Minimizing Maximum Response Time and Delay Factor in Broadcast Scheduling

Chandra Chekuri* Sungjin Im† Benjamin Moseley‡

June 11, 2009

Abstract

We consider *online* algorithms for pull-based broadcast scheduling. In this setting there are n pages of information at a server and requests for pages arrive online. When the server serves (broadcasts) a page p, all outstanding requests for that page are satisfied. We study two related metrics, namely maximum response time (waiting time) and maximum delay-factor and their *weighted* versions. We obtain the following results in the worst-case online competitive model.

- We show that FIFO (first-in first-out) is 2-competitive even when the page sizes are different. Previously this was known only for unit-sized pages [10] via a delicate argument. Our proof differs from [10] and is perhaps more intuitive.
- We give an online algorithm for maximum delay-factor that is $O(1/\epsilon^2)$-competitive with $(1+\epsilon)$-speed for unit-sized pages and with $(2+\epsilon)$-speed for different sized pages. This improves on the algorithm in [12] which required $(2+\epsilon)$-speed and $(4+\epsilon)$-speed respectively. In addition we show that the algorithm and analysis can be extended to obtain the same results for maximum *weighted* response time and delay factor.
- We show that a natural greedy algorithm modeled after LWF (Longest-Wait-First) is not $O(1)$-competitive for maximum delay factor with any constant speed even in the setting of standard scheduling with unit-sized jobs. This complements our upper bound and demonstrates the importance of the tradeoff made in our algorithm.

*Department of Computer Science, University of Illinois, 201 N. Goodwin Ave., Urbana, IL 61801. chekuri@cs.uiuc.edu. Partially supported by NSF grants CCF-0728782 and CNS-0721899.

†Department of Computer Science, University of Illinois, 201 N. Goodwin Ave., Urbana, IL 61801. im3@uiuc.edu

‡Department of Computer Science, University of Illinois, 201 N. Goodwin Ave., Urbana, IL 61801. bmosele2@uiuc.edu. Partially supported by NSF grant CNS-0721899.

1 Introduction

We consider *online* algorithms in pull-based broadcasting. In this model there are n pages (representing some form of useful information) available at a server and clients request a page that they are interested in. When the server transmits a page p, all outstanding requests for that page p are satisfied since it is assumed that all clients can simultaneously receive the information. It is in this respect that broadcast scheduling differs crucially from standard scheduling where each jobs needs its own service from the server. We distinguish two cases: all the pages are of same size (unit-size without loss of generality) and when the pages can be of different size. Broadcast scheduling is motivated by several applications in wireless and LAN based systems [1, 2, 26]. It has seen a substantial interest in the algorithmic scheduling literature starting with the work of Bartal and Muthukrishanan [5]; see [21]. In addition to the applications, broadcast scheduling has sustained interest due to the significant technical challenges that basic problems in this setting have posed for algorithm design and analysis. To distinguish broadcast scheduling from "standard" job scheduling, we refer to the latter as unicast scheduling — we use requests in the context of broadcast and jobs in the context of unicast scheduling.

In this paper, we focus on scheduling to minimize two related objectives: the maximum response time and the maximum delay factor. We also consider their *weighted* versions. Interestingly, the maximum response time metric was studied in the (short) paper of Bartal and Muthukrishnan [5] where they claimed that the online algorithm FIFO (for First In First Out) is 2-competitive for broadcast scheduling, and moreover that no deterministic online algorithm is $(2-\epsilon)$-competitive. (It is easy to see that FIFO is optimal in unicast scheduling). Despite the claim, no proof was published. It is only recently, almost a decade later, that Chang et al. [10] gave formal proofs for these claims for unit-sizes pages. This simple problem illustrates the difficulty of broadcast scheduling: the ability to satisfy multiple requests for a page p with a single transmission makes it difficult to relate the total "work" that the online algorithm and the offline adversary do. The upper bound proof for FIFO in [10] is short but delicate. In fact, [5] claimed 2-competitiveness for FIFO even when pages have different sizes. As noted in previous work [5, 14, 25], when pages have different sizes, one needs to carefully define how a request for a page p gets satisfied if it arrives midway during the transmission of the page. In this paper we consider the sequential model [14], the most restrictive one, in which the server broadcasts each page sequentially and a client receives the page sequentially without buffering; see [25] on the relationship between different models. The claim in [5] regarding FIFO for different pages is in a less restrictive model in which clients can buffer and take advantage of partial transmissions and the server is allowed to preempt. The FIFO analysis in [10] for unit-sized pages does not appear to generalize for different page sizes. Our first contribution in this paper is the following.

Theorem 1.1. *FIFO is 2-competitive for minimizing maximum response time in broadcast scheduling even with different page sizes.*

Note that FIFO, whenever the server is free, picks the page p with the earliest request and *non-preemptively* broadcasts it. Our bound matches the lower bound shown even for unit-sized pages, thus closing one aspect of the problem. Our proof differs from that of Chang et al.; it does not explicitly use the unit-size assumption and this is what enables the generalization to different page sizes. The analysis is inspired by our previous work on maximum delay factor [12] which we discuss next.

Maximum (Weighted) Delay Factor and Weighted Response Time: The delay factor of a schedule is a metric recently introduced in [10] (and implicitly in [7]) when requests have deadlines. Delay factor captures how much a request is delayed compared to its deadline. More formally, let $J_{p,i}$ denote the i'th request of page p. Each request $J_{p,i}$ arrives at $a_{p,i}$ and has a deadline $d_{p,i}$. The finish time $f_{p,i}$ of a request $J_{p,i}$ is defined to be the earliest time after $a_{p,i}$ when the page p is sequentially transmitted by the scheduler starting from the beginning of the page. Note that multiple requests for the same page can have the same finish time. Formally, the delay factor of the job $J_{p,i}$ is defined as $\max\{1, \frac{f_{p,i}-a_{p,i}}{d_{p,i}-a_{p,i}}\}$; we refer to the quantity $S_{p,i} = d_{p,i} - a_{p,i}$

as the *slack* of $J_{p,i}$. For a more detailed motivation of delay factor, see [12]. Note that for unit-sized pages, delay factor generalizes response time since one could set $d_{p,i} = a_{p,i} + 1$ for each request $J_{p,i}$ in which case its delay factor equals its response time. In this paper we are interested in online algorithms that minimize the maximum delay factor, in other words the objective function is $\min \max_{p,i}\{1, \frac{f_{p,i}-a_{p,i}}{d_{p,i}-a_{p,i}}\}$. We also consider a related metric, namely *weighted* response time. Let $w_{p,i}$ be a non-negative weight associated with $J_{p,i}$; the weighted response time is then $w_{p,i}(f_{p,i} - a_{p,i})$ and the goal is to minimize the maximum weighted response time. Delay factor and weighted response time have syntactic similarity if we ignore the 1 term in the definition of delay factor — one can think of the weight as the inverse of the slack. Although the metrics are some what similar we note that there is no direct way to reduce one to the other. On the other hand, we observe that upper bounds for one appear to translate to the other. We also consider the problem of minimizing the maximum weighted delay factor $\min \max_{p,i} w_{p,i}\{1, \frac{f_{p,i}-a_{p,i}}{d_{p,i}-a_{p,i}}\}$.

