

# A Comparison of Multicast Pull Models

Kirk R. Pruhs\*

Patchrawat Uthaisombut†

## Abstract

We consider the setting of a web server that receives requests for documents from clients, and returns the requested documents over a multicast/broadcast channel. We compare the quality of service (QoS) obtainable by optimal schedules under various models of the capabilities of the server and the clients to send and receive segments of a document out of order. We show that allowing the server to send segments out of order does not improve any reasonable QoS measure. However, the ability of the clients to receive data out of order can drastically improve the achievable QoS under some, but not all, reasonable/common QoS measures.

## 1 Introduction

We consider the setting of a web server that receives requests for documents from clients, and returns the requested documents over a multicast/broadcast channel. Such a setting might arise for several reasons. The server and the client may be on the same local area network, or wireless network, where broadcasting is the basic form of communication. Or the the server may be using multicast communication to provide better scalability. For example, to provide scalable data service the Hughes DirecPC system[5] uses a broadcast satellite system and Digital Fountain [4] uses IP-multicast. If multiple clients request the same document at approximately the same time, the server may aggregate these requests, and multicast the document only once. One would expect that the ability to aggregate requests would improve Quality of Service (QoS) for the same reason that proxy caches improve QoS, that is, it is common for different users to make requests to the same document.

We formalize this problem in the following manner. (There is a concrete example instance in next subsection that illustrates some of these concepts.) Request  $R_i$ ,  $1 \leq i \leq n$ , is for document  $D_{\sigma(i)} = D_j$ ,  $1 \leq j \leq m$ . We assume that at some level, say at the transport layer, the document  $D_j$  is partitioned into  $p_j$  equal sized segments. We assume that the transport layer can transmit one segment from the server to all the clients in one unit of time. We assume that request  $R_i$  is made by a client, and immediately communicated to the server, at some time  $r_i$ . The completion time  $C_i(S)$

---

\*Computer Science Department, University of Pittsburgh. Email: [kirk@cs.pitt.edu](mailto:kirk@cs.pitt.edu). Homepage: <http://www.cs.pitt.edu/~kirk/> Supported in part by NSF grant CCR-0098752, NSF grant ANIR-0123705, NSF grant ANI-0325353, and a grant from the United States Air Force.

†Computer Science Department, University of Pittsburgh. Email: [utp@cs.pitt.edu](mailto:utp@cs.pitt.edu). Homepage: <http://www.cs.pitt.edu/~utp/>

for this request in a schedule  $S$  is the earliest time that all of the segments of  $D_j$  have arrived at the requesting client after time  $r_i$ .

There are two standard QoS measures for request  $R_i$  in a schedule  $S$ . The *flow time*, or equivalently user perceived latency, for this request is  $F_i(S) = C_i(S) - r_i$ . The *stretch* for this request is  $ST_i(S) = F_i(S)/p_j$ . The motivation for considering stretch is that a human user may have some feeling for the size of the requested document (for example the user may know that video documents are generally larger than text documents) and may be willing to tolerate a larger response time for larger documents. Usually the QoS measures for the requests are combined using an  $l_p$  norm to get a QoS measure for the collection of requests. The  $l_p$  norm of the flow times for a particular schedule  $S$  is  $F^{(p)}(S) = (\sum_{i=1}^n F_i^p)^{1/p}$ , where  $1 \leq p \leq \infty$ . The most commonly used measure of system performance is  $F^{(1)}(S)/n$ , the average flow time. Also of particular interest is  $F^{(\infty)}(S)$ , the maximum flow time. The  $l_p$  norms for stretch, the average/total stretch, and maximum stretch are defined similarly, and are denoted by  $ST^{(p)}(S)$ ,  $ST^{(1)}(S)$ ,  $ST^{(\infty)}(S)$ , respectively. When the schedule  $S$  is understood we will drop it from our notation.

In the multicast pull setting, a new client might request a document in the middle of the server's transmission of that document to some other clients. Thus potentially this new client could buffer the end of the document at the transport layer. For example, TCP buffers out of order segments. In this case, after finishing the initial transmission of the document, the server need only retransmit the start of the document to satisfy the request of the new client. It is obvious that such client-side buffering can improve the QoS provided by the system.

However, this client buffering of the document may be difficult or impossible for the client to accomplish. One example is the Go-Back-N recovery method that is implemented in the HDLC protocol [12]; In Go-Back-N the receiver discards out of order packets in order to simplify its responsibilities. In the HTTP protocol the client could not generally buffer the end of a broadcast because the identification of the content of the broadcast is in the header. We assume that the client-side transport layer guarantees in order delivery of segments to the application layer. When the client-side transport layer receives a segment, it may either discard the segment, or pass up to the application layer this segment along with any buffered segments that do not violate the transport layer's in order guarantee. There seem to be three natural models of client-side buffering, which we list from least powerful to most powerful:

**no buffering:** When the client transport layer receives a segment, it must either immediately pass the segment up to the application layer, or discard the segment.

**cyclic buffering:** When the client transport layer receives a segment while its buffer is empty, it may buffer the segment. When the client transport layer receives segment  $k$  of a document while its buffer is not empty, it may buffer the segment if segment  $k - 1 \bmod p_i$  is also buffered.

**arbitrary buffering:** When the client transport layer receives a segment, it may buffer the segment.

For the no buffering and arbitrary buffering models, the best algorithm on a client is fixed. If a client cannot buffer, the best algorithm on the client is to discard all segments until it receives segment 1, which it can then pass the segment up to the application layer. After segment  $i$  has been passed up to the application layer, it must discard all segments until it receives segment  $i + 1$ , and so on. If a client can buffer arbitrarily, the best algorithm on the client is to buffer all segments of the requested document that it receives. If a client can buffer cyclically, the best algorithm on the client is unclear; it depends on the server model and/or server algorithm. After a client buffers a segment of a document, the sequence of segments that the client can buffer become fixed. Thus, a

client may choose not to buffer a segment when its buffer is empty in the hope that a later sequence of segments may satisfy the request faster.

