Broadcast Scheduling: Algorithms and Complexity

Jessica Chang* Thomas Erlebach† Renars Gailis‡ Samir Khuller §

Abstract

Broadcast Scheduling is a popular method for disseminating information in response to client requests. There are n pages of information, and clients request pages at different times. However, multiple clients can have their requests satisfied by a single broadcast of the requested page. In this paper we consider several related broadcast scheduling problems. One central problem we study simply asks to minimize the maximum response time (over all requests). Another related problem we consider is the version in which every request has a release time and a deadline, and the goal is to maximize the number of requests that meet their deadlines. While approximation algorithms for both these problems were proposed several years back, it was not known if they were NP-complete. One of our main results is that *both* these problems are NP-complete. In addition, we use the same unified approach to give a simple NP-completeness proof for minimizing the sum of response times. A very complicated proof was known for this version. Furthermore, we give a proof that FIFO is a 2-competitive online algorithm for minimizing the maximum response time (this result had been claimed earlier with no proof) and that there is no better deterministic online algorithm (this result was claimed earlier as well, but with an incorrect proof).

1 Introduction

Broadcasting is a widely used mechanism to disseminate data since multiple clients can have their requests satisfied simultaneously. A very large amount of work in the database and algorithms literature has focused on scheduling problems based on a broadcasting model [5, 2, 1, 7, 20] (including several Ph.D. theses from Maryland and Brown). Broadcasting is used in commercial systems, including the Intel Intercast System and the Hughes DirecPC system. We focus our attention on *pull-based* schemes, where clients request the data that they need and the data is delivered on a fast broadcast medium (often using wireless channels). Broadcast scheduling is becoming extremely relevant due to the proliferation of wireless technologies.

A key consideration is the design of a good broadcast schedule. The challenge is in designing an algorithm that *guarantees good response time* [7]. While the practical problem is clearly online [2, 19, 11, 21], it is interesting to study the complexity of the offline problem as well. In fact, a lot of recent algorithmic work has focused on minimizing the sum of response times [17, 13, 14, 15, 3, 4] as well as minimizing the maximum response time [7, 9].

In trying to evaluate the performance of online algorithms, it is useful to compare them to an optimal offline solution. In addition, when the demands are known for a small window of time into the future (also called the look-ahead model in online algorithms), being able to quickly compute an optimal offline solution can be extremely useful. Many kinds of demands for data (e.g., web traffic) exhibit good predictability over the short term, and thus knowledge of requests in the immediate future leads to a situation where one is trying to compute a good offline solution.

The informal description of the problem is as follows. There are n data items, $1, \ldots, n$, called pages. Time is broken into "slots". A time slot is defined as the unit of time to transmit one page on the wireless channel. Time slot t is the unit of time between time $t-1$ and t. A request for a page j arrives at time t and then waits. When page j has been transmitted, this request has been satisfied. The response time of a request is its waiting time. Several different objective functions can be considered. The one that has been the most studied is the one in which we wish to minimize the sum of response times.

In this paper we present the following results.

- For the offline problem of *minimizing the maximum response time* we show that the problem is NP-complete. Even though a simple 2-approximation algorithm was given by Bartal and Muthukrishnan [7] for the offline problem, there was no known proof

*Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195. Research done while this author was at the University of Maryland and supported by a Senior Summer Scholar award from the University of Maryland and by an NSF REU Supplement to Award CCF-0430650. E-mail : `jschang@cs.washington.edu`.

†Department of Computer Science, University of Leicester, Leicester, LE1 7RH, UK. E-mail: `te17@mcs.le.ac.uk`.

‡This work was done while this author was a graduate student at the University of Maryland.

§Research supported by NSF Awards CCF-0430650 and CCF-0728839. Department of Computer Science, University of Maryland, College Park, MD 20742. E-mail : `samir@cs.umd.edu`.

that the offline version is *NP*-complete. Moreover this also shows that the problem of scheduling a pre-specified number of requests within a certain delay is *NP*-complete (a 5-approximation was presented for this problem by Charikar and Khuller [9]). This closes a central open problem in this area.

- We also show a much simpler *NP*-completeness proof of the problem of minimizing the sum of response times. This problem was previously shown to be *NP*-complete by Erlebach and Hall [13], but by a much more complicated proof.

- For *minimizing the maximum response time* we give a proof that FIFO is 2-competitive in the online model where requests are not known in advance. This result was claimed in the paper by Bartal and Muthukrishnan [7] several years back, but no proof of this claim has been provided since. We also give a construction showing that there is no $(2-\epsilon)$-competitive deterministic online algorithm for any $\epsilon > 0$. This was also claimed in [7], but the construction given is incorrect.

- In addition, we are interested in the problem of scheduling the maximum number of requests with pre-specified windows. In other words, each request is released at a certain time and gives a specific deadline by which it should be satisfied. For this maximization problem a $\frac{3}{4}$-approximation was developed by Gandhi et al. [15], which is still the best known bound (an improved bound of $\frac{5}{6}$ was claimed by Bansal et al. [4], but has since been withdrawn). This algorithm works by rounding an LP relaxation of a natural Integer Program (IP) formulation. We show that this LP formulation has a gap of $\frac{17}{18}$. This suggests that this is the limit of any LP rounding approach that uses this IP formulation. In addition, using standard LP rounding techniques [14] (details omitted) we can show that all requests can be met by their deadlines using a 2-speed server, provided a fractional solution meeting all deadlines exists. Furthermore, we show that this problem has a very simple *NP*-completeness proof. In contrast, in the traditional scheduling version, even when requests are not limited to integral times, the problem can be solved in $O(n^5)$ time [10].