Surprisingly, the maximum weighted response time metric appears to not have been studied formally even in classical unicast scheduling; however a special case, namely maximum *stretch* has received attention. The stretch of a job is its response time divided by its processing time; essentially the weight of a job is the inverse of its processing time. Bender et al. [6, 8], motivated by applications to web-server scheduling, studied maximum stretch and showed very strong lower bounds in the online setting. Using similar ideas, in some previous work [12], we showed strong lower bounds for minimizing maximum delay factor even for unit-time jobs. In [12], constant competitive algorithms were given for minimizing maximum delay factor in both unicast and broadcast scheduling; the algorithms are based on resource augmentation [20] wherein the algorithm is given a speed $s > 1$ server while the offline adversary is given a speed 1 server. They showed that **SSF** (shortest slack first) is $O(1/\epsilon)$-competitive with $(1+\epsilon)$-speed in unicast scheduling. **SSF** does not work well in the broadcast scheduling. A different algorithm that involves waiting, **SSF-W** (shortest slack first with waiting) was developed and analyzed in [12]; the algorithm is $O(1/\epsilon^2)$-competitive for unit-size pages with $(2+\epsilon)$-speed and with $(4+\epsilon)$-speed for different sized pages. In this paper we obtain improved results by altering the analysis of **SSF-W** in a subtle and important way. In addition we show that the algorithm and analysis can be altered in an easy fashion to obtain the same bounds for weighted response time and delay factor.

Theorem 1.2. *There is an algorithm that is $(1+\epsilon)$-speed $O(1/\epsilon^2)$-competitive for minimizing maximum delay factor in broadcast scheduling with unit-sized pages. For different sized pages there is a $(2+\epsilon)$-speed $O(1/\epsilon^2)$-competitive algorithm. The same bounds apply for minimizing maximum weighted response time and maximum weighted delay factor.*

Remark 1.3. *Minimizing maximum delay factor is NP-hard and there is no $(2-\epsilon)$-approximation unless $P = NP$ for any $\epsilon > 0$ in the* offline *setting for unit-sized pages. There is a polynomial time computable 2-speed schedule with the optimal delay factor (with 1-speed) [10]. Theorem 1.2 gives a polynomial time computable $(1+\epsilon)$-speed schedule that is $O(1/\epsilon^2)$-optimal (with 1-speed).*

We remark that the algorithm **SSF-W** makes an interesting tradeoff between two competing metrics and we explain this tradeoff in the context of weighted response time and a lower bound we prove in this paper for a simple greedy algorithm. Recall that FIFO is 2-competitive for maximum response time in broadcast scheduling and is optimal for job scheduling. What are natural ways to generalize FIFO to delay factor and weighted response time? As shown in [12], **SSF** (which is equivalent to maximum weight first for weighted response time) is $O(1/\epsilon)$-competitive with $(1+\epsilon)$-speed for job scheduling but is not competitive for broadcast scheduling — it may end up doing much more work than necessary by transmitting a page repeatedly instead of waiting and accumulating requests for a page. One natural algorithm that extends FIFO for delay factor or weighted response time is to schedule the request in the queue that has the largest current delay factor (or weighted wait time). This greedy algorithm was labeled **LF** (longest first) since it can be seen as an extension of the well-studied **LWF** (longest-wait-first) for average flow time. Since **LWF**

is known to be $O(1)$-competitive with $O(1)$-speed for average flow time, it was suggested in [11] that **LF** may be $O(1)$-speed $O(1)$-competitive for maximum delay factor. We show that this is not the case even for unicast scheduling.

Theorem 1.4. *For any constants $s, c > 1$,* **LF** *is not c-competitive with s-speed for minimizing maximum delay factor (or weighted response time) in unicast scheduling of unit-time jobs.*

Our algorithm **SSF-W** can be viewed as an interesting tradeoff between **SSF** and **LF**. **SSF** gives preference to small slack requests while the **LF** strategy helps avoid doing too much extra work in broadcast scheduling by giving preference to pages that have waited sufficiently long even if they have large slack. The algorithm **SSF-W** considers all requests whose delay factor at time t (or weighted wait time) is within a constant factor of the largest delay factor at t and amongst those requests schedules the one with the smallest slack. This algorithmic principle may be of interest in other settings and is worth exploring in the future.

Other Related Work: We have focussed on maximum response time and its variants and have already discussed closely related work. Other metrics that have received substantial attention in broadcast scheduling are minimizing average flow time and maximizing throughput of satisfied requests when requests have deadlines. We refer the reader to a comprehensive survey on online scheduling algorithms by Pruhs, Sgall and Torng [24] (see also [23]). The recent paper of Chang et al. [10] addresses, among other things, the offline complexity of several basic problems in broadcast scheduling. Average flow-time received substantial attention in both the offline and online settings [21, 17, 18, 19, 3, 4]. For average flow time, there are three $O(1)$-speed $O(1)$-competitive online algorithms. **LWF** is one of them [14, 11] and the others are BEQUI [14] and its extention [16]. Our recent work [11] has investigated L_k norms of flow-time and showed that **LWF** is $O(k)$-speed $O(k)$-competitive algorithm. Constant competitive online algorithms for maximizing throughput for unit-sized pages can be found in [22, 9, 27, 13]. A more thorough description of related work is deferred to a full version of the paper.

Organization: We prove each of the theorems mentioned above in a different section. The algorithm and analysis for weighted response time and weighted delay factor are very similar to that for delay factor and hence, in this version, we omit the analysis and only describe the algorithm.

Notation: We denote the length of page p by ℓ_p. That is, ℓ_p is the amount of time a 1-speed server takes to broadcast page p non-preemptively. We assume without loss of generality that for any request $J_{p,i}$, $S_{p,i} \geq \ell_p$. For an algorithm A we let α^A denote the maximum delay factor witnessed by A for a given sequence of requests. We let α^* denote the optimal delay factor of an offline schedule. Likewise, we let ρ^A denote the maximum response time witnessed by A and ρ^* the optimal response time of an offline schedule. For a time interval $I = [a, b]$ we define $|I| = b - a$ to be the length of interval I.

2 Minimizing the Maximum Response Time

In this section we analyze **FIFO** for minimizing maximum response time when page sizes are different. We first describe the algorithm **FIFO**. **FIFO** broadcasts pages *non-preemptively*. Consider a time t when **FIFO** finished broadcasting a page. Let $J_{p,i}$ be the request in **FIFO**'s queue with earliest arrival time breaking ties arbitrarily. **FIFO** begins broadcasting page p at time t. At any time during this broadcast, we will say that $J_{p,i}$ *forced* **FIFO** to broadcast page p at this time. When broadcasting a page p all requests for page p that arrived before the start of the broadcast are simultaneously satisfied when the broadcast completes. Any request for page p that arrive during the broadcast are not satisfied until the next full transmission of p.

We consider **FIFO** when given a 1-speed machine. Let σ be an arbitrary sequence of requests. Let OPT denote some fixed offline optimum schedule and let ρ^* denote the optimum maximum response time. We will show that $\rho^{\mathbf{FIFO}} \leq 2\rho^*$. For the sake of contradiction, assume that **FIFO** witnesses a response time

$c\rho^*$ by some job $J_{q,k}$ for some $c > 2$. Let t^* be the time $J_{q,k}$ is satisfied, that is $t^* = f_{q,k}$. Let t_1 be the smallest time less than t^* such that at any time t during the interval $[t_1, t^*]$ the request which forces **FIFO** to broadcast a page at time t has response time at least ρ^* when satisfied. Throughout the rest of this section we let $I = [t_1, t^*]$. Let $\mathcal{J}_I$ denote the requests which forced **FIFO** to broadcast during I. Notice that during the interval I, all requests in $\mathcal{J}_I$ are completely satisfied during this interval. In other words, any request in $\mathcal{J}_I$ starts being satisfied during I and is finished during I.

We say that OPT *merges* two distinct requests for a page p if they are satisfied by the same broadcast.