In principle a server could transmit any segment of a document at any point of time. In practice, a server that transmits documents sequentially would be easier to implement, have less overhead, and thus have higher throughput. In the traditional unicast setting, where the server can not aggregate responses, it is easy to see that the ability to transmit a document in an arbitrary order would not improve any reasonable QoS measure. In the multicast setting, it is not at all clear whether the ability of the server to transmit segments out of order can improve some reasonable QoS measure. There seem to be three natural models of server-side transmission ordering, which we list from least powerful to most powerful:

**cyclic server ordering:** The first segment of a document that the server sends must be the first segment of that document. If the last segment of a document that the server sent was segment  $i$ , then the next segment of this document that the server sends must be  $(i + 1) \bmod p_i$ .

**cyclic ordering with restart** The first segment that the server sends must be the first segment of the document. If the last segment sent was segment  $i$ , then the next segment must be either segment  $(i + 1) \bmod p_i$  or the first segment. That is, the server may restart sending the document again, say if many new requests arrived for the document.

**arbitrary server ordering:** The server may send any segment at any time.

There is no obvious choice of the “right” model of server-side transmission ordering and client-side buffering. Different models are appropriate under different settings, and researchers have analyzed algorithms for multicast pull scheduling in various models. For example, in [3] it is stated without proof that the server scheduling algorithm First-Come-First-Served (FCFS) is  $O(1)$ -competitive for the objective function of minimizing the maximum flow time, under the assumption of cyclic server ordering and cyclic client buffering. As another example, in [6] it was shown that the server scheduling algorithm BEQUI, which broadcasts each document at a rate proportional to the number of requests for that document, is an  $O(1)$ -speed  $O(1)$ -approximation algorithm for average flow time assuming no client-side buffering and cyclic server-side order. In this context, an  $s$ -speed  $c$ -approximation algorithm  $A$  has the property that for all inputs, the value of the objective function of the schedule that  $A$  produces with speed  $s$  processors is at most  $c$  times the optimal value of the objective function for speed 1 processors [9, 13]. Intuitively an  $O(1)$ -speed  $O(1)$ -competitive algorithm is guaranteed to perform well if the system is not too heavily loaded.

## 1.1 An Example Instance of Multicast Pull Scheduling

To better understand the problem consider the instance shown in figure 1. There are two documents (designated by the rectangles with two different fill patterns in the input). The first document is of length 9, and the second document is of length 1 (designated by the horizontal lengths of the rectangles in the input). The first document is requested at times 0 and time 7 (designated by the horizontal positioning of the leftmost portion of the rectangles in the input). The second document is requested at times 1, 3 and 5. Let us assume cyclic server ordering. Presumably the server starts broadcasting the first document at time 0. At time 1, a request for the second document arrives and the server must decide whether to continue broadcasting the first document, or preempt and start broadcasting the second document. In the feasible schedules shown, the second document is broadcast from time 5 to time 6, satisfying all of the requests for the second document with this

single broadcast. Consider time 10, when these schedules finish broadcasting the first document. If there is no client-side buffering the server must again broadcast all 9 units of the first document. The average flow time for this schedule would then be  $(10 + 5 + 3 + 1 + 12)/5$ , where the flow times of the individual requests in the numerator are ordered by increasing arrival times of the requests. If client-side buffering is available, the server need only rebroadcast the first 7 segments of the first document. The average flow time for this schedule would then be  $(10 + 5 + 3 + 1 + 10)/5$ .

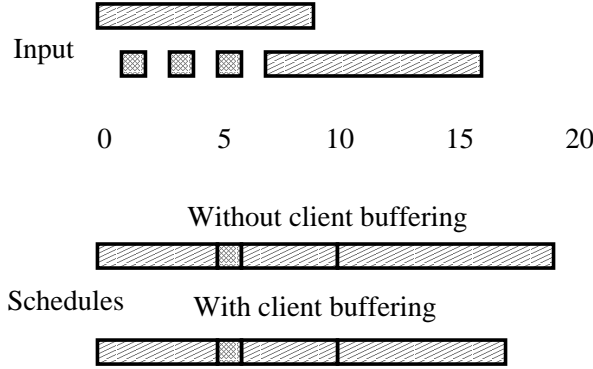


Figure 1: A example of the usefulness of client buffering

## 1.2 Our Results

Informally, the question that we address in this paper is, “How do the various models of server-side and client-side capabilities affect the QoS provided by the system?” We primarily focus on comparing the optimal schedules in the various models, which in some sense measures the inherent limit of the QoS obtainable from the system. For example, one concrete question we address is “How much better can the average flow time be in an optimal schedule when you have arbitrary server-side ordering and arbitrary client-side buffering as compared to the average flow time for the optimal schedule with cyclic server-side ordering and no client side buffering?” We give a reasonably complete comparison of the QoS achievable in the various models of server/client capabilities for QoS measures that are  $l_p$  norms of flow time and stretch.

One natural question that arises when one reads the results in [3, 6] is whether these results carry over to other models, e.g. “Is FCFS still  $O(1)$  competitive if you have arbitrary server-side ordering and arbitrary client-side buffering?” The results of our comparisons of optimal schedules can answer these questions for the results in [3, 6], and for some reasonable collection of future results that researchers might obtain. For example, if the QoS of the optimal schedule in a more powerful model (that is, where the server/client have additional capabilities) is not much better then the QoS of the optimal schedule in a weaker model, then one can conclude that a competitive online algorithm in the weaker model is competitive in the stronger model. Obviously a competitive online algorithm  $A$  in a stronger model is competitive in a weaker model, provided that  $A$  is implementable in this weaker model. Where it is possible to do so, we give a general method for converting a general online algorithm  $A$  in a stronger model, to a online algorithm  $B$  in a weaker model, in such a way that there is little or no deterioration in the QoS for all inputs. We also note when such a general transformation is not possible.