- Another way to relax the problem of scheduling requests within windows is to find a minimum *delay factor* α such that there is a broadcast schedule in which every request (j,t) is satisfied by time $t+\alpha(D_t^j - t)$, where D_t^j is the deadline for request (j,t). We show that if there is a $2-\epsilon$ approximation for the minimum delay factor, then $P = NP$.

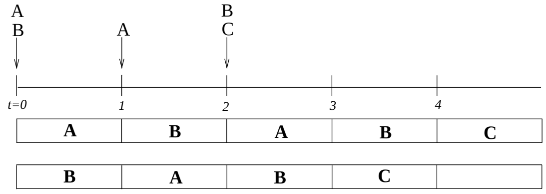

Figure 1: We show the schedules produced by FIFO, and an optimal offline schedule to minimize the maximum response time.

It is important to note that all the problems described above are completely trivial in the standard scheduling model where at each time step only one unit length job can be scheduled, and the server cannot handle multiple requests by scheduling the job once. For example, one can set up a bipartite graph with time slots on one side and job requests on the other side and solve the problem using matchings. A sample schedule is shown in Fig. 1.

The main contribution here is a unified approach to proving NP-completeness for several different objective functions for broadcast scheduling. Previously, the *only* version that was shown to be NP-complete was the min-sum version, and for that too the proof is extremely complex and is the primary result of the SODA 2002 paper by Erlebach and Hall [13]. In contrast, our proofs are very short and elementary. The paper by Bartal and Muthukrishnan [7] has several claims without proof, and no full version has ever appeared. Moreover as we show, the proof for the lower bound on the competitive ratio was actually incorrect. The proof for FIFO is interesting for that reason.

1.1 Formal Problem Definition. The problem is formally stated as follows. There are n possible pages, $P = \{1, 2, \ldots, n\}$. We assume that time is discrete and at time t, any subset of pages can be requested. Let (p,t) represent a request for page p at time t. Let r_t^p denote the number of requests (p,t). A time slot t is the window of time between time $t-1$ and time t. The server can broadcast a page in each time slot (see Fig. 1). When a page is broadcast in time-slot t, we will simply say that it has been broadcast at time t. We say that a request (p,t) is satisfied at time S_t^p, if S_t^p is the first time instance *after* t when page p is broadcast. The response time of request (p,t) is $S_t^p - t$.

1.2 Related Work. For the problem of minimizing the average response time, Kalyanasundaram et al. [17] showed that for any fixed $\epsilon, 0 < \epsilon \leq \frac{1}{3}$, it is possible to obtain a $\frac{1}{\epsilon}$-speed $\frac{1}{1-2\epsilon}$-approximation algorithm for minimizing the average response time, where a k-speed algorithm is one where the server is allowed to broadcast k pages in each time slot. For example by setting $\epsilon = \frac{1}{3}$ they obtain a 3-speed, 3-approximation. The approximation factor bounds the cost of the k-speed solution compared to the cost of an optimal 1-speed solution. (This kind of approximation guarantee is also referred to as a "bicriteria" bound in many papers.) Note that we cannot set $\epsilon = \frac{1}{2}$ to get a 2-speed, constant approximation. Their algorithm is based on rounding a fractional solution for a "network-flow" like problem that is obtained from an integer programming formulation (see [18] for the equivalence between the formulations in [17] and [14]). This problem has been shown to be NP-hard by Erlebach and Hall [13]. Recently, Gandhi et al. (see [16]) obtained a 2-speed 1-approximation, improving the results given earlier [14, 13, 15]. However, recently a $1 + \epsilon$ speed, $O(1/\epsilon)$ approximation for any $\epsilon > 0$ is also presented [3]. Bansal et al. [4] recently obtained an $O(\log^2 n)$ approximation for this measure without any increase in the speed, improving on the previous best result of $O(\sqrt{n})$ [3].

Another problem that has been considered before is that of maximizing throughput in broadcast scheduling. Here, the model is that every request is associated with a deadline and some requests can be dropped by the algorithm. The goal is to maximize the number of requests satisfied by their deadlines. The results of Bar-Noy et al. [6] imply a 1/2-approximation for this problem. This was improved to factor 3/4 by Gandhi et al. [15]. Even though approximation algorithms were developed, the complexity of the problem was not known. We show that this problem is *NP*-complete.

For the problem of minimizing the average response time, online competitive algorithms were given by [11]. These have an $O(1)$ competitive ratio, using $O(1)$ speed. In the online model a $\frac{1}{2}$ competitive algorithm was given when each request has a release time and deadline [19], and the objective is to maximize the throughput. If time is not slotted (i.e. arrival and broadcast times are not limited to integral times), they also show that if preemption is allowed and pages have unit length, a competitive ratio of 5.828 can be achieved [19]. This bound was later improved to 5 by [8] and again to 4.56 by [21].

2 NP-hardness of Minimizing the Maximum Response Time

We now consider the problem of minimizing the maximum response time.

THEOREM 2.1. *Broadcast scheduling with the objective of minimizing the maximum response time is NP-hard.*

Proof. We consider the decision version of the problem where a bound R on the maximum response time is given as part of the input and the task is to decide whether a schedule with maximum response time at most R exists. We show that this decision problem is *NP*-complete. It is easy to see that the problem is contained in *NP*. We prove that the problem is *NP*-hard by a polynomial reduction from the vertex cover problem.

Let an instance of the vertex cover problem be given by a graph $G = (V, E)$ and a bound k on the size of the vertex cover. The goal is to decide whether G has a vertex cover of size k. Let $n = |V|$ and $m = |E|$. We construct an instance of the decision version of broadcast scheduling with a bound of $R = 2n$ on the maximum response time. The bound on the maximum response time can be viewed as assigning to each request (p, t) a deadline of $t + 2n$.