Lemma 2.1. OPT *cannot merge any two requests in $\mathcal{J}_I$ into a single broadcast.*

Proof. Let $J_{p,i}, J_{p,j} \in \mathcal{J}_I$ such that $i < j$. Note that that $J_{p,i}$ is satisfied before $J_{p,j}$. Let t' be the time that **FIFO** *starts* satisfying request $J_{p,i}$. By the definition of I, request $J_{p,i}$ has response time at least ρ^*. The request $J_{p,j}$ must arrive after time t', that is $a_{p,j} > t'$, otherwise request $J_{p,j}$ is satisfied by the same broadcast of page p that satisfied $J_{p,i}$. Therefore, it follows that if OPT merges $J_{p,i}$ and $J_{p,j}$ then the finish time of $J_{p,i}$ in OPT is strictly greater than its finish time in **FIFO** which is already at least ρ^*; this is a contradiction to the definition of ρ^*. □

Lemma 2.2. *All requests in $\mathcal{J}_I$ arrived no earlier than time $t_1 - \rho^*$.*

Proof. For the sake of contradiction, suppose some request $J_{p,i} \in \mathcal{J}_I$ arrived at time $a_{p,i} < t_1 - \rho^*$. During the interval $[a_{p,i} + \rho^*, t_1]$ the request $J_{p,i}$ must have wait time at least ρ^*. However, then any request which forces **FIFO** to broadcast during $[a_{p,i} + \rho^*, t_1]$ must have response time at least ρ^*, contradicting the definition of t_1. □

We are now ready to prove Theorem 1.1, stating that **FIFO** is 2-competitive.

Proof. Recall that all requests in $\mathcal{J}_I$ are completely satisfied during I. Thus we have that the total size of requests in $\mathcal{J}_I$ is $|I|$. By definition $J_{q,k}$ witnesses a response time greater than $2\rho^*$ and therefore $t^* - a_{q,k} > 2\rho^*$. Since $J_{q,k} \in \mathcal{J}_I$ is the last request done by **FIFO** during I, all requests in $\mathcal{J}_I$ must arrive no later than $a_{q,k}$. Therefore, these requests must be finished by time $a_{q,k} + \rho^*$ by the optimal solution. From Lemma 2.2, all the requests $\mathcal{J}_I$ arrived no earlier than $t_1 - \rho^*$. Thus OPT must finish all requests in $\mathcal{J}_I$, whose total volume is $|I|$, during $I_{opt} = [t_1 - \rho^*, a_{q,k} + \rho^*]$. Thus it follows that $|I| \leq |[t_1 - \rho^*, a_{q,k} + \rho^*]|$, which simplifies to $t^* \leq a_{q,k} + 2\rho^*$. This is a contradiction to the fact that $t^* - a_{q,k} > 2\rho^*$. □

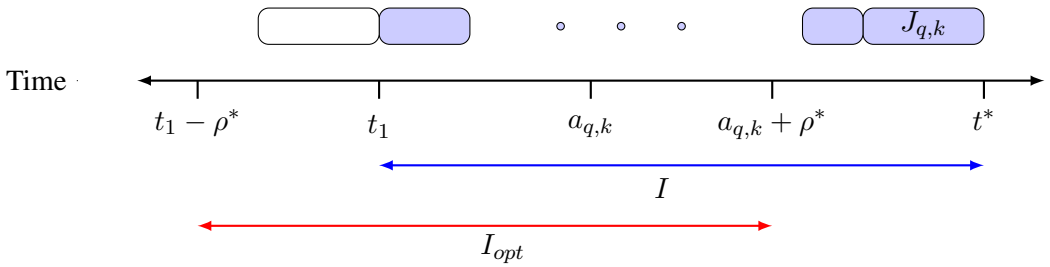

Figure 1: Broadcasts by **FIFO** satisfying requests in $\mathcal{J}_I$ are shown in blue. Note that $a_{q,k}$ and $a_{q,k} + \rho^*$ are not necessarily contained in I.

We now discuss the differences between our proof of **FIFO** for varying sized pages and the proof given by Chang et al. in [10] showing that **FIFO** is 2-competitive for unit sized pages. In [10] it is shown that at anytime t, $F(t)$, the set of *unique* pages in **FIFO**'s queue satisfies the following property: $|F(t) \setminus O(t)| \leq |O(t)|$ where $O(t)$ is the set of unique pages in OPT's queue. This easily implies the desired bound. To establish this, they use a slot model in which unit-sized pages arrive only during integer times which allows

one to define unique pages. This may appear to be a technicality, however when considering different sized pages, it is not so clear how one even defines unique pages since this number varies during the transmission of p as requests accumulate. Our approach avoids this issue in a clean manner by not assuming a slot model or unit-sized pages.

3 Minimizing Maximum Delay Factor and Weighted Response time

In this section we consider the problem of minimizing maximum delay factor and prove Theorem 1.2.

3.1 Unit Sized Pages

In this section we consider the problem of minimizing the maximum delay factor when all pages are of unit size. In this setting we assume preemption is not allowed. In the standard unicast scheduling setting where each broadcast satisfies exactly one request, it is known that the algorithm which always schedules the request with smallest slack at any time is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive [12]. However, in the broadcast setting this algorithm, along with other simple greedy algorithms, do not provide constant competitive ratios even with extra speed. The reason for this is that the adversary can force these algorithm to repeatedly broadcast the same page even though the adversary can satisfy each of these requests in a singe broadcast.

Due to this, we consider a more sophisticated algorithm called **SSF-W** (Shortest-Slack-First with Waiting). This algorithm was developed and analyzed in [12]. In this paper we alter the algorithm in a slight but practically important way. The main contribution is, however, a new analysis that is at a high-level similar in outline to the one in [12] but is subtly different and leads to much improved bound on its performance. **SSF-W** *adaptively* forces requests to wait after their arrival before they are considered for scheduling. The algorithm is parameterized by a real value $c > 1$ which is used to determine how long a request should wait. Before scheduling a page at time t, the algorithm determines the largest current delay factor of any request that is unsatisfied at time t, α_t. Amongst the unsatisfied requests that have a current delay factor at least $\frac{1}{c}\alpha_t$, the page corresponding to the request with smallest slack is broadcasted. Note that in the algorithm, each request is forced to wait to be scheduled until it has delay factor at least $\frac{1}{c}\alpha_t$. Thus **SSF-W** can be seen an adaptation of the algorithm which schedules the request with smallest slack in broadcasting setting with explicit waiting. Waiting is used to potentially satisfy multiple requests with similar arrival times in a single broadcast. Another interpretation, that we mentioned earlier, is that **SSF-W** is a balance between **LF** and **SSF**.

Algorithm: **SSF-W**

- Let α_t be the maximum delay factor of any request in **SSF-W**'s queue at time t.
- At time t, let $Q(t) = \{J_{p,i} \mid J_{p,i}$ has not been satisfied and $\frac{t-a_{p,i}}{S_{p,i}} \geq \frac{1}{c}\alpha_t\}$.
- If the machine is free at t, schedule the request in $Q(t)$ with the smallest slack *non-preemptively*.

First we note the difference between **SSF-W** above and the one described in [12]. Let α'_t denote the maximum delay factor witnessed so far by **SSF-W** at time t over all requests seen by t including satisfied and unsatisfied requests. In [12], a request $J_{p,i}$ is in $Q(t)$ if $\frac{t-a_{p,i}}{S_{p,i}} \geq \frac{1}{c}\alpha'_t$. Note that α'_t is monotonically increases with t while α_t can increase and decrease with t and is never more than α'_t. In the old algorithm it is possible that $Q(t)$ is empty and no request is scheduled at t even though there are outstanding requests! Our new version of **SSF-W** can be seen as more practical since there will always be requests in $Q(t)$ if there are outstanding requests and moreover it adapts and may reduce α_t as the request sequence changes

with time. It is important to note that our analysis and the analysis given in [12] hold for both definitions of **SSF-W** with some adjustments.

