On-Demand Broadcasting Under Deadline

Bala Kalyanasundaram* and Mahe Velauthapillai**

Georgetown University
{kalyan,mahe}@cs.georgetown.edu

Abstract. In broadcast scheduling multiple users requesting the same information can be satisfied with one single broadcast. In this paper we study preemptive on-demand broadcast scheduling with deadlines on a single broadcast channel. We will show that the upper bound results in traditional real-time scheduling does not hold under broadcast scheduling model. We present two easy to implement online algorithms BCast and its variant BCast2. Under the assumption the requests are approximately of equal length (say k), we show that BCast is $O(k)$ competitive. We establish that this bound is tight by showing that every online algorithm is $\Omega(k)$ competitive even if all requests are of same length k. We then consider the case where the laxity of each request is proportional to its length. We show that BCast is constant competitive if all requests are approximately of equal length. We then establish that BCast2 is constant competitive for requests with arbitrary length. We also believe that a combinatorial lemma that we use to derive the bounds can be useful in other scheduling system where the deadlines are often changing (or advanced).

1 Introduction

On demand pay-per-view services have been on the increase ever since they were first introduced. In this model, there is a collection of documents such as news, sports, movies, etc., for the users to view. Typically, broadcasts of such documents are scheduled ahead of time and the users are forced to choose one of these predetermined times. Moreover, the collection of documents broadcasted on such regular basis tend to be small. Even though the collection could change dynamically (but slowly), this collection is considered to be the collection of "hot" documents by the server. Recently many companies, for example *TIVO, REAL, YESTV* have introduced true on-demand services where a user dynamically makes a request for a document from a large set of documents. This has the advantage of dealing with larger set of documents and possibly satisfying the true demand of the users. Generally, the service provider satisfies the request (if possible) for each user by transmitting the document independently for each user. This leads to severe inefficiencies since the service provider may repeat

* Supported in part by NSF under grant CCR-0098271, Airforce Grant, AFOSR F49620-02-1-0100 and Craves Family Professorship funds.
** Supported in part by a gift from AT&T and McBride Endowed Chair funds.

G. Di Battista and U. Zwick (Eds.): ESA 2003, LNCS 2832, pp. 313–324, 2003.

the same transmission many times. Broadcasting has the advantage of satisfying many users with the same request with one broadcast [1,3,6,9]. But, shifting from transmitting at a fixed time or regular intervals to true on-demand broadcasting has a major disadvantage. A user does not know whether the request will be satisfied or not and may experience a long wait. Even if we minimize the average response time (see [4]) for the user, unpredictability of the response time may be completely unacceptable for many users. It would be appropriate if the user assigns a deadline after which the completion of the request bears no value to the user.

In this paper we study preemptive on-demand broadcasting with deadline on a single broadcast channel. We associate an arrival time, a requested document, a deadline and a profit with each request. The system receives requests at the arrival time and knows nothing regarding future demands when it decides to broadcast a piece of a document. Whenever a request is satisfied on or before its deadline, the system earns the profit specified by the request. Otherwise, the system does not earn any profit from the request. This is often referred to as *soft deadline*. Our goal is to maximize the overall profit of the system.

First we consider the case where all the documents are approximately equal in length, which we call the $O(1)$-*length condition*. This is motivated by the fact that most of the documents (e.g., movies) are about the same length. We present an easy to implement online algorithm which we call *BCast*. Then we prove that this algorithm is $O(k)$ competitive, where k is the length of the longest request. We also show that this result is tight by showing that every online algorithm is $\Omega(k)$ competitive. We then answer the following question: Under what condition can we find a constant competitive algorithm for this problem? We prove that BCast is constant competitive if laxity of each request is proportional to the length of the requested document (i.e., laxity assumption) and all documents are approximately of same length (i.e., length assumption). We then consider the case where the lengths of the requested documents differ arbitrarily. Does there exist an online algorithm with constant competitive ratio for this case? We answer the question by modifying BCast to handle arbitrary lengths. We prove that the modified algorithm, we call it BCast2, is constant competitive under *laxity assumption*. We also compare and contrast pervious results in real-time scheduling with deadline [1,12].