First, we outline the basic idea behind the construction. There is a page corresponding to each vertex in V. We refer to these pages as *vertex pages*, and to all other pages used in the construction as *extra pages*. We specify requests for extra pages and vertex pages in such a way that any schedule with maximum response time $R = 2n$ must partition the set of vertex pages into a subset V_k of size k and a subset V_{n-k} of size $n-k$. Furthermore, any such schedule must repeatedly broadcast all pages from V_k, then n extra pages, then all pages from V_{n-k}, and then again n extra pages. This structure is repeated m times, and the partition (V_k, V_{n-k}) is forced to be the same in all repetitions (but the order in which the pages in each part of the partition are broadcast is arbitrary in each repetition of the structure).

Finally, in each of the repetitions of this structure, we pick a different edge $\{u, v\} \in E$ and add requests for u and v, called *edge requests*, a suitable number of time steps before the schedule begins broadcasting the pages of V_k. Here, "suitable" means that the deadline for the edge requests is exactly when the schedule broadcasts the first page of V_{n-k}. If V_k corresponds to a vertex cover, the broadcast of the pages in V_k will satisfy at least one of the two edge requests, and the other one can be satisfied by the first page of V_{n-k} that is broadcast just before the deadline of the edge requests. If G does not have a vertex cover of size k, the broadcast of V_k will satisfy none of the two edge requests for some edge

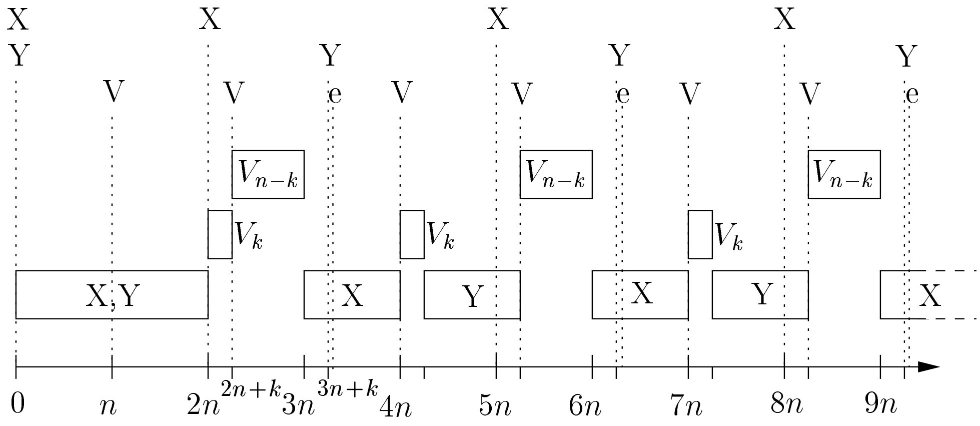

Figure 2: Illustration of the construction used in the *NP*-completeness proof for minimizing the maximum response time. The boxes correspond to broadcasts of the pages of the respective set (X, Y, V_k, or V_{n-k}). The letters above the boxes, together with the dotted vertical lines, indicate the times when requests for the pages in the corresponding set are made; the letter 'e' indicates a pair of edge requests.

$\{u, v\}$, and the schedule will have a maximum response time of at least $R + 1$.

Now we give the formal proof. Denote the pages corresponding to the vertices of G by $v_1, \ldots, v_n$. We use two sets of extra pages, $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$, thus $2n$ extra pages in total. We create the following requests (referred to as *basic requests*) to force the periodic structure (as discussed above) of the schedule. See Fig. 2 for an illustration.

- At time 0, all $2n$ extra pages are requested.
- At time n, all n vertex pages are requested.
- At time $2n + k + 3ni$, for $0 \le i < m$, and at time $n + 3ni$, for $1 \le i \le m$, all n vertex pages are requested.
- At time $2n + 3ni$, for $0 \le i \le m$, extra pages $x_1, \ldots, x_n$ are requested.
- At time $k + 3ni$, for $1 \le i \le m$, extra pages $y_1, \ldots, y_n$ are requested.

Finally, we add the edge requests: Let $e_1, \ldots, e_m$ be the edges of G in arbitrary order, and let $e_i = \{v_{i_1}, v_{i_2}\}$. For $1 \le i \le m$, we request pages v_{i_1} and v_{i_2} at time $k + 1 + 3ni$. This completes the description of the construction.

Now we prove correctness of the construction. First, consider the basic requests and any schedule that meets all deadlines. From time 1 to $2n$, all $2n$ extra pages must be broadcast. As the vertex pages are requested at time n, they must be broadcast from time $2n+1$ to $3n$ to meet their deadlines. Denote by V_k the set of vertex pages broadcast from time $2n+1$ to $2n+k$, and by V_{n-k} the remaining vertex pages. The extra pages $x_1, \ldots, x_n$ requested at time $2n$ must be broadcast from time $3n+1$ to $4n$. As the vertex pages were requested again at time $2n + k$, the requests for pages from V_k are still not satisfied by time $4n$, and so they must be broadcast from time $4n + 1$ to $4n + k$. The extra pages $y_1, \ldots, y_n$ requested at time $3n + k$ must then be broadcast from time $4n + k + 1$ to $5n + k$. The vertex pages were again requested at time $4n$, and only the requests for pages in V_k are already satisfied by time $5n + k$. Hence, the pages in V_{n-k} must be broadcast from time $5n + k + 1$ to $6n$. The extra pages $x_1, \ldots, x_n$ that were requested at time $5n$ must then be broadcast from time $6n + 1$ to $7n$. Note that the situation at time $6n$ is identical to the situation at $3n$. Hence, the structure of the schedule from time $3n+1$ to $6n$ repeats every $3n$ time steps, with m repetitions in total.