We analyze **SSF-W** when it is given a $(1+\epsilon)$-speed machine. Let $c > 1 + \frac{2}{\epsilon}$ be the constant which parameterizes **SSF-W**. Let σ be an arbitrary sequence of requests. We let OPT denote some fixed offline optimum schedule and let α^* and $\alpha^{\textbf{SSF-W}}$ denote the maximum delay factor achieved by OPT and **SSF-W**, respectively. We will show that $\alpha^{\textbf{SSF-W}} \leq c^2\alpha^*$. For the sake of contradiction, suppose that **SSF-W** witnesses a delay factor greater than $c^2\alpha^*$. We consider the *first* time t^* when **SSF-W** has some request in its queue with delay factor $c^2\alpha^*$. Let the request $J_{q,k}$ be a request which achieves the delay factor $c^2\alpha^*$ at time t^*. Let t_1 be the smallest time less than t^* such that at each time t during the interval $[t_1, t^*]$ if **SSF-W** is forced to broadcast by request $J_{p,i}$ at time t it is the case that $\frac{t-a_{p,i}}{S_{p,i}} \geq \alpha^*$ and $S_{p,i} \leq S_{q,k}$. Throughout this section we let $I = [t_1, t^*]$. The main difference between the analysis in [12] and the one here is in the definition of t_1. In [12], t_1 was implicitly defined to be $a_{q,k} + c(f_{q,k} - a_{q,k})$.

We let $\mathcal{J}_I$ denote the requests which forced **SSF-W** to schedule broadcasts during the interval $[t_1, t^*]$. We now show that any two request in $\mathcal{J}_I$ cannot be satisfied with a single broadcast by the optimal solution. Intuitively, the most effective way the adversary to performs better than **SSF-W** is to merge requests of the same page into a single broadcast. Here we will show this is not possible for the requests in $\mathcal{J}_I$. We defer the proof of Lemma 3.1 to the Appendix, since the proof is similar to that of Lemma 2.1

Lemma 3.1. OPT *cannot merge any two requests in $\mathcal{J}_I$ into a single broadcast.*

To fully exploit the advantage of speed augmentation, we need to ensure that the length of the interval I is sufficiently long.

Lemma 3.2. $|I| = |[t_1, t^*]| \geq (c^2 - c)S_{q,k}\alpha^*$.

Proof. The request $J_{q,k}$ has delay factor at least $c\alpha^*$ at any time during $I' = [t', t^*]$, where $t' = t^* - (c^2 - c)S_{q,k}\alpha^*$. Let $\tau \in I'$. The largest delay factor any request can have at time τ is less than $c^2\alpha^*$ by definition of t^* being the first time **SSF-W** witnesses delay factor $c^2\alpha^*$. Hence, $\alpha_\tau \leq c^2\alpha^*$. Thus, the request $J_{q,k}$ is in the queue $Q(\tau)$ because $c\alpha^* \geq \frac{1}{c}\alpha_\tau$. Moreover, this means that any request that forced **SSF-W** to broadcast during I', must have delay factor at least α^* and since $J_{q,k} \in Q(\tau)$ for any $\tau \in I'$, the requests scheduled during I' must have slack at most $S_{q,k}$. □

We now explain a high level view of how we lead to a contradiction. From Lemma 3.1, we know any two requests in $\mathcal{J}_I$ cannot be merged by OPT. Thus if we show that OPT must finish all these requests during an interval which is not long enough to include all of them, we can draw a contradiction. More precisely, we will show that all requests in $\mathcal{J}_I$ must be finished during I_{opt} by OPT, where $I_{opt} = [t_1 - 2S_{q,k}\alpha^* c, t^*]$. It is easy to see that all these requests already have delay factor α^* by time t^*, thus the optimal solution must finish them by time t^*. For the starting point, we will bound the arrival times of the requests in $\mathcal{J}_I$ in the following lemma. After that, we will draw a contradiction in Lemma 3.4.

Lemma 3.3. *Any request in $\mathcal{J}_I$ must have arrived after time $t_1 - 2S_{q,k}\alpha^* c$.*

Proof. For the sake of contradiction, suppose that some request $J_{p,i} \in \mathcal{J}_I$ arrived at time $t' < t_1 - 2S_{q,k}\alpha^* c$. Recall that $J_{p,i}$ has a slack no bigger than $S_{q,k}$ by the definition of I. Therefore at time $t_1 - S_{q,k}\alpha^* c$, $J_{p,i}$ has a delay factor of at least $c\alpha^*$. Thus any request scheduled during the interval $I' = [t_1 - S_{q,k}\alpha^* c, t_1]$ has a delay factor no less than α^*. We observe that $J_{p,i}$ is in $Q(\tau)$ for $\tau \in I'$; otherwise there must be a request with a delay factor bigger than $c^2\alpha^*$ at time τ and this is a contradiction to the assumption that t^* is the first time that **SSF-W** witnessed a delay factor of $c^2\alpha^*$. Therefore any request scheduled during I' has a slack no bigger than $S_{p,i}$. Also we know that $S_{p,i} \leq S_{q,k}$. In sum, we showed that any request done during I' had slack no bigger than $S_{q,k}$ and a delay factor no smaller than α^*, which is a contradiction to the definition of t_1. □

Now we are ready to prove the competitiveness of **SSF-W**.

Lemma 3.4. *Suppose c is a constant s.t. $c > 1 + 2/\epsilon$. If* **SSF-W** *has $(1+\epsilon)$-speed then $\alpha^{\textbf{SSF-W}} \leq c^2\alpha^*$.*

Proof. For the sake of contradiction, suppose that $\alpha^{\textbf{SSF-W}} > c^2\alpha^*$. During the interval I, the number of broadcasts which **SSF-W** transmits is $(1+\epsilon)|I|$. From Lemma 3.3, all the requests processed during I have arrived no earlier than $t_1 - 2c\alpha^* S_{q,k}$. We know that the optimal solution must process these requests before time t^* because these requests have delay factor at least α^* by t^*. By Lemma 3.1 the optimal solution must make a unique broadcast for each of these requests. Thus, the optimal solution must finish all of these requests in $2c\alpha^* S_{q,k} + |I|$ time steps. Thus, the it must hold that $(1+\epsilon)|I| \leq 2c\alpha^* S_{q,k} + |I|$. Using Lemma 3.2, this simplifies to $c \leq 1 + 2/\epsilon$, which is a contradiction to $c > 1 + 2/\epsilon$,. □

The previous lemmas prove the first part of Theorem 1.2 when $c = 1 + 3/\epsilon$. Namely that **SSF-W** is a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive algorithm for minimizing the maximum delay factor in broadcast scheduling with unit sized pages.

We now compare proof of Theorem 1.2 and the proof of Theorem 1.1 with the analysis given in [12]. The central technique used in [12] and in our analysis is to draw a contradiction by showing that the optimal solution must complete more requests than possible on some time interval I. This technique is well known in unicast scheduling. At the heart of this technique is to find the which requests to consider and bounding the length of the interval I. This is where our proof and the one given in [12] differ. Here we are more careful on how I is defined and how we find requests the optimal solution must broadcast during I. This allows us to show tighter bounds on the speed and competitive ratios while simplifying the analysis. In fact, our analysis of **FIFO** and **SSF-W** shows the importance of these definitions. Our analysis of **FIFO** shows that a tight bound on the length of I can force a contradiction without allowing extra speed-up given to the algorithm. Our analysis of **SSF-W** shows that when the length of I varies how resource augmentation can be used to force the contradiction.