In section 2 we show that in all client models, it is the case that the optimal schedule with

cyclic server ordering dominates the optimal schedule with arbitrary server ordering; A schedule  $S$  *dominates* a schedule  $T$  if the completion time of every request in  $S$  is no later than the completion time of that request in  $T$ . When the client cannot buffer, it is not generally possible to transform an online server algorithm  $A$  in a stronger server model to an equally good online algorithm  $B$  in a weaker server model. In contrast, for cyclic client-side buffering and arbitrary client-side buffering, we can convert any online server algorithm using arbitrary ordering to one that is just as good but uses only cyclic server ordering.

For optimal offline server, arbitrary server ordering provides no benefit above and beyond cyclic server ordering under all reasonable QoS measures (those that are monotone in the completion times of the requests). We thus assume cyclic server ordering for the rest of the paper. Given cyclic server ordering, it is easy to see that the optimal schedule with cyclic client-side buffering dominates the optimal schedule with arbitrary client-side buffering. Thus arbitrary buffering offers no advantages over cyclic buffering. The rest of paper is then devoted to comparing cyclic client-side buffering with no client-side buffering.

In section 3 we show that the optimal maximum flow time obtainable with client-side buffering is no less than half of the optimal maximum flow time that is obtainable without client-side buffering. We also show that the same result holds when the QoS measure is maximum stretch. Thus client-side buffering doesn't drastically improve the optimal maximum flow time or maximum stretch. And as a consequence of this result, we can conclude that the result from [3], that FCFS is  $O(1)$ -competitive with respect to maximum flow time, extends to all models considered here. We also show that this bound of half is tight, that is, there are schedules where buffering reduces the value of the maximum flow time and the value of the maximum stretch by a factor of two.

In contrast we show, in section 4, that for finite  $p$ , the  $l_p$  norm of the flow times and stretches in the optimal schedule with client-side buffering can be a factor of  $n^{\Theta(1/p)}$  less than those in the optimal schedule with no client-side buffering. Thus for these QoS measures, client-side buffering can vastly improve the QoS provided by the system. In particular we show that the value of  $F^{(1)}$ ,  $ST^{(1)}$  and  $ST^{(2)}$  with client-side buffering can be at least a factor of  $n^{1/3}$  smaller than those with no client-side buffering. Further, we show that  $F^{(p)}$ ,  $p > 1$ , can be at least a factor of  $n^{1/2p}$  better, and that  $ST^{(p)}$ ,  $p > 2$ , can be at least a factor of  $n^{1/p}$  better, with client-side buffering than without client-side buffering. We show that for these QoS measures, client-side buffering can improve the the achievable QoS by at most a factor of  $n^{1/p}$ . It is at least of academic interest to close the gap between these upper and lower bounds. While we are not able to fully close the gap here, we make some progress for average flow time. That is, for the case that  $p = 1$ , where the obvious upper bound is  $n$ , we show that client side buffering can improve the average flow time by at most a factor of  $n^{1/2}$ .

In section 5 we show that the optimal schedule with a 2-speed server (that is, it can transmit two segments in unit time) assuming no client-side buffering dominates the optimal schedule for a 1-speed server assuming client-side cyclic buffering. Thus there is at most an 2-speed difference between any of the models. As a consequence of this result, we can conclude that the result from [6], that BEQUI is an  $O(1)$ -speed  $O(1)$ -approximation algorithm, extends to all models considered here.

We try to summarize these many results (other than the resource augmentation result) in the following figure and table. In figure 2, each small circle represents a model with characteristics described by the corresponding vertical and horizontal descriptions. A solid edge between two models signifies that the optimal schedule in the two models are equivalent. The label of each edge tells where the result is stated and proved.

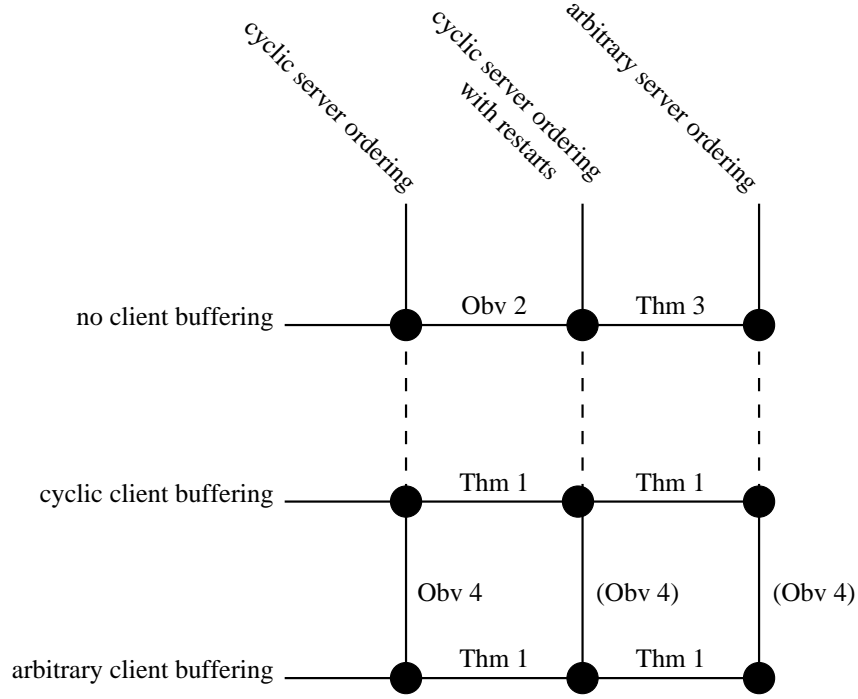


Figure 2: Equivalence of the optimal schedule among models.