We see that a schedule meets all deadlines of the basic requests if and only if it has the described structure. In particular, for $1 \le i \le m$, the schedule from time $3ni + 1$ to $3n(i + 1)$ must broadcast pages as follows:

- first, extra pages $x_1, \ldots, x_n$ in arbitrary order,
- then, the pages in V_k in arbitrary order,
- then, extra pages $y_1, \ldots, y_n$ in arbitrary order,
- then, the pages in V_{n-k} in arbitrary order.

Finally, consider the edge requests for the endpoints of edge $e_i = \{v_{i_1}, v_{i_2}\}$. They are made at time $3ni + k + 1$. The only broadcasts that can potentially satisfy these two edge requests are the broadcasts of pages from V_k in time steps $3ni+n+1$ to $3ni+n+k$, and the broadcast of a page from V_{n-k} in time step $3ni+2n+k+1$. If at least one of the endpoints of e_i is in V_k, a feasible schedule that meets all deadlines can be obtained (since the page of V_{n-k} that is broadcast at time $3ni+2n+k+1$ can be chosen arbitrarily). If for some edge none of its endpoints are in V_k, at least one request will not meet its deadline. Hence, a schedule with response time $2n$ exists if and only if G has a vertex cover of size k. □

3 NP-hardness of Minimizing the Total Response Time

The total response time is defined as $\sum_t \sum_p r_t^p (S_t^p - t)$. The aim is to minimize this sum.

THEOREM 3.1. *Broadcast scheduling with the objective of minimizing the total response time is NP-hard.*

Proof. We again reduce the vertex cover problem to the broadcast scheduling problem. If the optimal objective function of the formulated schedule falls into a certain range, then the vertex cover exists, and vice versa. An illustration of the reduction is given in Fig. 3.

Let an instance of the vertex cover problem be given by a graph $G = (V, E)$ and a bound k on the size of the vertex cover. The goal is to decide whether G has a vertex cover of size k. Let $n = |V|$, $m = |E|$ and $\ell = n-k$. For each node in V we will associate a *vertex page*; we will also use extra pages $X_1, \ldots, X_{2m+1}$ to ensure that either requests are satisfied in a particular way or the total response time is higher than the specified range.

Suppose that at time $t = 0$, we request each of the n vertex pages $2m+1$ times. Then the best schedule will contribute to the total response time a quantity of $(2m+1)(\sum_{i=1}^n i)$. Note that the best schedule (as defined thus far) is not unique; all permutations of the n vertex pages will yield the same quantity. This will allow us some freedom in changing the order of pages broadcast. This will be our building block (call it a "vertex block"). Within this block, we want all vertex pages scheduled in interval $[0, \ell]$ to correspond to vertices that are not in our vertex cover; similarly, vertex pages in $[\ell, n]$ should form a vertex cover of size k.

We will place m vertex blocks, with each pair of blocks separated by a "dead block" of length $2m+1$. At the beginning of each dead block, we request the $2m+1$ extra pages $X_1, \ldots, X_{2m+1}$. Each extra page will be requested $(2m+1)^4$ times; i.e. $r_t^{X_i} = (2m+1)^4$. The intuition behind the insertion of dead blocks is that these extra pages are requested in such high quantity that the optimal schedule must schedule them right away or else pay too high a cost to the objective function; hence no requests for the vertex pages can be satisfied during this interval.

For each vertex block, we make another $2m+1$ requests for each of the n distinct pages at the block offset ℓ. Note that waiting until after the dead block to satisfy a request made before the dead block is expensive, especially since each time a vertex page is requested, it is requested $2m+1$ times. However, such a situation is unavoidable given that all n vertex pages are requested at block offset ℓ. The optimal schedule should minimize the number of requests for which this occurs.

The best schedule will have following properties:

1. Each vertex page is scheduled exactly once in every vertex block;
2. If a vertex page is scheduled in $[0, \ell]$, then it must be scheduled in all vertex blocks in offset $[0, \ell]$;
3. Likewise, if a vertex page appears in $[\ell, n]$, then it is also in all blocks in offset $[\ell, n]$.

The first property must be satisfied in the optimal solution. If it is not, then we will pay an extra penalty of at least $(2m+1)$ to the objective function, which will be too much. If either the second or third property is not satisfied, we will pay an extra penalty at least $2m+1$: suppose a vertex page n_i is scheduled in offset $[0, \ell]$. Then it is not scheduled in offset $[\ell, n]$. The second $2m+1$ requests for n_i made at block offset ℓ will remain unsatisfied until after the dead-block. If in the next vertex block, n_i is scheduled after offset $[0, \ell]$, we pay an extra cost of at least $2m+1$ to the objective function. The argument is similar for the third property. Let K be the objective function of the optimal schedule for the construction thus far.

We now add requests specific to the edges in G. Let each edge $e = (u, v)$ in G correspond to a vertex block. Add 2 more requests at offset $\ell - 1$ of this block: one request for vertex page u and one for vertex page v. Since the permutation of scheduled vertex pages within each interval does not matter, they can be arranged as pleased. For each edge and its corresponding vertex block, there are three cases that can occur:

1. Both vertex pages u and v appear in block offset $[0, \ell]$. Then one of the requests will be satisfied no earlier than in the next vertex block, which is at least $2m+1+k$ time-slots away, therefore paying this as an extra amount to the objective function.
2. One vertex page is in offset $[0, \ell]$ and the other is in $[\ell, n]$. Then both requests can be satisfied paying 3 to objective function.