3.2 Weighted Response Time and Weighed Delay Factor

Before showing that **SSF-W** is $(2+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive for minimizing the maximum delay factor with different sized pages, we show the connection of our analysis of **SSF-W** to the problem of minimizing *weighted* response time. In this setting a request $J_{p,i}$ has a weight $w_{p,i}$ instead of a slack. The goal is to minimize the maximum weighted response time $\max_{p,i} w_{p,i}(f_{p,i} - a_{p,i})$. We develop an algorithm which we call **BWF-W** for Biggest-Wait-First with Waiting. This algorithm is defined analogously to the definition of **SSF-W**. The algorithm is parameterized by a constant $c > 1$. At any time t before broadcasting a page, **BWF-W** determines the largest weighted wait time of any request which has yet to be satisfied. Let this value be ρ_t. The algorithm then chooses to broadcast a page corresponding to the request with largest weight amongst the requests whose current weighted wait time at time t is larger than $\frac{1}{c}\rho_t$.

Algorithm: **BWF-W**

- Let ρ_t be the maximum weighted wait time of any request in **BWF-W**'s queue at time t.
- At time t, let $Q(t) = \{J_{p,i} \mid J_{p,i}$ has not been satisfied and $w_{p,i}(t - a_{p,i}) \geq \frac{1}{c}\rho_t\}$.
- If the machine is free at t, schedule the request in $Q(t)$ with largest weight *non-preemptively*.

Although minimizing the maximum delay factor and minimizing the maximum weighted flow time are very similar metrics, the problems are not equivalent.

It may also be of interest to minimize the maximum *weighted* delay factor. In this setting each request has a deadline and a weight. The goal is to minimize $\max_{p,i} w_{p,i}(f_{p,i} - a_{p,i})/S_{p,i}$. For this setting we develop another algorithm which we call **SRF-W** (Smallest-Ratio-First with Waiting). The algorithm takes the parameter c. At any time t before broadcasting a page, **SRF-W** determines the largest weighted delay factor of any request which has yet to be satisfied. Let this value be α_t^w. The algorithm then chooses to broadcast a page corresponding to the request with the smallest ratio of the slack over the weight amongst the requests whose current weighted delay factor at time t is larger than $\frac{1}{c}\alpha_t^w$. The algorithm can be formally expressed as follows.

Algorithm: **SRF-W**

- Let α_t^w be the maximum weighted delay factor of any request in **SRF-W**'s queue at time t.
- At time t, let $Q(t) = \{J_{p,i} \mid J_{p,i}$ has not been satisfied and $w_{p,i}(t - a_{p,i})/S_{p,i} \geq \frac{1}{c}\alpha_t^w\}$.
- If the machine is free at t, schedule the request in $Q(t)$ with smallest slack over weight *non-preemptively*.

For the problems of minimizing the maximum weighted response time and weighted delay factor, the upper bounds shown for **SSF-W** in this paper also hold for **BWF-W** and **SRF-W**, respectively. The analysis of **BWF-W** and **SRF-W** is very similar to that of **SSF-W** and the proofs are omitted.

3.3 Varying Sized Pages

Here we extend our ideas to the case where pages can have a different sizes for the objective of minimizing the maximum delay factor. We develop a generalization of **SSF-W** for this setting which is similar to the generalization of **SSF-W** given in [12]. For each page p, we let ℓ_p denote the length of page p. Since pages have different lengths, we allow preemption. Therefore, if t_1 is the time where the broadcast of page p is started and t_2 is the time that this broadcast is completed it is the case that $\ell_p \leq t_2 - t_1$. A request for the page p is satisfied by this broadcast only if the request arrives before time t_1. A request that arrives during the interval $(t_1, t_2]$ does not start being satisfied because it must receive a sequential transmission of page p starting from the beginning. It is possible that a transmission of page p is *restarted* due to another request for page p arriving which has smaller slack. The original transmission of page p in this case is abandoned. Notice that this results in wasted work by the algorithm. It is because of this wasted work that more speed is needed to show the competitiveness of **SSF-W**.

We outline the details of modifications to **SSF-W**. As before, at any time t, the algorithm maintains a queue $Q(t)$ at each time where a request $J_{p,i}$ is in $Q(t)$ if and only if $\frac{t-a_{p,i}}{S_{p,i}} \geq \frac{1}{c}\alpha_t$. The algorithm broadcasts a request with the smallest slack in $Q(t)$. The algorithm may preempt a broadcast of p that is forced by request $J_{p,i}$ if another request $J_{p',j}$ becomes available for scheduling such that $S_{p',j} < S_{p,i}$. If the request $J_{p,i}$ ever forces **SSF-W** to broadcast again, then **SSF-W** continues to broadcast page p from where it left off before the preemption. If another request for page p forces **SSF-W** to broadcast page p before $J_{p,i}$ is satisfied, then the transmission of page p is *restarted*. A key difference between our generalization of **SSF-W** and the one from [12] is that in our new algorithm, requests can be forced out of Q even after they have been started. Hence, in our version of **SSF-W** every request in $Q(t)$ has current delay factor at least $\frac{1}{c}\alpha_t$ at time t. Our algorithm breaks ties arbitrarily. In [12], ties are broken arbitrarily, but the algorithm ensures that if a request $J_{p,k}$ is started before a request $J_{p',j}$ then $J_{p,k}$ will be finished before request $J_{p',j}$. Here, this requirement is not needed. Note that the algorithm may preempt a request $J_{p,i}$ by another request $J_{p,k}$, for the same page p if $S_{p,k} < S_{p,i}$. In this case the first broadcast of page p is abandoned. Notice that multiple broadcasts of page p can repeatedly be abandoned.

We now analyze the extended algorithm assuming that it has a $(2+\epsilon)$-speed advantage over the optimal offline algorithm. As mentioned before, the extra speed is needed to overcome the wasted work by abandoning broadcasts.

As before, let σ be an arbitrary sequence of requests. We let OPT denote some fixed offline optimum schedule and let α^* denote the optimum delay factor. Let $c > 1 + \frac{4}{\epsilon}$ be the constant that parameterizes **SSF-W**. We will show that $\alpha^{\textbf{SSF-W}} \leq c^2\alpha^*$. For the sake of contradiction, suppose that **SSF-W** witnesses a delay factor greater than $c^2\alpha^*$. We consider the *first* time t^* when **SSF-W** has some request in its queue with delay factor $c^2\alpha^*$. Let the request $J_{q,k}$ be a request which achieves the delay factor $c^2\alpha^*$ at time t^*. Let t_1 be the smallest time less than t^* such that at each time t during the interval $[t_1, t^*]$ if **SSF-W** is forced to broadcast by request $J_{p,i}$ at time t it is the case that $\frac{t-a_{p,i}}{S_{p,i}} \geq \alpha^*$ and $S_{p,i} \leq S_{q,k}$. Throughout this section we let $I = [t_1, t^*]$. Notice that some requests that force **SSF-W** to broadcast during I could have started being satisfied before t_1.

We say that a request *starts* being scheduled at time t if it is the request which forces **SSF-W** to broadcast at time t and t is the first time the request forces **SSF-W** to schedule a page. Notice that a request can only start being satisfied once and at most one request starts being scheduled at any time. We now show a lemma analogous to Lemma 3.1.

Lemma 3.5. *Consider two distinct requests $J_{x,j}$ and $J_{x,i}$ for some page x. If $J_{x,j}$ and $J_{x,i}$ both start being scheduled by* **SSF-W** *during the interval I then* OPT *cannot satisfy $J_{x,j}$ and $J_{x,i}$ by a single broadcast.*

Proof. Without loss of generality say that request $J_{x,j}$ was satisfied before request $J_{x,i}$ by **SSF-W**. Let t' be the time that **SSF-W** *starts* satisfying request $J_{x,j}$. By the definition of I, request $J_{x,j}$ must have delay factor at least α^* at this time. We also know that the request $J_{x,i}$ must arrive after time t', otherwise request $J_{x,i}$ must also be satisfied at time t'. If the optimal solution combines these requests into a single broadcast then the request $J_{x,j}$ must wait until the request $J_{x,i}$ arrives to be satisfied. However, this means that the request $J_{x,j}$ must achieve a delay factor greater than α^* by OPT, a contradiction of the definition of α^*. □

The next two lemmas have proofs similar to Lemma 3.2 and Lemma 3.3, we defer the proofs to the Appendix.