	$p = 1$ (total flow time or total stretch)	$p \geq 2$	$p = \infty$ (max flow time or max stretch)
flow time	$\Omega(n^{1/3})$ , $p = 1$ (Thm 12) $\leq \sqrt{n} + 1$ , $p = 1$ (Thm 15)	$\Omega(n^{1/2p})$ , $p \geq 2$ (Thm 12) $O(n^{1/p})$ , $p \geq 2$ (Thm 14)	$\geq 2 - \frac{2}{n+2}$ (Thm 8) $\leq 2$ (Thm 7)
stretch	$\Omega(n^{1/3})$ , $p = 1, 2$ (Thm 13) $O(n)$ , $p = 1$ (Thm 14)	$\Omega(n^{1/p})$ , $p \geq 3$ (Thm 13) $O(n^{1/p})$ , $p \geq 2$ (Thm 14)	$\geq 2 - \frac{1}{2^{n-2}}$ (Thm 9) $\leq 2$ (Thm 7)

Table 1: Bounds on flow time and stretch between the optimal schedules with and without client-side buffering

### 1.3 Additional Related Results

A nice introduction to multicast pull scheduling, and some experimental results, can be found in [1]. In [3] it is shown that minimizing the maximum flow time can be approximated well offline, under the assumption of cyclic server order and cyclic client buffering. There are also results in the literature concerning the problem of minimizing the average flow time of non-preemptable unit jobs, in part because unit jobs are easier to deal with combinatorially. This problem was shown to be NP-hard in [7]. An offline  $O(1)$ -speed  $O(1)$ -approximation algorithm can be found in [10]. The constants are improved in [7, 8]. A online variation of BEQUI is shown to be an  $O(1)$ -speed  $O(1)$ -approximation algorithm in [6]. There has been a fair amount of research into push-based broadcast systems, sometimes called broadcast disks, where the server pushes information to the clients without any concept of a request, such as a TV broadcast [2, 11]. It seems that good strategies for push-based systems have little to do with good strategies for pull-based systems.

## 2 Server-Side Ordering

We show that for all client models, the optimal schedule in the weakest server model dominates the optimal schedule in the strongest server model. For client-side cyclic buffering and client-side arbitrary buffering this is relatively straightforward to see.

**Theorem 1** If the clients use either cyclic buffering, or arbitrary buffering, then the optimal schedule with cyclic server ordering dominates the optimal schedule with arbitrary server ordering.

*Proof:* It is sufficient to explain how to turn a schedule  $S$ , with arbitrary server ordering, into a schedule  $T$ , with cyclic server ordering, in such a way that none of the completion times increase. For all times  $t$ , the document that  $T$  is broadcasting at the time  $t$  is identical to the document that  $S$  is broadcasting at the time  $t$ , although the segment within the document that  $S$  and  $T$  broadcast at time  $t$  may be different. The first time that  $S$  broadcasts any segment of a document  $D_j$ ,  $T$  broadcasts the first segment of  $D_j$ . This fixes  $T$ 's schedule. To see that  $T$  doesn't increase any completion time note that a request  $R_i$  for document  $D_j$  can not complete in  $S$  until  $p_j$  segments of  $D_j$  have been broadcast, which will be when  $R_i$  completes in  $T$ . ■

Note that the above construction can be carried out online. That is, if clients have some form of buffering, then an online server algorithm  $A$  that uses arbitrary ordering can be converted to an online server algorithm  $B$  that uses only cyclic ordering, with no loss of QoS on all inputs. Assuming that the server uses cyclic ordering, the best client-side algorithm with cyclic buffering is now apparent; A client should start buffering when it receives any segment of the requested document.

We now turn our attention to the case when there is no client-side buffering. It is easy to see that allowing restarts will not improve the optimal offline schedule with cyclic ordering (any partial attempts to broadcast a document before a restart may be omitted without increasing any completion time).

**Observation 2** If the clients cannot buffer, then the optimal schedule, with cyclic server ordering, dominates the optimal schedule, with cyclic server ordering with restarts.

We are thus left to prove that arbitrary server ordering is no better than cyclic ordering with restarts.

**Theorem 3** If the clients cannot buffer, then the optimal schedule, with cyclic server ordering with restarts, dominates the optimal schedule with arbitrary server ordering.

*Proof:* Consider an arbitrary schedule  $S$  with arbitrary server ordering. Let  $D_j$  be an arbitrary document. Initially,  $T_0$  is set to  $S$  and  $t_0$  is set to 0. In iteration  $k$ , we will transform  $T_{k-1}$  into  $T_k$  so that (1)  $T_k$  follows the cyclic server ordering property for document  $D_j$  up through some time  $t_k$  where  $t_k > t_{k-1}$ , (2) the last piece of  $D_j$  is broadcast at time  $t_k$ , and (3) the completion time of every request does not increase. We can repeat the procedure so that the the schedule follows the cyclic server ordering property at any time and for any document.

We now describe the procedure for iteration  $k$ . Let  $C$  be the earliest time strictly greater than  $t_{k-1}$  such that some requests to  $D_j$  complete at time  $C$  in  $T_{k-1}$ . Let  $W$  be the set of requests (to  $D_j$ ) that complete at time  $C$  in  $T_{k-1}$ . Let  $R_l$  be the request in  $W$  with the largest release time.