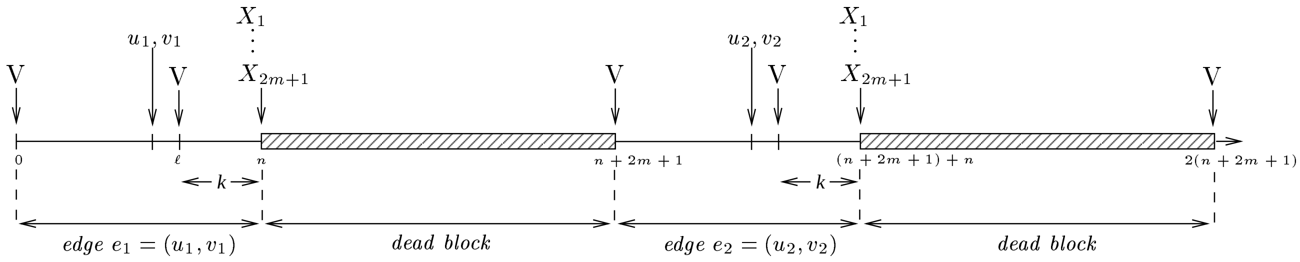

Figure 3: V denotes the set of vertex pages. Each of the m vertex blocks corresponds to an edge e_i and is followed by a dead block, i.e., an interval when pages X_i must be scheduled.

3. Both pages are in $[\ell, n]$; then both requests can be satisfied paying 5 to the objective function.

If the vertex cover exists, the resulting optimal objective function will be between $K+3m$ and $K+5m$. If no such vertex cover exists, then there exists an edge that falls into case 1, and we pay an extra $2m+1$ to the objective function, pushing it above the $K+5m$ limit.

Conversely, if there is a schedule with an objective function less than or equal to $K+5m$, then it is impossible for case 1 to be true for any vertex block in the schedule. Thus for every edge, at least one of the incident vertices must be in block offset $[\ell, n]$ and so it is in the vertex cover. □

4 Broadcast Scheduling with Deadlines

THEOREM 4.1. *Broadcast scheduling with windows is NP-hard.*

Proof. The NP-completeness of this problem follows from the proof that Broadcast scheduling to minimize the maximum response time is NP-hard, since that is a special case when all the requests have identical length windows.

However, for the windows scheduling problem there is a very simple reduction from vertex cover which is given next. As before, the instance of the vertex cover problem is described by a graph $G = (V, E)$ and a bound k. We construct an instance of broadcast scheduling with windows, and ask if this instance has a schedule satisfying all the requests. This will be shown to be the case if and only if G has a vertex cover of size k. Each request (p, t) should be satisfied by its deadline D_t^p. The window is the time between t and D_t^p. We only use two window sizes – n and 3. Let $\ell = n - k$ (see Fig. 4).

There is a distinct page for each vertex in V. We refer to these pages as *vertex pages*. We request the entire set of pages in V at times $0, n, 2n, 3n, \ldots, (m-1)n$. Each block of requests for V will be referred to as an edge block. We also request the entire set V again at times $\ell, n+\ell, 2n+\ell$ etc., once in each of the m edge blocks. Each vertex request has a window size of n. We now add a set of *edge requests*. For the i^{th} edge (u_i, v_i) we add a request at time $n(i-1)+\ell-1$ for both the pages u_i, v_i with a window size of 3.

We first argue that if the graph has a vertex cover of size k then a feasible solution exists that satisfies all the requests. Let V_k be the k nodes forming a vertex cover. Let V_ℓ be the remaining vertices. In each edge block we first schedule V_ℓ, and then schedule V_k. Thus all the requests at time 0 are satisfied within their windows. Note that the requests made at time ℓ for all pages in V are also satisfied since first V_k is broadcast and then V_ℓ is broadcast from the next edge block. Notice that within the group V_ℓ and V_k we have control over the precise permutation of pages within each group.

Consider the requests made for u_i, v_i at time $n(i-1)+\ell-1$. Note that the window length is 3. If both u_i and v_i are in V_k (nodes forming the vertex cover) then we can schedule these two pages as the first two pages in V_k. If $u_i \in V_\ell$ and $v_i \in V_k$ then we schedule u_i as the last page broadcast in V_ℓ and v_i as the first page broadcast in V_k. The other case is identical. We can thus argue that if every edge has at least one end point in V_k, then within its window of size 3 the request is satisfied.

The proof in the other direction is not difficult and left for the reader. □

The above proof can however be modified to show the following.

THEOREM 4.2. *If there is a $(2-\epsilon)$ approximation for minimizing the delay factor for broadcast scheduling problem with deadlines, then $P = NP$.*

Proof. We show that it is NP-hard to distinguish instances where the optimal solution has delay factor $\alpha = 1$ from instances where any solution has $\alpha \geq 2$.

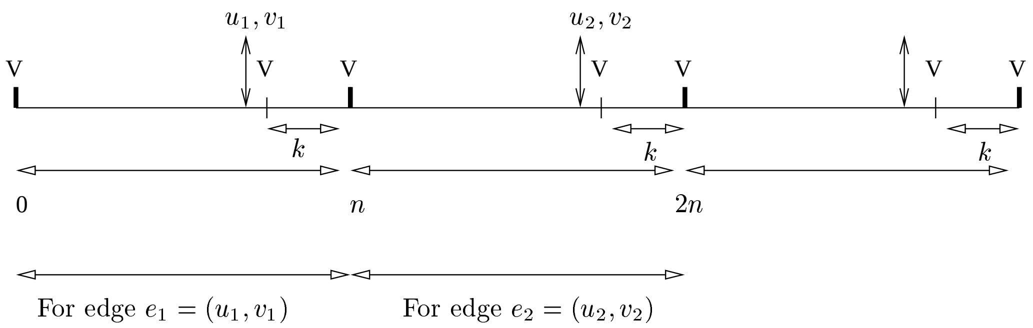

Figure 4: Reduction from Vertex Cover to Broadcast Scheduling with windows. V denotes the set of all n vertex pages, where n is the number of vertices of the given graph.

