*]$.*

*Proof* Let $\tau = \langle p, w, b, k, L\rangle$ be the first tuple that $\mathcal{S}'$ schedules at a time $t^*$ during $[t_1, \rho_1^*]$ for a fixed page $p$. We can assume that $\tau$ exists because otherwise the lemma trivially holds. Let $J_{p,i}$ be the request that gave $\tau$ its width. By Lemma 4.6 it is the case that $a_{p,i} \geq t_1 - w^*$. Further, by definition of $\rho_1^*$, it must be the case that $b < t_1$. Knowing that $a_{p,i} \leq b$ we have $a_{p,i} \leq t_1$. Now we know that $w \leq w^*$ by definition of $[t_1, t_2]$. We also know that $J_{p,i}$ completes at time $a_{p,i} + w$ in $\mathcal{S}$ by definition of $w$. Thus, we know that $J_{p,i}$ completes at time $a_{p,i} + w \leq a_{p,i} + w^* \leq t_1 + w^*$. Since no request can have a tuple associated with it until it is completed in $\mathcal{S}$ and $a_{p,i} \geq t_1 - w^*$ it must be the case that $\mathcal{S}$ broadcasts a total volume of $\sigma_p$ units of page $p$ during $[t_1 - w^*, t^*)$.

This and the definition of $T(b, t^*)$ imply that $T(b, t^*)$ is a subset of unit times that page $p$ is scheduled by $\mathcal{S}$ during $[t_1 - w^*, t^*)$ a subinterval of $[t_1 - w^*, t_1 + w^*)$. Further, a volume of $\sigma_p$ units of page $p$ is scheduled during $[t_1 - w^*, t)$. By Lemma 4.3 and the definition of $\rho_1^*$ any tuple scheduled for page $p$ by $\mathcal{S}'$ during $[t_1, \rho_1^*)$ has start time $b$. Thus at any time $t$ during $[t_1, \rho_1^*)$ a tuple $\tau = \langle p, w', b, k', L'\rangle$ scheduled by $\mathcal{S}'$ for page $p$ can use a speed according to the time slots in $T(b, t^*) \setminus L$. Knowing that the algorithm always chooses the fastest speed possible, it must be the case that the number of tuples scheduled by $\mathcal{S}'$ is at most the total number of unit times slots $\mathcal{S}$ devotes to page $p$ during $[t_1 - w^*, t_1 + w^*)$. □

The previous two lemmas imply that we can map all tuples for page $p$ scheduled by $\mathcal{S}'$ during $[\rho_1^*, t_2]$ to unique time slots page $p$ is scheduled in $\mathcal{S}$ during $[t_1 - w^*, t_2]$. Further, we can map all tuples for page $p$ scheduled by $\mathcal{S}'$ during $[t_1, \rho_1]$ to unique time slots page $p$ is scheduled in $\mathcal{S}$ during $[t_1 - w^*, t_1 + w^*]$. This implies that for every tuple scheduled by $\mathcal{S}$ during $[t_1, t_2]$ we can map it to a time slot during $[t_1 - w^*, t_2]$ such that each time slot in $[t_1 + w^*, t_2]$ is mapped to at most once and each time slot in $[t_1 - w^*, t_1 + w^*]$ is mapped to at most twice. We know that $\mathcal{S}'$ is always busy during $[t_1, t_2]$ and schedules a tuple every $1/(1+\epsilon)$ time steps. Thus, a total of $(t_2 - t_1)(1+\epsilon)$ tuples are scheduled by $\mathcal{S}'$ during $[t_1, t_2]$. The mapping implies that $(t_2 - t_1)(1+\epsilon) \leq (t_2 - t_1) + 4w^*$. However, $t_2 - t_1 \geq \frac{5}{\epsilon} w^*$ by assumption. This is a contradiction and we have proved Theorem 2.4.

## 5 Conclusion

In this paper we initiated the study of energy in the broadcast scheduling model. We showed a scalable algorithm for average flow time plus energy. It is important to note that the algorithm BLAPS explicitly depends on the speed $\epsilon$. That is, $\beta$ depends on the parameter $\epsilon$. Recently, many algorithms developed for scheduling have this dependency (Chekuri et al. 2009b; Edmonds and Pruhs 2012; Gupta et al. 2010). It would be interesting to find an algorithm which does not depend on $\epsilon$ or show no such algorithm exists. It would also be of interest to consider other objective functions that include energy minimization for the broadcast setting.

**Acknowledgments** This work was partially done while the author was at the University of Illinois. Partially supported by NSF Grants CNS-0721899 and CCF-0728782

## References

Acharya, S., Franklin, M., & Zdonik, S. (Dec 1995). Dissemination-based data delivery using broadcast disks. *Personal Communications IEEE*, *2*(6), 50–60. see also IEEE Wireless Communications.

Aksoy, D., & Franklin, M. J. (1999). R×W: A scheduling approach for large-scale on-demand data broadcast. *IEEE/ACM Transaction on Networking*, *7*(6), 846–860.

Albers, S., & Fujiwara, H. (2007). Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, *3*(4), 49.

Bansal, N., Chan, H. L., & Pruhs, K. (2009). Speed scaling with an arbitrary power function. *In Proceedings of the twentieth annual ACM-SIAM symposium on discrete algorithms* (pp. 693-701).

Bansal, N., Charikar, M., Khanna, S., & Naor, J. S. (2005). Approximating the average response time in broadcast scheduling. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 215–221).

Bansal, N., Coppersmith, D., & Sviridenko, M. (2008). Improved approximation algorithms for broadcast scheduling. *SIAM Journal on Computing*, *38*(3), 1157–1174.