1.1 Definitions and Model

We assume that users request for document from a collection $\{m_1, m_2, \ldots\}$. This collection could be dynamically changing since our upper bounds are independent of the number of documents in the collection. A **document** m_i has ℓ_i indivisible or non-preemptable segments or chapters. We say that ℓ_i is the length of the document m_i and it does not vary over time. We assume that segments are approximately identical in size so that exactly one segment of any document can be broadcasted at a time on a single channel.

With respect to any document, we assume that the broadcast schedule is cyclical in nature. That is, if a document has 4 segments, (namely 1,2,3 and 4) then the ith broadcast of the document will be segment $(i-1)\ mod\ 4+1$.

We assume that users request only entire documents. The *length of the request* is nothing but the length of the requested document. Moreover, users can assemble a document m_i if they receive all of the ℓ_i segments in any of the ℓ_i cyclical orders. Further, schedule on a single channel may choose different documents on consecutive time units as long as cyclical schedule is maintained with respect to each document. It is not hard to establish that noncyclic broadcast does not benefit the system if partial document is of no use to the individual users. See [1] for more details about on-demand broadcasting with deadlines.

In this paper, we deal with single channel broadcast scheduling. But, when we establish lower bounds, we show that even multiple channels or multiple broadcast per unit time does not provide significant benefit to the online algorithm. In order to establish such lower bound results, we introduce the following definitions.

We say that an algorithm is *s-speed* algorithm, if the algorithm is allowed to schedule s broadcasts for each time unit. For $s > 1$, more than one broadcast of a document at any time is possible.

We say that an algorithm is *m-channel* algorithm, if the algorithm is allowed to schedule broadcasts of m different documents at each time. Multiple broadcast of the same document is not allowed at any time.

Finally, we give a natural extension (to broadcast scheduling) of two standard algorithms from traditional real-time scheduling. Ties are broken arbitraly.

Earliest Deadline First (**EDF**): At each broadcasting step, among all documents, **EDF** selects the one that has a pending satisfiable request with earliest deadline.

Least Laxity First (**LLF**): At each broadcasting step, among all documents, **LLF** selects the one that has a pending satisfiable request with least laxity.

The problem we consider in this paper is online in nature. The request for documents are presented to the system at the arrival time. A request R_i is a four tuple $(r_i, d_i, m_{z(i)}, p_i)$ which consists of an arrival time r_i, a deadline d_i, a requesting document $m_{z(i)}$ and payment p_i. The length of the request is $\ell_{z(i)}$. The use of $z(i)$ is to indicate that the request R_i does not always deal with document i.

The deadline specified in a request is a *soft deadline*. It means that the system gets paid p_i if the request is satisfied by the deadline d_i. But failure to satisfy R_i by its deadline does not bring any catastrophic consequence other than the loss of potential pay p_i to the system. Our objective is to maximize the revenue for the system.

Suppose I be the input given to *s-speed* online algorithm A. Let $C \subseteq I$ be the set of inputs satisfied by A by their deadline. We use the notation $A_s(I)$ to denote the $\sum_{R_i \in C} p_i$, the total profit earned by *s-speed* algorithm A on input I. We also use the notation $\mathrm{OPT}(I)$ to denote the maximum profit that an offline optimal *1-speed* algorithm can earn.

An algorithm A is said to be a *s-speed c-approximation* algorithm if

$$\max_{\text{Inputs } I} \frac{A_s(I)}{\text{OPT}(I)} \leq c.$$

An algorithm A is said to be *c-competitive*, or said to have *competitive ratio* c, if A is a 1-speed c-approximation algorithm.

Request Pay-off Density Δ_i: This quantity for a request $R_i=(r_i, d_i, m_{z(i)}, p_i)$ is denoted by Δ_i and is defined to be $p_i/\ell_{z(i)}$.
For constants $\epsilon > 0$ and $c \geq 1$, we say that a set of requests I and the set of documents $\{m_1, m_2, \ldots\}$, satisfy
a. **ϵ-laxity condition**, if for all requests $R_i \in I$, $d_i - r_i \geq (1+\epsilon)\ell_{z(i)}$.
b. **c-length condition** if for all pairs of documents m_i and m_j, we have $\ell_i/\ell_j \leq c$.