The reduction is similar to the proof of Theorem 4.1. Again, we start with an instance of vertex cover given by a graph $G = (V, E)$ and an integer k. Let $n = |V|$ and $m = |E|$. We construct an instance of broadcast scheduling with deadlines such that a schedule with $\alpha = 1$ exists if and only if G has a vertex cover of size at most k, and $\alpha \geq 2$ for any schedule otherwise. An approximation algorithm with ratio $2-\varepsilon$ would thus allow to distinguish between yes-instances and no-instances of the vertex cover problem.

Compared to the proof of Theorem 4.1, the main additional trick is to block parts of the schedule using requests that have window size 1. In any schedule with $\alpha < 2$, one cannot afford to delay such a request, and so no other request can be scheduled in such a step.

The set of pages contains a vertex page v for every $v \in V$, and a blocking page b. We force a schedule structure (for any schedule with $\alpha < 2$) that consists of m repetitions of a basic structure. The r-th repetition of the basic structure is as follows:

(1) A schedule of a set S of $n - k$ pages from V.

(2) A blocked part of length $2rn - 2n$.

(3) A schedule of the set $C = V \setminus S$ of k pages from V.

(4) A blocked part of length $2rn - n$.

In each time step of a blocked part, the blocking page b is requested with a window of size 1. This ensures that no vertex page can be scheduled in a blocked part. Note that the length of the blocked parts grows linearly with r. All pages V are requested just before (1) and just before (3) in each repetition. The deadline for the pages requested just before (1) is just before (4) in the same repetition (so the window length for these requests is $2rn - n$), and the deadline for the pages requested just before (3) is just before (2) in the next repetition (window length $2rn$). Note that delaying a request beyond the blocked part after its deadline increases the delay factor to at least 2. Thus, as in the proof of Theorem 4.1, it is ensured that the sets S and C of vertex pages scheduled in (1) and (3) are the same in each repetition. Furthermore, the pages in S and C can be scheduled in an arbitrary permutation in each repetition.

Assign each edge of the graph G to a distinct repetition. In the repetition to which the edge $\{u, v\}$ is assigned, we request u and v one time step before the end of (1), with deadline two steps after the beginning of (3) (and thus a window length of $1 + (2rn - 2n) + 2 = 2rn - 2n + 3$). If C is a vertex cover, then at most one of u, v is not in C, and that page can be served in the last time step of (1). The one or two remaining requests among $\{u, v\}$ can be satisfied in the first two time steps of (3), thus meeting the deadline. If C is not a vertex cover, then there is an edge $\{u, v\}$ both of whose endpoints are not in C. For that edge $\{u, v\}$, one of the two requests u, v can be satisfied immediately, but the other request cannot be served before the beginning of (1) in the next repetition, thus incurring a response time of at least $1 + (2rn - 2n) + k + (2rn - n) + 1 \geq 2(2rn - 2n + 3)$ (where we assume $n \geq 4$, without loss of generality), again forcing $\alpha \geq 2$. □

5 Analysis of FIFO

Consider the online problem of minimizing the maximum response time for a schedule, given a set of requests. Recall that the $FIFO$ algorithm schedules pages in the order that they are requested. If requests for multiple distinct pages arrive at the same time, $FIFO$ schedules these pages in arbitrary order.

We will prove that $FIFO$ is 2-competitive and that no deterministic online scheme can do better.

We proceed first by defining a few concepts. The queue at time t of the schedule produced by algorithm ALG, denoted as A_t, shall refer to the set of pages for which outstanding requests exist at time t.

THEOREM 5.1. *For any constant $\epsilon > 0$, there exists no $(2 - \epsilon)$-competitive algorithm for the online model of minimizing the maximum response time.*

Proof. Consider the following example. Let the client request pages $1, 2, \ldots, n$ at time $t = 0$, and then request whatever the online algorithm broadcasts immediately after that page is broadcast. This is done for a total of $n - 1$ steps. At time $t = n$, we see that OPT has nothing in its queue while the online algorithm has $n-1$ outstanding requests. The construction in [7] stops here, and as is evident the maximum response time is n for both FIFO, and the optimal solution. Then, suppose that at time $t = n$, the client requests a set of n new pages, say pages $n + 1, \ldots, 2n$. OPT can schedule these items in the next n timeslots, but the online algorithm now has n new pending requests in addition to the $n - 1$ pages that were already in its queue. Even in the best case where the online algorithm continues to schedule what was originally in its queue, the requests for the new n items will not begin to be satisfied until $n - 1$ time units after the requests were made. Therefore, the last page scheduled by the online algorithm would have waited a total of $2n-1$ time slots, while OPT would have scheduled pages $n + 1, \ldots, 2n$ within a maximum response time of n. The competitive ratio, $\frac{2n-1}{n}$, approaches 2 as n increases. □

THEOREM 5.2. *FIFO is a 2-competitive online algorithm.*

Proof. Since the particular advantage of broadcast scheduling over unicast scheduling is the ability to satisfy more than one request at a time, we shall refer to such occasions as the "merging" of requests. In other words, the merging of two requests (p, i) and (p, j), for $i \neq j$, refers to the occasion where $S_i^p = S_j^p$.