Lemma 3.6. $|I| = |[t_1, t^*]| \geq (c^2 - c)S_{q,k}\alpha^*$.

Lemma 3.7. *Any request which forced* **SSF-W** *to schedule a page during I must have arrived after time $t_1 - 2S_{q,k}\alpha^* c$.*

Using the previous lemmas we can bound the competitiveness of **SSF-W**. In the following lemma the main difference between the proof for unit sized pages and the proof for varying sized pages can be seen. The issue is that there can be some requests which start being satisfied before time t_1 which force **SSF-W** to broadcast a page during the interval I. When these requests were started, their delay factor need not be bounded by α^*. Due to this, it is possible for these requests to be merged with other requests which forced **SSF-W** to broadcast on the interval I.

Lemma 3.8. *Suppose c is a constant s.t. $c > 1 + 4/\epsilon$. If* **SSF-W** *has $(2+\epsilon)$-speed then $\alpha^{\textbf{SSF-W}} \leq c^2\alpha^*$.*

Proof. For the sake of contradiction, suppose that $\alpha^{\textbf{SSF-W}} > c^2\alpha^*$. Let $\mathcal{A}$ be the set of requests which start being satisfied before time t_1 which force **SSF-W** to broadcast at some time during I. Notice that no two requests in $\mathcal{A}$ are for the same page. Let $\mathcal{B}$ be the set of requests which start being satisfied during the interval I. Note that the sets $\mathcal{A}$ and $\mathcal{B}$ may consist of requests whose corresponding broadcast was abandoned at some point and that $\mathcal{A} \cap \mathcal{B} = \emptyset$ by definition. Let $V_{\mathcal{A}}$ and $V_{\mathcal{B}}$ denote the total sum of size of the requests in $\mathcal{A}$ and $\mathcal{B}$, respectively.

During the interval I, the volume of broadcasts which **SSF-W** transmits is $(2+\epsilon)|I|$. Notice that $V_{\mathcal{A}} + V_{\mathcal{B}} \geq (2+\epsilon)|I|$, since $\mathcal{A} \cup \mathcal{B}$ accounts for all requests which forced **SSF-W** to broadcast their pages during I. From Lemma 3.7, *all* the requests processed during I have arrived no earlier than $t_1 - 2c\alpha^* S_{q,k}$. We know that the optimal solution must process these requests before time t^* because these requests have delay factor at least α^* by this time.

By Lemma 3.5 the optimal solution must make a unique broadcast for each request in $\mathcal{B}$. We also know that no two requests in $\mathcal{A}$ can be merged because no two requests in $\mathcal{A}$ are for the same page. The optimal solution, however, could possibly merge requests in $\mathcal{A}$ with requests in $\mathcal{B}$. Thus, the optimal solution must broadcast at least a $\max\{V_{\mathcal{A}}, V_{\mathcal{B}}\}$ volume of requests during the interval $[t_1 - 2c\alpha^* S_{q,k}, t^*]$. Notice that $\max\{V_{\mathcal{A}}, V_{\mathcal{B}}\} \geq \frac{1}{2}(V_{\mathcal{A}} + V_{\mathcal{B}}) \geq \frac{1}{2}(2+\epsilon)|I|$ and that $|[t_1 - 2c\alpha^* S_{q,k}, t^*]| = 2c\alpha^* S_{q,k} + |I|$. Therefore, it must hold that $\frac{1}{2}(2+\epsilon)|I| \leq 2c\alpha^* S_{q,k} + |I|$. With Lemma 3.6, this simplifies to $c \leq 1 + 4/\epsilon$. This is a contradiction to $c > 1 + 4/\epsilon$. □

Thus, we have the second part of Theroem 1.2 by setting $c = 1 + 5/\epsilon$. Namely that **SSF-W** is $(2+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive for minimizing the maximum delay factor for different sized pages.

4 Lower Bound for a Natural Greedy Algorithm LF

In this section, we consider a natural algorithm which is similar to **SSF-W**. This algorithm, which we will call **LF** for Longest Delay First, always schedules the page which has the largest delay factor. Notice that **LF** is the same as **SSF-W** when $c = 1$. However, we are able to show a negative result on the algorithm for minimizing the maximum delay factor. This demonstrates the importance of the tradeoff between scheduling a request with smallest slack and forcing requests to wait. The algorithm **LF** was suggested and analyzed in our recent work [11] and is inspired by **LWF** which was shown to be $O(1)$-competitive with $O(1)$-speed for average flow time [15]. In [11] **LF** is shown to be $O(k)$-competitive with $O(k)$-speed for L_k norms of flow time and delay factor in broadcast scheduling for unit sized pages. Note that **LF** is a simple greedy algorithm. It was suggested in [11] that **LF** may be competitive for maximum delay factor which is the L_∞-norm of delay factor.

To show the lower bound on the **LF**, we will show that it is not $O(1)$-speed $O(1)$-competitive, even in the standard unicast scheduling setting with unit sized jobs. Since we are considering the unicast setting where processing a page satisfies exactly one request, we drop the terminology of 'requests' and use 'jobs'. We also drop the index of a request $J_{p,i}$ and use J_i since there can only be one request for each page. Let us say that J_i has a wait ratio of $r_i(t) = \frac{t-a_i}{S_i}$ at time $t > a_i$, where a_i and S_i is the arrival time and slack size of J_i. Note that the delay factor of J_i is $\max(1, r_i(f_i))$ where f_i is J_i's finish time. We now formally define **LF**. The algorithm **LF** schedules the request with the largest wait ratio at each time. **LF** can be seen as a natural generalization of **FIFO**. This is because **FIFO** schedules the request with largest wait time at each time. Recall that **SSF-W** forces requests to wait to help merge potential requests in a single broadcast. The algorithm **LF** behaves similarly since it implicitly delays each request until it is the request with the largest wait ratio, potentially merging many requests into a single broadcast. Hence, this algorithm is a natural candidate for the problem of minimizing the maximum delay factor and it does not need any parameters like the algorithm **SSF-W**. However, this algorithm cannot have a constant competitive ratio with any constant speed.

For any speed-up $s \geq 1$ and any constant $c \geq 2$, we construct the following adversarial instance σ. For this problem instance we will show that **LF** has wait ratio at least c, while OPT has wait ratio at most 1. Hence, we can force **LF** to have a competitive ratio of c, for any constant $c \geq 2$. In the instance σ, there are a series job groups $\mathcal{J}_i$ for $0 \leq i \leq k$, where k is a constant to be fixed later. We now fix the jobs in each group. For simplicity of notation and readability, we will allow jobs to arrive at negative times. We can

simply shift each of the times later, so that all arrival times are positive. It is also assumed that s and c are integers in our example.

All jobs in each group $\mathcal{J}_i$ have the same arrival time $A_i = -(sc)^{k-i+1} - \sum_{j=0}^{k-i-1}(sc)^j$ and have the same slack size of $S_i = \frac{c(sc)^{k-i}}{(1-1/sc)^{k-i}}$. There will be $s(sc)^{k+1}$ jobs in the group $\mathcal{J}_0$ and $s(sc)^{k-i}$ jobs in the group $\mathcal{J}_i$ for $1 \leq i \leq k$.