Let  $U = \{u_1, \dots, u_b\}$  be the set of times in the interval  $[\max\{t_{k-1} + 1, r_l\}, C]$  at which  $T_{k-1}$  was broadcasting a segment of  $D_j$ . Since  $T_{k-1}$  satisfies the cyclic server ordering property up through time  $t_{k-1}$ , then the client that issues  $R_l$  cannot receive any segment of  $D_j$  before time  $t_{k-1} + 1$ . At any rate, since  $R_l$  completes at time  $C$ , then there are times in the interval  $[\max\{t_{k-1} + 1, r_l\}, C]$  in  $T_{k-1}$  at which the  $p_j$  segments of  $D_j$  are broadcast in order. Let  $V = \{v_1, \dots, v_{p_j}\}$  be the set of such times. Note that  $V \subseteq U$ . We may assume that  $u_1 < \dots < u_b$  and  $v_1 < \dots < v_{p_j}$ .

We modify  $T_{k-1}$  into  $T_k$  by rearranging and removing broadcasts of segments of  $D_j$  in the interval  $[t_{k-1} + 1, u_b]$  as follows. In  $T_k$ , broadcast the  $p_j$  segments of  $D_j$  in order at times  $u_1, \dots, u_{p_j}$ . Note, these are also the segments that are broadcast in  $T_{k-1}$  at the times in the set  $V$ . Now consider the segments that are broadcast in  $T_{k-1}$  at the times in the set  $U - V$ . Broadcast these segments in  $T_k$  at times  $u_{p_j+1}, \dots, u_b$  in the same order that they are broadcast in  $T_{k-1}$ . In addition, if  $t_{k-1} + 1 < r_l$ , remove any broadcast of any segment of  $D_j$  in the interval  $[t_{k-1} + 1, u_1 - 1]$ . Set  $t_k$  to  $u_{p_j}$ . Note that  $t_k = u_{p_j} \geq u_1 + p_j - 1 \geq t_{k-1} + p_j$ . This completes iteration  $k$ .

We now show that the modification in iteration  $k$  does not increase the flow time of any request. First, the requests to other documents are not affected. Second, the requests in  $W$  are satisfied by the broadcasts at times  $u_1, \dots, u_{p_j}$ . Third, by definition of  $C$ , any request to  $D_j$  that completes strictly before time  $C$  completes no later than time  $t_{k-1}$ . Thus, they are not affected. Finally, consider any request  $R_i$  to  $D_j$  that completes strictly after time  $C$ . By definition of  $R_l$ , it is the case that  $r_l < r_i$ . Thus, in  $T_{k-1}$ ,  $R_i$  does not store any of the segments in  $V$ . Otherwise, it would complete at time  $C$ , which would be a contradiction. Therefore, in  $T_k$ ,  $R_i$  can store none of the segments broadcast at times  $u_1, \dots, u_{p_j}$ .

The remaining broadcasts (at times  $u_{p_j+1}, \dots, u_b$ ) of  $D_j$  in  $T_k$  are only moved later in time compared to  $T_{k-1}$ . Thus,  $R_i$  does not miss a chance to receive them. Also, any broadcast after time  $C$  is not affected, thus the completion time of  $R_i$  does not increase.  $\blacksquare$

We can now conclude that in every client model, the QoS of the optimal schedule in the strongest server model is no better than the QoS of the optimal schedule in the weakest server model. Assuming cyclic server ordering, it is then easy to see that the optimal schedule with cyclic client buffering dominates the optimal schedule for arbitrary client buffering. To see this note that since the documents are broadcast cyclically, the client doesn't even have the opportunity to buffer an out of order segment.

**Observation 4** If the server uses cyclic ordering, then the optimal schedule, with cyclic client-side buffering dominates the optimal schedule, with arbitrary client-side buffering.

The rest of the paper assumes cyclic server ordering and is devoted to comparing cyclic client-side buffering (which we will simply refer to as buffering) and no client-side buffering.

As an aside we note that when the client does not have any buffering, it is not generally possible to transform an online server algorithm  $A$  in a stronger server model to an equally good online algorithm  $B$  in a weaker server model, as was the case when clients had some form of buffering. We merely sketch the arguments here. First consider the case that  $A$  is an online server algorithm that uses cyclic ordering with restart. Assume that  $A$  gets  $k$  requests for some document  $D_j$ , and immediately starts transmitting  $D_j$ . If  $B$  doesn't immediately start transmitting  $D_j$  then it could have much worse total flow time than  $A$  if the server becomes heavily saturated right after  $A$  finishes transmitting  $D_j$ . If  $B$  starts transmitting  $D_j$  when  $A$  does, then assume that  $k^2$  requests arrive for  $D_j$ , and that as a result  $A$  starts broadcasting  $D_j$  again from the beginning. The algorithm  $B$  can not start retransmitting  $D_j$  since it does not have the power of restart. Thus if the server becomes saturated after  $A$  finishes  $D_j$ , then  $B$ 's total time will be much worse than  $A$ 's flow time.

Now consider the case that  $A$  can transmit arbitrary segments, and  $B$  must use cyclic ordering with restart. Assume that initially  $k^2$  requests arrive for a document  $D_j$ , and that  $A$  immediately begins transmitting  $D_j$ . To have any hope of being competitive with  $A$ ,  $B$  must also immediately begin transmitting  $D_j$ . After time  $p_j/2$ ,  $k$  new requests arrive for  $D_j$ , and  $A$  begins to start retransmitting  $D_j$  again from the beginning. If  $B$  does not immediately switch to transmitting  $D_j$  again from the beginning, then it can not satisfy all requests by time  $3p_j/2$  even though  $A$  can. Thus if the system become saturated at time  $3p_j/2$  then  $B$  will not be competitive. On the other hand, if  $B$  switches to rebroadcasting  $D_j$ , then  $A$  switches back to continuing its first broadcast of  $D_j$ .  $B$  can not switch back since it may only restart. Thus  $A$  satisfies the  $k^2$  requests before  $B$  does, and  $B$  won't be competitive with respect to total flow time if the server becomes saturated right after time  $p_j$ .