The following two definitions are based on the **online algorithm** and the set of requests I under consideration. For ease of notation we will not indicate the online algorithm under consideration in the notation. It will be very clear from the context, since we only consider two different online algorithms and they are in two different sections.

Set of Live Requests $L_i(t)$: A request $R_i=(r_i, d_i, m_{z(i)}, p_i)$ is live at time t if the request has not been completed at time t and has a chance of being completed if the algorithm were to broadcast $m_{z(i)}$ exclusively from time t until its deadline. That is, $(d_i - t) \geq (\ell_{z(i)} - b)$ where $b \geq 0$ is the number of broadcasts of document $m_{z(i)}$ during the interval $[r_i, t)$. Given I, let $L_j(t)$ be the set of live requests for the document m_j at time t.

Document Pay-off Density $M_i(t)$: It is the sum of all the pay-off densities of the live-request pending for the document at time t. $M_i(t) = \sum_{R_j \in L_i(t)} \Delta_j$

1.2 Previous Results and Our Results

Broadcast scheduling problem has been studied previously by [1,3,6,5,9,10]. Most of the results consider average response time for the users. In these papers, there is no deadline associated with each request. Every request is eventually satisfied. But, each user experiences a response time equal to time-of-completion minus time-of-request. First, we [9] showed that there is an offline 3-speed 3-approximation for this problem using LP-based techniques. Later Gandhi et.al [6,7] improved the bounds for this offline case. Recently, Edmonds et. al [4] developed O(1)-speed O(1)-approximation online algorithm for the average response time case. They proved it by showing how to convert online algorithm from traditional scheduling domain to broadcasting domain. Our paper differs fundamentally from all of the previous work in broadcast scheduling. Independant to our work, Kim et. al [11] obtained constant competitive algorithm for the broadcasting problem with deadline when O(1)-length condition is satisfied.

In section 2 we prove lower bound results. We first consider soft deadline case where the objective function is to maximize the overall profit. We prove that the competitive ratio of every deterministic online algorithm is $\Omega(k)$ (where k is

the length of the longest request) for the on-demand broadcasting problem with deadlines and preemption. Then we show that the competitive ratio does not improve significantly even if we allow m simultaneous broadcast of different documents at each time step for the online algorithm while offline optimal broadcasts only once. In this case we show a lower bound of $\Omega(k/m)$ on the competitive ratio.

Next we consider hard deadline case where we must satisfy each and every request. We consider only those set of requests I, such that there exists a schedule that broadcasts at most once each time, and satisfy all the requests in I. In the traditional single processor real-time scheduling, it is well known that LLF and EDF produces such schedule. For the single channel broadcast scheduling problem, we prove that even *s-speed* LLF and EDF algorithms do not satisfy every request even if 1-speed optimal satisfy all. Further, we show that there is no 1-speed online algorithm that can finish all the requests, even if 1-speed optimal satisfy all.

In section 3 we prove upper bound results. We do this by defining two algorithms BCast and BCast2. We first prove that BCast is $O(kc)$ competitive where k is the length of the longest request and c is the ratio of the length of the longest to the shortest request. As a corollary, if the set of documents satisfy $O(1)$-length condition, then BCast is $O(k)$ competitive. We then show that *BCast* is constant competitive if the set of requests and the set of documents satisfy both $O(1)$-length condition and $O(1)$-laxity condition. We then modify BCast, which we call BCast2, in order to relax the $O(1)$-length condition. We prove that BCast2 is $O(1)$ competitive if $O(1)$-laxity condition alone is satisfied. Due to page limitations proofs of many theorems and lemmas have been omitted.

2 Lower Bound Results

In this section we prove lower bound results on broadcast scheduling with deadlines. We also compare these lower bound results with some of the lower and upper bound results in traditional (non-broadcasting setup) real-time scheduling.

2.1 Soft Deadlines

Recall that there is a simple constant competitive algorithm for traditional real-time scheduling with soft deadlines if all jobs are approximately of the same length [8]. In contrast, we show that it is not the case in broadcast scheduling under soft deadline.