Bansal, N., Krishnaswamy, R., & Nagarajan, V. (2010). Better scalable algorithms for broadcast scheduling. In *International Colloquium Automata, Languages and Programming*.

Bansal, N., Pruhs, K., & Stein, C. (2009). Speed scaling for weighted flow time. *SIAM Journal on Computing*, *39*(4), 1294–1308.

Bar-Noy, A., Bhatia, R., Naor, J. S., & Schieber, B. (2002). Minimizing service and operation costs of periodic scheduling. *Mathematics of Operations Research*, *27*(3), 518–544.

Bartal, Y., & Muthukrishnan, S. (2000). Minimizing maximum response time in scheduling broadcasts. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 558–559).

Chan, H. L., Edmonds, J., Lam, T. W., Lee, L. K., Marchetti-Spaccamela, A., & Pruhs, K. (2009). Nonclairvoyant speed scaling for flow and energy. In *International Symposium on Theoretical Aspects of Computer Science STACS* (pp. 255–264).

Chang, J., Erlebach, T., Gailis, R., & Khuller, S. (2008). Broadcast scheduling: Algorithms and complexity. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 473–482). Society for Industrial and Applied Mathematics.

Chekuri, C., Im, S., & Moseley, B. (2009a). Longest wait first for broadcast scheduling. In *Proceedings of the 7th Workshop on Approximation and Online Algorithms*.

Chekuri, C., Im, S., & Moseley, B. (2009b). Minimizing maximum response time and delay factor in broadcast scheduling. In *ESA '09: Seventeenth Annual European Symposium on Algorithm* (pp. 444–455).

Chen, B., Jamieson, K., Balakrishnan, H., & Morris, R. (2002). Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networks, 8*(5), 481–494.

Deb, R. K. (1984). Optimal control of bulk queues with compound poisson arrivals and batch service. *Opsearch*, *21*, 227–245.

Deb, R. K., & Serfozo, R. F. (1973). Optimal control of batch service queues. *Advances in Applied Probability*, *5*, 340–361.

Edmonds, J., Im, S., & Moseley, B. (2010). Online scalable scheduling for the $\ell_k$-norms of flow time without conservation of work. Manuscript.

Edmonds, J., & Pruhs, K. (2003). Multicast pull scheduling: When fairness is fine. *Algorithmica*, *36*(3), 315–330.

Edmonds, J., & Pruhs, K. (2005). A maiden analysis of longest wait first. *ACM Transactions on Algorithms*, *1*(1), 14–32.

Edmonds, J., & Pruhs, K. (2012). Scalably scheduling processes with arbitrary speedup curves. *ACM Transactions on Algorithms*, *8*(3), 28.

Erlebach, T., & Hall, A. (2004). NP-hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow. *Journal of Scheduling*, *7*(3), 223–241.

Gandhi, R., Khuller, S., Parthasarathy, S., & Srinivasan, A. (2006). Dependent rounding and its applications to approximation algorithms. *Journal of the ACM*, *53*(3), 324–360.

Gupta, A., Im, S., Krishnaswamy, R., Moseley, B., & Pruhs, K. (2010). Scheduling jobs with varying parallelizability to reduce variance. In *Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures*.

Gupta, A., Krishnaswamy, R., & Pruhs, K. (2010). Scalably scheduling power-heterogeneous processors. In *International Colloquium Automata, Languages and Programming*.

Gupta, A., Krishnaswamy, R., & Pruhs, K. (2012). Online primal-dual for non-linear optimization with applications to speed scaling. In *WAOA '12: Proceedings of 10th Workshop on Approximation and Online Algorithms*.

Hall, A., & Täubig, H. (2003). Comparing push-and pull-based broadcastingo or: Would Microsoft watches profit from a transmitter?. In *Proceedings of the 2nd International Workshop on Experimental and Efficient Algorithms (WEA 03)* (pp. 148-164).

Im, S., & Moseley, B. (2012). An online scalable algorithm for average flow time in broadcast scheduling. *ACM Transactions on Algorithms*, *8*(4), 39.

Im, S., Moseley, B., & Pruhs, K. (June 2011). A tutorial on amortized local competitiveness in online scheduling. *SIGACT News*, *42*, 83–97.

Irani, S., Shukla, S., & Gupta, R. (2007). Algorithms for power savings. *ACM Transactions on Algorithms*, *3*(4), 41.

Kalyanasundaram, B., & Pruhs, K. (2000). Speed is as powerful as clairvoyance. *Journal of the ACM*, *47*(4), 617–643.

Kalyanasundaram, B., Pruhs, K., & Velauthapillai, M. (2000). Scheduling broadcasts in wireless networks. *Journal of Scheduling*, *4*(6), 339–354.

Lam, T. W., Lee, L. K., To, I. K., & Wong, P. W. (2008). Speed scaling functions for flow time scheduling based on active job count. In *Proceedings of the European Symposium on Algorithms* (pp. 647–659).

Pruhs, K., Uthaisombut, P., & Woeginger, G. (2008). Getting the best response for your erg. *ACM Transactions on Algorithms*, *4*(3), 38.

Weiss, J. (1979). Optimal control of batch service queues with nonlinear waiting costs. *Modeling and Simulation*, *10*, 305–309.

Weiss, J., & Pliska, S. (1982). Optimal policies for batch service queueing systems. *Opsearch*, *19*(1), 12–22.

Wong, J. W. (1988). Broadcast delivery. In *Proceedings of the IEEE* (pp 1566-1577), 76(12).

Xu, Y., Heidemann, J., & Estrin, D. (2000). Adaptive energy-conserving routing for multihop ad hoc networks. In Technical report, research report 527, USC/Information Sciences Institute.