Before proceeding on to the next section we need to introduce some terms and notation.

**Definition 5** The *lifespan* of a request  $R_i$  in schedule  $S$  to be the time interval  $L_i(S) = [r_i, C_i(S)]$ .

Note that if there are two requests  $R_i$  and  $R_j$  for the same document with  $r_i < r_j$ , then  $C_i(S) \leq C_j(S)$  since the segments that the client that issued  $R_i$  has seen is a strict superset of the segments that the client that issued  $R_j$  has seen.

In a schedule with no client buffering, the transmissions of segments of a document  $D_j$  in a schedule  $S$  can be partitioned into *broadcasts of  $D_j$* .

**Definition 6** A *broadcast* of a document  $D_j$  begins with the transmission of the first segment of  $D_j$  and ends at the next time the last segment of  $D_j$  is transmitted.

Thus the flow time for a request for  $D_j$  in an unbuffered schedule is the time until the start of the next broadcast of  $D_j$  plus the duration of that broadcast.

### 3 Cyclic Buffering vs. No Buffering in $l_\infty$ Norms

In this section, we show that lack of client buffering does not greatly affect the QoS if the QoS measure is maximum flow time or maximum stretch. More precisely, it at most doubles the maximum flow time and maximum stretch. Further we show that this bound is tight.

**Theorem 7** For any schedule  $S$  with cyclic client buffering, there exists an unbuffered schedule  $T$  such that both  $F^{(\infty)}(T) \leq 2F^{(\infty)}(S)$  and  $ST^{(\infty)}(T) \leq 2ST^{(\infty)}(S)$  hold.

*Proof:* Consider some arbitrary document  $D_j$ . We determine when  $T$  broadcasts  $D_j$  by iteratively repeating the following process. Let  $R_l$  be the latest request to  $D_j$  that we have not previously considered. There must be exactly  $p_j$  times in  $L_l(S)$  when  $S$  is broadcasting segments of  $D_j$ ; let  $k_i$ ,  $1 \leq i \leq p_j$ , be the  $i$ th such time. Then at time  $k_i$ ,  $T$  is broadcasting segment  $i$  of  $D_j$ . Let  $\mathcal{R}$  be the set of requests to document  $D_j$  that we have not previously considered, whose lifespan intersects with  $L_l(S)$ . Request  $R_l$  and the requests in  $\mathcal{R}$  are removed from further consideration.

To see that the schedule  $T$  satisfies the desired conditions, consider a request  $R_l$  for  $D_j$  that was selected at some iteration. Notice that  $C_l(T) = C_l(S)$  by construction, and hence  $F_l(T) = F_l(S)$ , and  $ST_l(T) = ST_l(S)$ . Further for each request  $R_a \in \mathcal{R}$  it is the case that  $C_a(S) \leq C_l(S)$  since  $r_a \leq r_l$  by the definition of  $R_l$ . Since  $L_a(S)$  and  $L_l(S)$  overlap, it must then be the case that  $F_a(T) \leq F_a(S) + F_l(S)$ . Since requests  $R_a$  and  $R_l$  are for the same document  $ST_a(T) \leq ST_a(S) + ST_l(S)$ . Hence, the flow time (stretch) of every request in  $T$  is at most the sum of the flow times (stretches) of two requests in  $S$ . The claimed results then immediately follow. ■

Next we show a matching lower bound.

**Theorem 8** For all even  $n$ , there exists an instance  $I_n$  with  $n$  requests such that the maximum flow time of the optimal unbuffered schedule is at least  $2 - \frac{2}{n+2}$  times the maximum flow time of the optimal buffered schedule. Similarly, with the same instance  $I_n$ , the maximum stretch of the optimal unbuffered schedule is at least  $2 - \frac{2}{n+2}$  times the maximum stretch of the optimal buffered schedule. Here  $n$  is the number of requests in  $I_n$ .

*Proof:* First we describe how to construct  $I_n$ . A total of  $n/2$  distinct documents will be requested. Each of the documents has  $n + 2$  segments. For  $i = 1, \dots, n/2$ , there are 2 requests to document  $D_i$ ; request  $R_{2i-1}$  arrives at time  $r_{2i-1} = (2n + 2)(i - 1)$ , and request  $R_{2i}$  arrives at time  $r_{2i} = (2n + 2)(i - 1) + n$ .

Consider the buffered schedule  $S$  where document  $D_i$  is broadcast cyclically from time  $r_{2i-1} = (2n + 2)(i - 1)$  to time  $r_{2i+1} = (2n + 2)i$ . Request  $R_{2i-1}$  completes at time  $(2n + 2)(i - 1) + n + 2$ . Request  $R_{2i}$  is completes at time  $(2n + 2)i$ . Thus, in  $S$  the flow time and stretch of each request is  $n + 2$  and 1, respectively.

We will show that every unbuffered schedule  $T$  has a request with flow time at least  $2n + 2$  and stretch at least  $2 - 2/(n + 2)$ . First consider the case that there exist two requests  $R_{2i-1}$  and  $R_{2i}$  that are satisfied by the same broadcast  $B$ . In order to satisfy  $R_{2i}$ ,  $B$  cannot begin before  $r_{2i}$ . The completion time of  $R_{2i-1}$  can then be no earlier than  $r_{2i} + n + 2 = (2n + 2)i$ . Then the flow time of  $R_{2i-1}$  in  $T$  is at least  $(2n + 2)i - r_{2i-1} = (2n + 2)i - (2n + 2)(i - 1) = (2n + 2)$ . Thus we have established the desired result in this case.