Theorem 1. *Suppose all the documents are of same length k. Then every deterministic online algorithm is $\Omega(k)$ competitive for the on-demand broadcasting problem with deadlines and preemption.*

Proof. (of Theorem 1) Let $k > 0$ and A be any deterministic online algorithm. The adversary uses $k+1$ different documents. The length of each document is k and the payoff for each request is 1. We will construct a sequence of requests such that A is able to complete only one request while the offline completes k requests. The proof proceeds in time steps. At time 0, $k+1$ requests for $k+1$ different documents arrive. That is, $0 \leq i \leq k$, $R_i = (0, k, m_i, 1)$. WLOG, A broadcasts m_0 during the interval $[0, 1]$. For time $1 \leq t \leq k-1$, let $A(t)$ be the document that A broadcasts during the interval $[t, t+1]$. Adversary then issues k requests for k different documents other than $A(t)$ at time t where each request has zero laxity. Since each request has zero laxity, A can complete only one request. Since there are $k+1$ different documents and A can switch broadcast at most k times during $[0, k]$, there is a document with k requests which the offline optimal satisfies. ■

In the proof of the above theorem, the offline optimal satisfied k requests out of $\Theta(k^2)$ possible requests and A satisfied one request. In the next section we will study the performance of some well known online algorithms assuming the offline algorithm must completely satisfy all the requests.

We now show that no online algorithm performs well even if online algorithm is allowed m broadcasts per unit time while offline optimal performs one broadcast per unit time.

Theorem 2. *Suppose all the documents are of same length k. For $m > 0$, every m-broadcast deterministic online algorithm is $\Omega(k/m)$ competitive for the on-demand broadcasting problem with deadlines and preemption.*

2.2 Hard Deadlines

In this subsection, we consider the input instance where offline optimal completes **all** the requests before their deadline. Recall that in the traditional single processor real-time scheduling, it is well known that LLF and EDF are optimal. However, we show that LLF and EDF perform poorly for broadcast scheduling even if we assume that they have s-speed broadcasting capabilities.

Theorem 3. *Let s be any positive integer. There exists a sequence of requests that is fully satisfied by the optimal (offline) algorithm, but not by* s-speed *EDF. There exists another sequence of requests that is fully satisfied by the optimal (offline) algorithm, but not by s-speed LLF.*

Recall that the proof of Theorem 1 uses $\Theta(k^2)$ requests where the optimal offline can finish $\Theta(k)$ requests to establish a lower bound for online algorithm. The following theorem shows that no online algorithm can correctly identify a schedule to satisfy each and every request if one such schedule exists.

Theorem 4. *Let A be any online algorithm. Then there exists a sequence of requests that is satisfied by the optimal (offline) algorithm, but not by A.*

3 Upper Bound

Before we describe our algorithms and their analysis, we give intuitive reasoning to the two assumptions (length and laxity) as well as their role in the analysis of the algorithm.

When an online algorithm schedules broadcasts, it is possible that a request is partially satisfied before its deadline is reached. Suppose each user is willing to pay proportional to the length of the document he/she receives. Let us call it partial pay-off. On the contrary, we are interested actual pay-off which occurs only when the request is fully satisfied. Obviously, partial pay-off is at least equal to actual pay-off.

Definition 1. *Let $0 < \alpha \leq 1$ be some constant. We say that an algorithm for the broadcast scheduling problem is α-greedy, if at any time the pay-off density of the chosen document of the algorithm is at least α times the pay-off density of any other document.*

Our algorithms are α-greedy for some α. Using this greedy property and applying $O(1)$-length property, we will argue that actual pay-off is at least a constant fraction of partial pay-off. Then applying $O(1)$-laxity property, we will argue that the partial pay-off defined above is at least a fraction of the pay-off received by the offline optimal.

3.1 Approximately Same Length Documents

In this subsection we assume that the length of the requests are approximately within a constant factor of each other, which we call $O(1)$-length condition. We first present a simple algorithm that we call *BCast*. We prove that the competitive ratio of this algorithm is $O(k)$ where k is the length of the longest request, thus matching the lower bound shown in Theorem 1. We then show that if in addition to $O(1)$-length condition $O(1)$-laxity condition is also satisfied then *BCast* is constant competitive.