Consider the difference between $FIFO$'s queue and OPT's queue at time t, denoted as $|F_t \backslash O_t|$. If, at all times t, $|F_t \backslash O_t|$ is upper bounded by OPT, then the claim follows easily, since $|F_t \cap O_t| \leq |O_t| \leq OPT$. Suppose that $|F_t \backslash O_t|$ first exceeds OPT at time t^*. We derive the following contradiction: if $|F_{t^*} \backslash O_{t^*}| > OPT$, then $|F_{t^*-\delta} \backslash O_{t^*-\delta}| > OPT$ for some earlier time $t^* - \delta$.

Let P be the set of pages in $F_{t^*} \backslash O_{t^*}$. In general, a page p_i may have several outstanding requests at a time t. Since their response times cannot exceed that of the earliest outstanding request for p_i, these subsequent requests can be ignored without changing the performance of the schedule. Thus, without loss of generality, we can assume that there is exactly one outstanding request at time t^* for each page in P. Let the pages of P be ordered by the time of their request, i.e. $\{(p_1, t_1), \ldots, (p_k, t_k)\}$ such that $t_1 \leq t_2 \leq \ldots \leq t_k$.

Let $t^* - \delta$ be the time that the request for page p_1 arrives, and define interval W as the interval $[t^* - \delta, t^*]$. Note that $\delta \geq k > OPT$, since OPT satisfies all k requests for the distinct p_i's in interval W, but may schedule other pages during that time also. At the same time, $FIFO$ satisfies no requests of P during interval W since these requests are still in $FIFO$'s queue at time t^*. Let the set of pages scheduled by $FIFO$ during interval W be $Q = \{q_1, \ldots, q_\delta\}$, ordered by the time in which they were scheduled. All of these pages were in $FIFO$'s queue at time $t^* - \delta$ since we know that $FIFO$ never scheduled p_1 in interval W. Therefore, all pages in Q must be distinct.

We want to prove that at most $(\delta - k)$ of these pages are also in OPT's queue at time $t^* - \delta$, i.e., OPT can schedule at most $(\delta - k)$ of these pages in interval W. (Note that given the length of window W, OPT cannot satisfy requests for the pages of Q after time t^*, as this would increase the maximum response time beyond OPT.) If OPT does not merge requests from set Q and set P, then it follows that OPT can schedule at most $(\delta - k)$ pages of Q in interval W and must schedule the rest before time $t^* - \delta$. Then, $|F_{t^*-\delta} \backslash O_{t^*-\delta}| \geq k > OPT$. Lemma 5.1, which we prove below, shows that OPT does not merge requests from Q and P, concluding the proof. □

LEMMA 5.1. *OPT does not merge outstanding requests from sets P and Q.*

Proof. Suppose OPT merges outstanding requests from set Q and set P in the interval W. Consider the merge, for maximum i (see Fig. 5), between the request(s) satisfied by $FIFO$ in broadcasting page q_i and the request not satisfied by $FIFO$ in interval W for page p_ℓ (scheduled at some time $t^* - \delta + j$, $j > i$). Let Q' be the set of pages $\{q_{i+1}, \ldots, q_\delta\}$. By assumption, no pages of Q' can be merged with pages of P by OPT. Therefore, OPT can schedule at most $(\delta - k)$ pages of Q' in interval W since OPT is busy scheduling the pages of P during the other k time slots. Then, at least $(\delta - i) - (\delta - k) = (k - i)$ pages of Q' were scheduled by OPT before interval W. Because of the nature of $FIFO$, everything in Q' was scheduled after page q_i was requested, so the wait in OPT's schedule for q_i is at least $k - i + j > k > OPT$, a contradiction. □

6 LP Gap Example

In this section we refer to the IP formulation from [15]. The binary variable $y_{t'}^p = 1$ iff page p is broadcast at time t'. The binary variable $x_t^p = 1$ iff request (p, t) is satisfied at some time $t', t < t' \leq D_t^p$. The first set

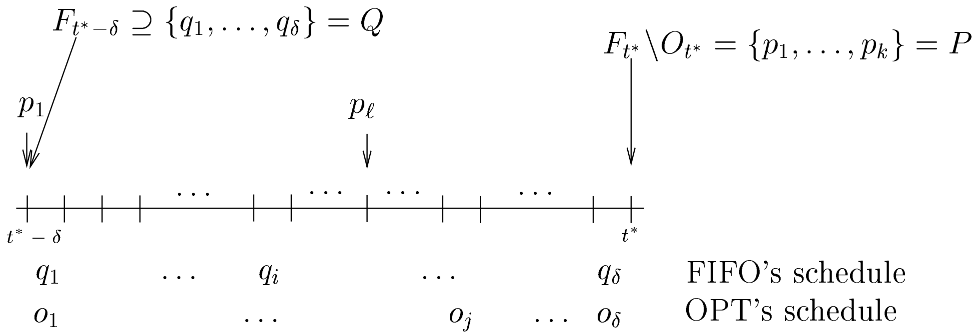

Figure 5: Proof of Lemma 5.1.

of constraints ensure that whenever a request (p, t) is satisfied, page p is broadcast at $t', t < t' \leq D_t^p$. The second set of constraints ensure that at most one page is broadcast at any given time. The last two constraints ensure that the variables assume integral values. By letting the domain of x_t^p and $y_{t'}^p$ be $0 \leq x_t^p, y_{t'}^p \leq 1$, we obtain the LP relaxation for the problem.