We now explain how **LF** and OPT behave for the instance σ. For simplicity, we will refer to $\mathcal{J}_i$, instead of a job in $\mathcal{J}_i$, since all jobs in the same group are indistinguishable to the schedular. For the first group $\mathcal{J}_0$, **LF** starts and keeps processing $\mathcal{J}_0$ upon its arrival until completing it. On the other hand, we let OPT procrastinate $\mathcal{J}_0$ until OPT finishes all jobs in $\mathcal{J}_1$ to $\mathcal{J}_k$. This does not hurt OPT, since the slack size of the jobs in $\mathcal{J}_0$ is so large. In fact, we will show that OPT can finish $\mathcal{J}_0$ by its deadline. For each group $\mathcal{J}_i$ for $1 \leq i \leq k$, OPT will start $\mathcal{J}_i$ upon its arrival and complete each job in $\mathcal{J}_i$ without interruption. To the contrary, for each $1 \leq i \leq k$ **LF** will not begin scheduling $\mathcal{J}_i$ until the jobs have been substantially delayed. The delay of $\mathcal{J}_k$ is critical for **LF**, since the slack of $\mathcal{J}_k$ is small. For intuitive understanding, we refer the reader to Figure 2.

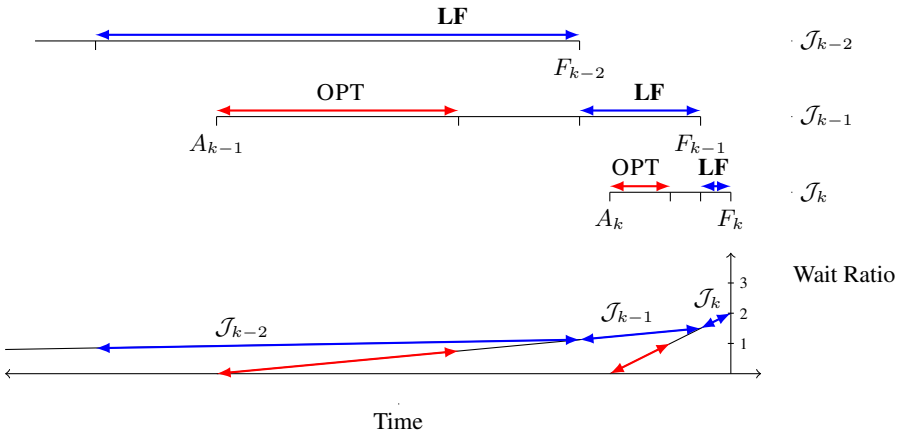

Figure 2: Comparison of scheduling of group $\mathcal{J}_k$, $\mathcal{J}_{k-1}$, and $\mathcal{J}_{k-2}$ by **LF** and OPT.

We now formally prove that **LF** achieves wait ratio c, while OPT has wait ratio at most 1 for the given problem instance σ. Let $F_i = A_i + (sc)^{k-i+1}, 0 \leq i \leq k$. Let R_i be the maximum wait ratio for any job in $\mathcal{J}_i$ witnessed by **LF**. We now define k to be a constant such that $(1 - \frac{1}{sc})^k c \leq \frac{1}{3s}$.

Lemma 4.1. **LF**, *given speed s, processes $\mathcal{J}_0$ during $[A_0, F_0]$ and $\mathcal{J}_i$ during $[F_{i-1}, F_i]$, $1 \leq i \leq k$.*

Proof Sketch: By simple algebra one can check that the length of the time intervals $[A_0, F_0]$ and $[F_{i-1}, F_i]$ is the exact amount of time for **LF** with s-speed needs to completely process $\mathcal{J}_0$ and $\mathcal{J}_i$, respectively.

First we show that $\mathcal{J}_0$ is finished during $[A_0, F_0]$ by **LF**. It can be seen that at time F_0 the jobs in $\mathcal{J}_j$ for $2 \leq j \leq k$ have not arrived, so we can focus on the class $\mathcal{J}_1$. The jobs in $\mathcal{J}_1$ can be shown to have the same wait ratio as the jobs in $\mathcal{J}_0$ at time F_0 and therefore the jobs in $\mathcal{J}_1$ have smaller wait ratio than the jobs in $\mathcal{J}_0$ at all times before F_0. This is because $\mathcal{J}_0$ has a bigger slack than $\mathcal{J}_1$. Hence, **LF** will finish all of the jobs in $\mathcal{J}_0$ before beginning the jobs in $\mathcal{J}_1$.

To complete the proof, we show that $\mathcal{J}_i$ is finished during $[F_{i-1}, F_i]$ by **LF**. It can be seen that at time F_i the jobs in $\mathcal{J}_j$ for $i+2 \leq j \leq k$ have not arrived, so we can focus on the class $\mathcal{J}_{i+1}$. The jobs in $\mathcal{J}_i$ can be shown to have the same wait ratio as the jobs in $\mathcal{J}_{i+1}$ at time F_i and therefore the jobs in $\mathcal{J}_{i+1}$ have smaller wait ratio than the jobs in $\mathcal{J}_i$ at all times before F_i. Hence, **LF** will finish all of the jobs in $\mathcal{J}_i$ before beginning the jobs in $\mathcal{J}_{i+1}$.

□

Using Lemma 4.1 and the given arrival times of each of the jobs we have the following lemma.

Lemma 4.2. $R_i = c(1 - \frac{1}{sc})^{k-i}$ *for* $0 \leq i \leq k$.

Notice that Lemma 4.2 implies that $R_k \geq c$. Hence, the maximum delay factor witnessed by **LF** is at least c. In the following lemma, we show that there exists a valid scheduling by OPT where the maximum wait ratio is at most one. This will show that **LF** achieves a competitive ratio of c. Note that Lemma 4.2 shows that $R_0 \leq \frac{1}{3s}$.

Lemma 4.3. *Consider a schedule which processes each job in* $\mathcal{J}_0$ *during* $[F_k, F_k + |\mathcal{J}_0|]$ *and each job in* $\mathcal{J}_i$ *during* $[A_i, A_i + |\mathcal{J}_i|]$ *for* $1 \leq i \leq k$. *This schedule is valid and, moreover, the maximum wait ratio witnessed by this schedule is at most one.*

Proof Sketch: It is not hard to show that the time intervals $[F_k, F_k + |\mathcal{J}_0|]$ and $[A_i, A_i + |\mathcal{J}_i|]$ for $1 \leq i \leq k$ do not overlap, therefore this is a valid schedule.

The wait ratio witnessed by the jobs in groups $\mathcal{J}_i$ for $1 \leq i \leq k$ can easily be seen to be at most 1. This is because this schedule processes each of these jobs as soon as they arrive. We now show that the wait ratio of each of the jobs in $\mathcal{J}_0$ is at most 1. Recall that R_0, the maximum wait ratio of $\mathcal{J}_0$ by **LF** is at most $\frac{1}{3s}$ at time F_0. Using the fact $sc \geq 2$, we can easily show that $|F_k - A_0| \leq 2|F_0 - A_0|$. Note that OPT can finish $\mathcal{J}_0$ during $[F_k, F_k + s|F_0 - A_0|]$, since **LF** with s-speed could finish $\mathcal{J}_0$ during $[A_0, F_0]$. Thus the wait ratio of $\mathcal{J}_0$ at time $F_k + s|F_0 - A_0|$ is at most $\frac{2+s}{3s} \leq 1$. □

From Lemma 4.2 the maximum delay factor witnessed by **LF** is c and by Lemma 4.3 the maximum delay factor witnessed by OPT is 1. Hence we have the proof of Theorem 1.4.

5 Conclusion

In this paper, we showed an almost fully scalable algorithm[1] for minimizing the maximum delay factor in broadcasting for unit sized jobs. The slight modification we make to **SSF-W** from [12] makes the algorithm more practical. Using the intuition developed for the maximum delay factor, we proved that **FIFO** is in fact 2-competitive for varying sized jobs closing the problem for minimizing the maximum response time online in broadcast scheduling.