BCast: At each time step, the algorithm broadcasts a chapter of a document. We will now describe what document the algorithm chooses at each time step. With respect to any particular document, the algorithm broadcasts chapters in the cyclical wrap-around fashion. In order to do so, the algorithm maintains the next chapter that it plans to transmit to continue the cyclical broadcast. The following description deals with the selection of document at each time step.

1. **At time** 0, choose the document m_i with the highest $M_i(0)$ (document pay-off density) to broadcast.
2. **At time** t
   a) Compute $M_i(t)$'s for each document and let m_j be the document with highest pay-off density $M_j(t)$.
   b) Let m_c be the last transmitted document. If $M_j(t) \geq 2M_c(t)$ then transmit m_j. Otherwise continue transmitting m_c.

End BCast

Observation 1 *BCast is $\frac{1}{2}$-greedy for the broadcast scheduling problem. On the negative side, it is quite possible that BCast never satisfy even a single request. This happens when there are infinitely many requests such that the pay-off density of some document is exponentially approaching infinity. So, we assume that the number of requests is finite.*

Definition 2. *1. For ease of notation, we use A to denote online algorithm BCast.*
2. Let $m_{A(t)}$ be the document transmitted by algorithm A (i.e., BCast) at time t. For ease of presentation, we abuse the notation and say that $A(t)$ is the document transmitted by A at time t.
3. Let t_0 be the starting time, $t_1, \ldots t_N$ be the times at which BCast changed documents for broadcast and t_{N+1} be the time at which BCast terminates.
4. For $0 \leq i \leq N-1$, let C_i be the set of all requests completed by BCast during the interval $[t_i, t_{i+1})$.
5. C_N be the set of all requests completed by BCast during the interval $[t_N, t_{N+1}]$.
6. $C = \cup_{i=0}^{N} C_i$.

Next we will proceed to show that the algorithm BCast is $O(k)$ competitive. First we prove some preliminary lemmas.

Lemma 1. *For any $0 \leq i \leq N$, $M_{A(t_i)}(t_i) \leq M_{A(t_i)}(t_{i+1}) + \sum_{R_j \in C_i} \Delta_j$.*

Lemma 2. *Let k be the length of the longest document. $\sum_{t \in [t_t, t_{i+1})} M_{A(t)}(t) \leq kM_{A(t_i)}(t_{i+1}) + \sum_{R_j \in C_i} p_j$.*

Lemma 3. $\sum_{i=0}^{N} M_{A(t_i)}(t_{i+1}) \leq 2 \sum_{R_j \in C} \Delta_j$.

Proof. (of Lemma 3)
We prove this by a point distribution argument. Whenever a request R_j is completed by BCast during the time interval $[t_i, t_{i+1})$, we will give $2\Delta_j$ points to R_j. Observe that total points that we gave is equal to the right hand side of the equation in the lemma. We will now partition the points using a redistribution scheme into $N+1$ partitions such that the ith partition receives at least $M_{A(t_i)}(t_{i+1})$. The lemma then follows.

All partitions initially have 0 points. Our distribution process has $N+1$ iterations where at the end of i iteration, $N+2-i$th partition will receive $2M_{A(t_{N+1-i})}(t_{N+2-i})$ points. During the $i+1$st iteration $N+2-i$th partition will donate $M_{A(t_{N+1-i})}(t_{N+2-i})$ points to $N+1-i$th partition. Also, $2\Delta_j$ points given each R_j completed during the interval $[t_{N+1-i}, tN+2-i]$ is also given to $N+1-i$th partition.

We argue that $N+1-i$th partition receives $2M_{A(t_{N-i})}(t_{N+1-i})$. At time t_{N+1-i}, BCast jumps to a new document. So, $2M_{A(t_{N-i})}(t_{N+1-i}) \leq M_{A(t_{N+1-i})}(t_{N+1-i})$. Apply lemma 1, we have $M_{A(t_{N+1-i})}(t_{N+1-i}) \leq \sum_{R_j \in C_{N+1-i}} \Delta_j + M_{A(t_{N+1-i})}(t_{N+2-i})$. Combining these two inequalities we get, $2M_{A(t_{N-i})}(t_{N+1-i}) \leq \sum_{R_j \in C_{N+1-i}} \Delta_j + M_{A(t_{N+1-i})}(t_{N+2-i})$. The result then follows. ∎

Lemma 4. *Let k be the maximum length of any request.* $\sum_{t=0}^{t_{N+1}} M_{A(t)}(t) \leq k \sum_{i=0}^{N} M_{A(t_i)}(t_{i+1}) + \sum_{R_j \in C} p_j$.