Now consider the case that no two requests are satisfied by the same broadcast. There must then be a total of  $n$  broadcasts in  $T$ . Since all documents have  $n + 2$  segments, some broadcast in  $T$  finishes no earlier than time  $n(n + 2)$ . The latest release time is  $r_n = (2n + 2)(n/2 - 1) + n = n^2 - 2$ . Thus, there must be a request with flow time  $n(n + 2) - r_n = 2n + 2$ .

In either case, there is a request with flow time  $2n + 2$ . Since all documents have  $n + 2$  segments, then there is a request with stretch  $(2n + 2)/(n + 2) = 2 - 2/(n + 2)$ . ■

It should be noted that one can improve the maximum stretch lower bound to  $2 - 1/2^{n-2}$  at the cost of complicating the construction slightly.

**Theorem 9** For all  $n \geq 3$ , there exists an instance  $I_n$  with  $n$  request such that the maximum stretch of the optimal unbuffered schedule is at least  $2 - \frac{1}{2^{n-2}}$  times the maximum stretch of the optimal buffered schedule.

*Proof Sketch:* We only describe how to construct  $I_n$ , but omit the argument on computing the stretch. A total of  $n - 1$  distinct documents will be requested. For  $j = 1, \dots, n - 1$ , document  $D_j$  has  $2^{n-1-j}$  segments. Request  $R_1$  is to document  $D_1$  and  $r_1 = 0$ . Requests  $R_2$  through  $R_n$  are to documents  $D_1$  through  $D_{n-1}$ , respectively. They arrive back to back in order beginning at time  $r_2 = 2^{n-2} - 1$ . ■

## 4 Cyclic Buffering vs. No Buffering in $l_p$ Norms

In this section we first show that if  $p$  is a finite integer, then the  $l_p$  norms of the flow times and stretches of buffered schedules can be a factor of  $n^{\Theta(1/p)}$  better than those of unbuffered schedules.

### 4.1 Lower Bounds

All lower bound instances to be presented have the same structure, which we now define formally. This structure is depicted in Figure 3.

**Definition 10** Let  $(k_1, 1, p_1 ; k_2, 1, p_1 ; k_3, m_3, p_3 ; k_4, m_4, p_4)$  denote an input instance for the broadcast scheduling problem where the requests are partitioned into four groups  $Q_1, Q_2, Q_3$  and  $Q_4$  as follows:

- There are  $k_1$  requests in  $Q_1$ , all of which arrive at time 0, and are to a document  $D_1$  which has length  $p_1$ .
- There are  $k_2$  requests in  $Q_2$ , all of which arrive at time  $p_1/2$ , and are to document  $D_1$ .
- There are  $k_3 m_3$  requests in group  $Q_3$ . From time  $p_1$  to  $p_1 + p_3(m_3 - 1)$ , there are  $k_3$  requests to a new document of length  $p_3$  every  $p_3$  time units.
- There are  $k_4 m_4$  requests in group  $Q_4$ . From time  $p_1 + p_3 m_3 + p_1/2$  to time  $p_1 + p_3 m_3 + p_1/2 + p_4(m_4 - 1)$  there are  $k_4$  requests to a new document of length  $p_4$  every  $p_4$  time units.

Note that all requests that arrive at different times are to distinct documents with the exception of requests in  $Q_1$  and  $Q_2$  which are to the same document  $D_1$ . Thus, the number of documents and the number of requests, respectively, are

$$m = 1 + m_3 + m_4 \tag{1}$$

$$n = k_1 + k_2 + k_3 m_3 + k_4 m_4 \tag{2}$$

The structure of a schedule  $S$  with client-side buffering is shown in Figure 3. This schedule will be used as an upper bound for the optimal schedule with client-side buffering. Except for the requests for document  $D_1$  at time  $p_1/2$ ,  $S$  starts broadcasting a document as soon as the requests for that document arrive, and continues broadcasting the document without interruption until the requests are satisfied. The requests for document  $D_1$  at time  $p_1/2$  arrive while document  $D_1$  is being broadcast. The clients that initiate these requests begin receiving and buffering the second half of

the document immediately. The first half of the document is broadcast again in  $S$  between time  $p_1 + m_3p_3$  and  $p_1 + m_3p_3 + p_1/2$ . The  $l_p$  norm of flow time and stretch for  $S$  is calculated as follows:

$$F^{(p)}(S) = (k_1p_1^p + k_2(p_1 + m_3p_3)^p + k_3m_3p_3^p + k_4m_4p_4^p)^{1/p} \quad (3)$$

$$ST^{(p)}(S) = (k_1 + k_2(1 + m_3p_3/p_1)^p + k_3m_3 + k_4m_4)^{1/p} \quad (4)$$