We close this paper with the following open problems. Although the new algorithm for the maximum delay factor with unit sized jobs is almost fully scalable, it explicitly depends on speed given to the algorithm. Can one get another algorithm independent of this dependency? For different sized pages, it is still open on whether there exists a $(1 + \epsilon)$-speed algorithm that is $O(1)$-competitive. For minimizing the maximum response time offline it is of theoretical interest to show a lower bound on the approximation ratio that can be achieved or to show an algorithm that is a c-approximation for some $c < 2$.

References

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

[2] Demet Aksoy and Michael J. Franklin. "rxw: A scheduling approach for large-scale on-demand data broadcast. *IEEE/ACM Trans. Netw.*, 7(6):846–860, 1999.

[1] An algorithm is said to be almost fully scalable if for any fixed $\epsilon > 0$, it is $O(1 + \epsilon)$-speed $O(1)$-competitive.

[3] Nikhil Bansal, Moses Charikar, Sanjeev Khanna, and Joseph (Seffi) Naor. Approximating the average response time in broadcast scheduling. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 215–221, 2005.

[4] Nikhil Bansal, Don Coppersmith, and Maxim Sviridenko. Improved approximation algorithms for broadcast scheduling. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 344–353, 2006.

[5] Yair Bartal and S. Muthukrishnan. Minimizing maximum response time in scheduling broadcasts. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 558–559, 2000.

[6] Michael A. Bender, Soumen Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 270–279, 1998.

[7] Michael A. Bender, Raphaël Clifford, and Kostas Tsichlas. Scheduling algorithms for procrastinators. *J. Scheduling*, 11(2):95–104, 2008.

[8] Michael A. Bender, S. Muthukrishnan, and Rajmohan Rajaraman. Improved algorithms for stretch scheduling. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 762–771, 2002.

[9] Wun-Tat Chan, Tak Wah Lam, Hing-Fung Ting, and Prudence W. H. Wong. New results on on-demand broadcasting with deadline via job scheduling with cancellation. In Kyung-Yong Chwa and J. Ian Munro, editors, *COCOON*, volume 3106 of *Lecture Notes in Computer Science*, pages 210–218, 2004.

[10] Jessica Chang, Thomas Erlebach, Renars Gailis, and Samir Khuller. Broadcast scheduling: algorithms and complexity. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 473–482. Society for Industrial and Applied Mathematics, 2008.

[11] Chandra Chekuri, Sungjin Im, and Benjamin Moseley. Longest wait first for broadcast scheduling. Manuscript, 2009.

[12] Chandra Chekuri and Benjamin Moseley. Online scheduling to minimize the maximum delay factor. In *SODA '09: Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithm*, 2009.

[13] Marek Chrobak, Christoph Dürr, Wojciech Jawor, Lukasz Kowalik, and Maciej Kurowski. A note on scheduling equal-length jobs to maximize throughput. *J. of Scheduling*, 9(1):71–73, 2006.

[14] Jeff Edmonds and Kirk Pruhs. Multicast pull scheduling: When fairness is fine. *Algorithmica*, 36(3):315–330, 2003.

[15] Jeff Edmonds and Kirk Pruhs. A maiden analysis of longest wait first. *ACM Trans. Algorithms*, 1(1):14–32, 2005.

[16] Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. In *SODA '09: Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithm*, 2009.

[17] Thomas Erlebach and Alexander Hall. Np-hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 194–202, 2002.

[18] Rajiv Gandhi, Samir Khuller, Yoo-Ah Kim, and Yung-Chun (Justin) Wan. Algorithms for minimizing response time in broadcast scheduling. *Algorithmica*, 38(4):597–608, 2004.

[19] Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. *J. ACM*, 53(3):324–360, 2006.

[20] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

[21] Bala Kalyanasundaram, Kirk Pruhs, and Mahendran Velauthapillai. Scheduling broadcasts in wireless networks. *Journal of Scheduling*, 4(6):339–354, 2000.

[22] Jae-Hoon Kim and Kyung-Yong Chwa. Scheduling broadcasts with deadlines. *Theor. Comput. Sci.*, 325(3):479–488, 2004.

[23] Kirk Pruhs. Competitive online scheduling for server systems. *SIGMETRICS Perform. Eval. Rev.*, 34(4):52–58, 2007.

[24] Kirk Pruhs, Jiri Sgall, and Eric Torng. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter Online Scheduling. 2004.

[25] Kirk Pruhs and Patchrawat Uthaisombut. A comparison of multicast pull models. *Algorithmica*, 42(3-4):289–307, 2005.

[26] J. Wong. Broadcast delivery. *Proceedings of the IEEE*, 76(12):1566–1577, 1988.

[27] Feifeng Zheng, Stanley P. Y. Fung, Wun-Tat Chan, Francis Y. L. Chin, Chung Keung Poon, and Prudence W. H. Wong. Improved on-line broadcast scheduling with deadlines. In Danny Z. Chen and D. T. Lee, editors, *COCOON*, volume 4112 of *Lecture Notes in Computer Science*, pages 320–329, 2006.

A Omitted Proofs

A.1 Proof of Lemma 3.1

Proof. Let $J_{x,i}, J_{x,j} \in \mathcal{J}_I$ such that $i < j$. Let t' be the time that **SSF-W** starts satisfying request $J_{x,i}$. By the definition of I, request $J_{x,i}$ must have delay factor at least α^* at time $f_{x,i}$. We also know that the request $J_{x,j}$ must arrive after time t', otherwise request $J_{x,j}$ must also be satisfied at time t'. If the optimal solution combines these requests into a single broadcast then the request $J_{x,i}$ must wait until the request $J_{x,j}$ arrives to be satisfied. However, this means that the request $J_{x,i}$ must achieve a delay factor greater than α^* by OPT, a contradiction of the definition of α^*. □

A.2 Proof of Lemma 3.6

Proof. The request $J_{q,k}$ has delay factor at least $c\alpha^*$ at any time t during $I' = [t', t^*]$, where $t' = t^* - (c^2 - c)S_{q,k}\alpha^*$. The largest delay factor any request can have during I' is less than $c^2\alpha^*$ by definition of t^* being the first time **SSF-W** witnesses delay factor $c^2\alpha^*$. Hence the request $J_{q,k}$ is in the queue $Q(t)$ at any time t during I'. Therefore, any request that forced **SSF-W** to broadcast on I', must have delay factor at least α^* and since $J_{q,k} \in Q(t)$ for all $t \in I'$, the requests scheduled on I' must have slack at most $S_{q,k}$. □

A.3 Proof of Lemma 3.7

Proof. For the sake of contradiction, suppose that some request $J_{p,i}$ that forced **SSF-W** to broadcast page p on the interval I arrived at time $t' < t_1 - 2S_{q,k}\alpha^* c$. Recall that $J_{p,i}$ has a slack no bigger than $S_{q,k}$ by the definition of I. Therefore at time $t_1 - S_{q,k}\alpha^* c$, $J_{p,i}$ has a delay factor of at least $c\alpha^*$. Thus any request scheduled during the interval $[t_1 - S_{q,k}\alpha^* c, t_1]$ has a delay factor no less than α^*. We observe that $J_{p,i}$ is in $Q(\tau)$ for $\tau \in [t_1 - S_{q,k}\alpha^* c, t_1]$; otherwise must be a request with a delay factor bigger than $c^2\alpha^*$ at time τ and this is a contradiction to the assumption that t^* is the first time that **SSF-W** witnessed a delay factor of $c^2\alpha^*$. Therefore any request that forced **SSF-W** to broadcast during $[t_1 - S_{q,k}\alpha^* c, t_1]$ has a slack no bigger than $S_{p,i}$. Also we know that $S_{p,i} \leq S_{q,k}$ by the definition of I. In sum, we showed that any request that forced **SSF-W** to do a broadcast during $[t_1 - S_{q,k}\alpha^* c, t_1]$ have a slack no bigger than $S_{q,k}$ and a delay factor no smaller than α^*, which is a contradiction of the definition of t_1. □