Lemma 5. *Let c be the constant representing the ratio of the length of longest document to the length of shortest document.* $\sum_{R_i \in C} p_i \geq \frac{1}{2c+1} \sum_{t=0}^{t_N} M_{A(t)}(t)$.

Proof. (of Lemma 5) By using Lemma 3 and Lemma 4 we get $\sum_{t=0}^{t_N} M_{A(t)}(t) \leq 2k \sum_{R_j \in C} \Delta_j + \sum_{R_j \in C} p_j$. That is, $\sum_{t=0}^{t_N} M_{A(t)}(t) \leq 2 \sum_{R_j \in C} k\Delta_j + \sum_{R_j \in C} p_j$. By definition of Δ_j $\sum_{t=0}^{t_N} M_{A(t)}(t) \leq 2 \sum_{R_j \in C} k(p_j/\ell_j) + \sum_{R_j \in C} p_j$. Since c is the ratio of the length of longest document to the length of shortest document, $\sum_{t=0}^{t_N} M_{A(t)}(t) \leq (2c+1) \sum_{R_j \in C} p_j$. ∎

Lemma 6. *Let C, OPT be the requests completed by BCast and offline optimal respectively. Then,* $2k \sum_{t=0}^{t_{N+1}} M_{A(t)}(t) \geq \sum_{R_j \in OPT} p_j - \sum_{R_j \in C} p_j$.

Proof. (of Lemma 6) For a moment imagine that offline optimal gets paid p_j/ℓ_j only for the first received chapter for each request $R_j \in OPT - C$. Let $FO(t)$ be the set of requests in OPT that receive their first broadcast at time t based on the schedule *opt*. Let $FOPT(t)$ be the sum of pay-off densities of the requests in $FO(t)$. Observe that $\sum_{t=0}^{t_{N+1}} FOPT(t) \geq \sum_{R_j \in (OPT-C)} \Delta_j$ and $\sum_{t=0}^{t_{N+1}} M_{A(t)}(t) \geq 1/2 \sum_{t=0}^{t_{N+1}} FOPT(t)$. Combining the above two inequalities, $\sum_{t=0}^{t_{N+1}} M_{A(t)}(t) \geq 1/2 \sum_{R_j \in (OPT-C)} \Delta_j$. Multiplying by k and expanding the right hand side we get, $2k \sum_{t=0}^{t_{N+1}} M_{A(t)}(t) \geq \sum_{R_j \in OPT} p_j - \sum_{R_j \in C} p_j$. ∎

Theorem 5. *Algorithm BCast is $O(kc)$ competitive where k is the length of the longest request and c is the ratio of the length of the longest to the shortest document.*

Proof. (of Theorem 5) From Lemma 5 $2k(2c+1) \sum_{R_i \in C} p_i \geq 2k \sum_{t=0}^{t_{N+1}} M_{A(t)}(t)$. From Lemma 6, $2k(2c+1) \sum_{R_i \in C} p_i \geq \sum_{R_i \in OPT} p_i - \sum_{R_i \in C} p_i$. Simplyfying, $[2k(2c+1)+1] \sum_{R_i \in C} p_i \geq \sum_{R_i \in OPT} p_i$. ∎

Corollary 1. *BCast is $O(k)$ competitive if requests are approximately same length.*

Next we will prove that the BCast algorithm is $O(1)$ competitive if the laxity is proportional to length.

For ease of presentation, we use the notation *opt* to represent the offline optimal algorithm and OPT be the set of requests satisfied by *opt*. First, we prove a key lemma that we use to derive upper bounds for two algorithms. Intuitively, each request in OPT is reduced in length to a small fraction of its original length. After reducing the length of each request, we advance the deadline of each request R_i to some time before $d_i - (1+\eta)\ell_{z(i)}$). We then show that there exists a pair of schedules S_1 and S_2 such that their union satisfy these reduced requests before their new deadline. Since a fraction of each request in OPT is scheduled, the partial pay-off is proportional to the total pay-off earned by the offline optimal schedule. We then argue that our greedy algorithm does better than both S_1 and S_2. We think that this lemma may have applications in other areas of scheduling where one deals with sudden changes in deadlines.