$$\text{Maximize} \sum_{(p,t)} r_t^p \cdot x_t^p$$

$$(6.1) \qquad \sum_{t'=t+1}^{D_t^p} y_{t'}^p - x_t^p \geq 0 \qquad \forall p, t$$

$$\sum_p y_{t'}^p \leq 1 \qquad \forall t'$$

We now consider the following instance with three pages A, B, C. Each page is requested at each time slot $t = 1 \ldots T$. The window length for each request for A is 2, the window length of each request for B is 3 and the window length for each request for C is 6. By broadcasting $1/2$ unit of page A at each timeslot, $1/3$ unit of page B and $1/6$ unit of page C, we obtain a fractional solution where all requests are satisfied and 1 unit is transmitted in each time slot. We now show that the best integral solution can only schedule at most 17 out of 18 requests. First of all, it is easy to see that there is no schedule that satisfies all the requests. Suppose such a schedule exists. If we schedule C, then we must schedule an A in the time slots before and after it, since we cannot have two adjacent slots without an A. Now we have a consecutive block of 3 pages without a B, thus at least one B will get dropped.

If a schedule broadcasts α pages of C, at most 6α requests for C can be satisfied. At the same time, for every page C scheduled, at least one request for page A or B is not satisfied. This is true, even for pages C scheduled close to each other.

Consider a time interval of length $6T$. A schedule that broadcasts fewer than T pages for C, say $T - \beta$ pages, will lose at least $T + 5\beta > T$ requests. (The schedule can satisfy at most $6(T - \beta)$ requests for C, and so will lose at least 6β requests for C, in addition to the $T - \beta$ requests lost for either A or B.) On the other hand, a schedule that broadcasts more than T pages of C will lose more than T requests for A or B. The schedule that broadcasts exactly T pages of C loses at least T of the $18T$ requests.

7 Conclusion

We close several open problems in the area of broadcast scheduling – both the problem of minimizing the maximum response time, as well as the problem of maximizing throughput. It is possible that there is a better offline algorithm for minimizing the maximum response time, namely one with an approximation factor better than 2.

Acknowledgments. We would like to thank Moses Charikar, Sudipto Guha, Seffi Naor and Kirk Pruhs for useful discussions.

References

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

[2] D. Aksoy, and M. Franklin. RxW: A scheduling approach for large-scale on-demand data broadcast. In *IEEE/ACM Transactions On Networking*, Volume 7, Number 6, 846-860, 1999.

[3] N. Bansal, M. Charikar, S. Khanna, and J. Naor. Approximating the average response time in broadcast scheduling. In *Proceedings of 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, 215–221, 2005.

[4] N. Bansal, D. Coppersmith, and M. Sviridenko. Improved approximation algorithms for broadcast scheduling. In *Proceedings of 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, 344–353, 2006.

[5] A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieber. Minimizing service and operation costs of periodic scheduling. In *Proceedings of 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 11-20, 1998.

[6] A. Bar-Noy, S. Guha, Y. Katz, J. Naor, B. Schieber and H. Schachnai. Throughput Maximization of Real-Time Scheduling with Batching, in *Proceedings of 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 742-751, 2002.

[7] Y. Bartal and S. Muthukrishnan. Minimizing maximum response time in scheduling broadcasts. *Proceedings of 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, 558-559, 2000.

[8] W. Chan, T. Lam, H. Ting, and P. Wong. New results on on-demand broadcasting with deadline via job scheduling with cancellation. In *10th COCOON*, LNCS 3106, Springer-Verlag, 210-218, 2004.

[9] M. Charikar, S. Khuller. A robust maximum completion time measure for scheduling *Proceedings of 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 324–333, 2006.

[10] M. Chrobak, C. Durr, W. Jawor, L. Kowalik, M. Kurowski. A note on scheduling equal-length jobs to maximize throughput. *Journal of Scheduling*, Vol 9(1):71–73 (2006).

[11] J. Edmonds and K. Pruhs. Multicast pull scheduling: when fairness is fine. In *Proc. of 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 421-430, 2002.

[12] J. Edmonds and K. Pruhs. A maiden analysis of longest wait first. In *Proc. of 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, 811-820, 2004.

[13] T. Erlebach, A. Hall. NP-Hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow. In *Proc. of 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 194-202, 2002.

[14] R. Gandhi, S. Khuller, Y. Kim, and Y.C. Wan. Algorithms for minimizing response time in broadcast scheduling. In *Proc. Ninth Conference on Integer Programming and Combinatorial Optimization* (May 2002), vol. 2337 of *Lecture Notes in Computer Science*, Springer, pp. 415–424.

[15] R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan. Dependent rounding in bipartite graphs. In *Proc. IEEE Symposium on Foundations of Computer Science* (2002), pp. 323–332.

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

[17] B. Kalyanasundaram, K. Pruhs, and M. Velauthapillai. Scheduling broadcasts in wireless networks. In *European Symposium of Algorithms*, LNCS 1879, Springer-Verlag, 290-301, 2000.

[18] S. Khuller and Y. Kim. Equivalence of two linear programming relaxations for broadcast scheduling, *Operations Research Letters*, Vol 32 (5): 473–478 (2004).

[19] J. Kim and K. Chwa. Scheduling broadcasts with deadlines. *Theoretical Computer Science*, Vol 325(3):479–488 (2004).

[20] J. Wong. Broadcast Delivery. In *Proc. of the IEEE*, 76(12):1566-1577, 1988.

[21] F. Zheng, S. Fung, W. Chan, F. Chin, C. Poon and P. Wong. Improved online broadcast scheduling with deadlines. *Proc. of the 11th International Computing and Combinatorics Conference (COCOON)*, pp. 320–329, (2006).