Lemma 7. *Suppose $\delta = 2\epsilon/9$ and $\epsilon < 1/2$. Under ϵ-laxity assumption, there exists two schedules S_1 and S_2 such that the following property holds:*
For all $R_i \in OPT$, the number of broadcasts of document m_i in both S_1 and S_2 during the interval $[r_i, d_i - (1 + \delta + \epsilon/2)\ell_i]$ is at least $\delta\ell_i$.

In the following lemma, we establish the fact that the partial pay-off for A (i.e., *BCast*) is at least a constant fraction of the pay-off earned by offline optimal algorithm *opt* when $O(1)$-laxity condition is met.

Lemma 8. *Under the ϵ-laxity assumption and for some $\gamma > 0$ the following holds. $\sum_t M_{A(t)}(t) \geq \gamma \sum_{R_i \in OPT} p_i$.*

Theorem 6. *Under both $O(1)$- length and ϵ-laxity conditions, the algorithm BCast is $O(1)$ competitive.*

3.2 Arbitrary Length Documents

In this subsection, we consider the case where the length of the document vary arbitrarily. However, we continue to assume that ϵ-laxity condition is satisfied.

We will present a modified online algorithm, which we call BCast2, and prove that it is $O(1)$ competitive under ϵ-laxity condition.

Before we proceed to modify *BCast*, we point out the mistake that *BCast* makes while dealing with arbitrary length documents. When *BCast* jumps from one document (say m_i) to another (say m_j) at time t, it does so based only on the density of the documents and bluntly ignores their length. At time t, we have $M_j(t) \geq 2M_i(t)$. But at time $t+1$, it could be the case that $M_j(t+1)$ gone down to a level such that $M_j(t+1)$ is just greater than $\frac{1}{2}M_i(t+1)$. However, this does not trigger the algorithm to switch back to document m_i from m_j. As a consequence, for long documents such as m_i, we will accumulate lots of partially completed requests and thus foil our attempt to show that the total cost earned by completing requests is not a constant fraction of partial pay-off (i.e., accumulated pay-off if even partially completed requests pay according to the percentage of completion).

In order to take care of this situation, our new algorithm *BCast2* maintains a stack of previously transmitted document. Now switching from one document to another is based on the result of checking two conditions. First, make sure that the density of the document on top of the stack is still a small fraction of the density of the transmitting document. This is called *condition 1* in the algorithm. Second, make sure that there is no other document with very high density. This is called *condition 2* in the algorithm. If any one or both these conditions are violated then the algorithm will switch to a new document to broadcast. In order to make this idea clear (and make it work), we introduce two additional labeling on the requests. As before, these definitions critically depends on the algorithm under consideration.

Startable Request: We say that a request R_i is startable at time t, if the algorithm has not broadcasted document m_i during $[r_i, t]$ and $r_i \leq t \leq d_i - (1 + \epsilon/2)\ell_i$.

Started Request: We say that a request R_i is started at time t if it is live at time t and broadcast of document m_i took place in the interval $[r_i, d_i-(1+\epsilon/2)\ell_i]$. Observe that the document pay-off density is redefined to be based on the union of started and startable requests as opposed to live requests.

$M_k(t)$ denote the sum of the densities of the started or startable request at time t for the document m_k.

$SM_k(t)$ denote the sum of the densities of the started request at time t for the document m_k.

TM_k denote the density of document m_k at the time of entry into the stack (T stands for the threshold value). As long as the document m_k stays on the stack, this value TM_k does not change.

BCast2 is executed by the service providers. Assume the service provider has n distinct documents. The algorithm maintains a stack; each item in the stack has the following two information:

1) Document name say m_k.

2) Started density value $SM_k(t)$ of the document at the time t it goes on the stack. We refer it TM_k for document m_k and it is time independent.

Initially stack is empty.

BCast2: c_1 and α are some positive constants that we will fix later.

1. At time $t = 0$ choose the document with the highest M_i (document pay-off density) and transmit:
2. **For** $t = 1, 2 \ldots$
   a) Let m_j be the document with the highest $M_j(t)$ value.
   b) Let m_k be the document on top of the stack (m_k is undefined if the stack is empty).
   c) Let m_i be the document that was broadcast in the previous time step.
   d) **While** $((SM_k(t) \leq \frac{1}{2}TM_k)$ and Stack Not Empty)
   e) pop stack.
   f) Now m_k be the document on top of stack.
   g) Condition 1. $SM_k(t) \geq \frac{c_1\epsilon}{(1+\alpha)}M_i(t)$
   h) Condition 2. $M_j(t) \geq \frac{2(1+\alpha)}{c_1\epsilon}M_i(t)$
   i) If both conditions are false continue broadcasting document m_i.
   j) If condition (1) is true then broadcast m_k, pop m_k from the stack (do not push m_i on the stack).
   k) If condition (2) is true the push m_i on the stack along with the value $M_i(t)$ (which is denoted by TM_i), broadcast m_j.
   l) If both conditions are true then choose m_j to broadcast only if $M_j(t) \geq \frac{2(1+\alpha)}{c_1\alpha}M_k(t)$. Otherwise broadcast m_k, pop m_k from the stack (do not push m_i on the stack). We will later establish the fact that both conditions 1 and 2 are false for the current choice of broadcast m_k.
3. **EndFor**

End BCast2

For ease of presentation we overload the term $BCast2$ to represent the set of all requests completed by BCast2 before their deadline. As before, we use A to denote algorithm BCast2 in our notation. Without the $O(1)$-length condition, we will now establish the fact that the total pay-off for completed requests for BCast2 is proportional to the partial pay-off where every request pays proportional to the percentage of completion.

Lemma 9. *For* $c_1 \leq \frac{3}{32}$, $\sum_{Rj \in BCast2} bj \geq \frac{\alpha}{2(1+\alpha)} \sum_t M_{A(t)}(t)$

Theorem 7. *Assuming* ϵ*-laxity condition, BCast2 is constant competitive algorithm for the broadcast scheduling problem.*

References

1. A. Aacharya and S. Muthukrishnan. Scheduling On-demand Broadcasts: New Metrics and Algorithms. In *MobiCom*, 1998.
2. A. Bar-Noy, S. Guha, y. Katz, and J. Naor. Throughput maximization of real-time scheduling with batching. In *Proceedings of ACM/SIAM Symposium on Discrete Algorithms*, January 2002.
3. Y. Bartal and S. Muthukrishnan. Minimizing Maximum Response Time in Scheduling Broadcasts. In *SODA*, pages 558–559, 2000.
4. J. Edmonds and K. Pruhs. Multicast pull scheduling: When fairness is fine. In *Proceedings of ACM/SIAM Symposium on Discrete Algorithms*, January 2002.
5. T. Erlebach and A. Hall. Np-hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow. In *Proceedings of ACM/SIAM Symposium on Discrete Algorithms*, January 2002.
6. R. Gandhi, S. Khuller, Y. Kim, and Y-C Wan. Approximation algorithms for broadcast scheduling. In *Proceedings of Conference on Integer Programming and Combinatorial Optimization*, 2002.
7. R. Gandhi, S. Khuller, S. Parthasarathy, and S. Srinivasan. Dependent rounding in bipartite graphs. *IEEE Symposium on Foundations of Computer Science*, 2002.
8. B. Kalyanasundaram and K.R. Pruhs. Speed is as Powerful as Clairvoyance. *IEEE Symposium on Foundation of Computation*, pages 214–221, 1995.
9. B. Kalyanasundaram, K.R. Pruhs, and M. Velauthapillai. Scheduling Broadcasts in Wireless Networks. *Journal of Scheduling*, 4:339–354, 2001.
10. C. Kenyon, N. Schabanel, and N Young. Polynomial-time approximation schemes for data broadcasts. In *Proceedings of Symposium on Theory of Computing*, pages 659–666, 2000.
11. J. H. Kim and K. Y. Chowa. Scheduling broadcasts with deadlines. In *COCOON*, 2003.
12. C. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. In *ACM Symposium on Theory of Computing*, pages 140–149, 1997.