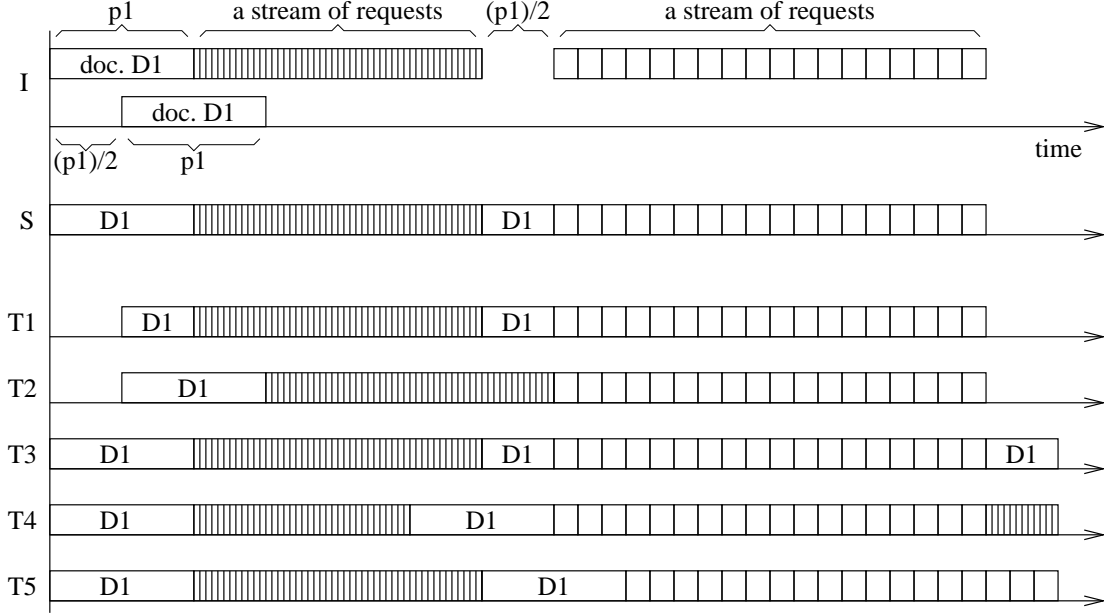


Figure 3: The structure of lower bound input instances, buffered schedule  $S$ , and representative unbuffered schedules  $T_1, T_2, T_3, T_4$  and  $T_5$ .

Lemma 11 If  $I = (k_1, 1, p_1 ; k_2, 1, p_1 ; k_3, m_3, p_3 ; k_4, m_4, p_4)$ , then for any unbuffered schedule  $T$  for  $I$  and for any integer  $p \geq 1$ ,

$$F^{(p)}(T) = \Omega \left( \min \left\{ k_1^{1/p} m_3 p_3, (k_3 m_3)^{1/p} p_1, k_2^{1/p} m_4 p_4, (k_3 p_1 / p_3)^{1/p} m_4 p_4, (k_4 m_4)^{1/p} p_1 \right\} \right)$$

and

$$ST^{(p)}(T) = \Omega \left( \min \left\{ \frac{k_1^{1/p} m_3 p_3}{p_1}, \frac{(k_3 m_3)^{1/p} p_1}{p_3}, \frac{k_2^{1/p} m_4 p_4}{p_1}, \frac{\left( \frac{k_3 p_1}{p_3} \right)^{1/p} m_4 p_4}{p_3}, \frac{(k_4 m_4)^{1/p} p_1}{p_4} \right\} \right).$$

*Proof:* We consider flow first. Without loss of generality, we can assume that document  $D_1$  is broadcast no more than twice in schedule  $T$ . We break the proof into cases. A representative schedule for each case is shown in Figure 3. Intuitively, in any unbuffered schedule, many requests from one of the groups will finish much later than that in schedule  $S$ , which is a buffered schedule. In schedules  $T_1, T_2, T_3, T_4, T_5$ , such groups are  $Q_1, Q_3, Q_2, Q_3, Q_4$ , respectively.

We give an overview of the cases in the proof. If  $D_1$  is broadcast only once (case 1), then there is a contention between requests in groups  $Q_1, Q_2$ , and  $Q_3$ . Either groups  $Q_1$  and  $Q_2$  will finish after  $Q_3$  (case 1.1, schedule  $T_1$ ) or vice versa (case 1.2, schedule  $T_2$ ). If  $D_1$  is broadcast twice (case 2),

then there is contention between group  $Q_2$ ,  $Q_3$ , and  $Q_4$ . Either  $Q_2$  will finish last (case 2.1, schedule  $T_3$ ), or  $Q_3$  will finish last (case 2.2.1, schedule  $T_4$ ), or  $Q_4$  will finish last (case 2.2.2, schedule  $T_5$ ).

**Case 1:** In this case we assume that  $D_1$  is broadcast once in  $T$ . Let  $C_1$  be the time that the server finishes the broadcast of  $D_1$  in  $T$ . We consider two subcases.

**Case 1.1:** In this case we assume that  $F_1 = C_1 \geq p_1 + m_3p_3/2$ . Then  $F^{(p)}(T)$  is at least  $(k_1F_1^p)^{1/p} \geq (k_1(m_3p_3/2)^p)^{1/p} = \Omega(k_1^{1/p}m_3p_3)$ . Schedule  $T_1$  in Figure 3 is a representative for this case.

**Case 1.2:** In this subcase we assume that  $C_1 < p_1 + m_3p_3/2$ . Since  $D_1$  is broadcast only once in  $T$ , the server begins broadcasting  $D_1$  no earlier than time  $p_1/2$ . At time  $p_1$ , requests in  $Q_3$  begin to arrive while there are at least  $p_1/2$  time units of document  $D_1$  left to be broadcast. Since  $C_1 < p_1 + m_3p_3/2$ , which is the time that the second half of requests in  $Q_3$  arrive, some requests in  $Q_3$  will be delayed. The order that the requests in  $Q_3$  are serviced doesn't affect the flow time for these documents since each document has the same length. Thus we can assume, without loss of generality, that requests in  $Q_3$  are serviced in  $T$  in the order that they arrive. and there are an equal number of requests to each document. Thus, there are at least  $k_3m_3/2$  requests in  $Q_3$ , each of which has flow time at least  $p_1/2$ . Based on the flow time of these requests,  $F^{(p)}(T)$  is at least  $((k_3m_3/2)(p_1/2)^p)^{1/p} = (k_3m_3/2)^{1/p}(p_1/2) = \Omega((k_3m_3)^{1/p}p_1)$ . Schedule  $T_2$  in Figure 3 is a representative for this case.

**Case 2:** In this case we assume that  $D_1$  is broadcast twice in  $T$ . Without loss of generality, we can assume that the requests in  $Q_2$  are satisfied by the second broadcast of  $D_1$  in  $T$ . Let  $C_2$  be the time that the server finishes the second broadcast of  $D_1$  in  $T$ . We break the proof into subcases.

**Case 2.1:** In this subcase we assume that  $C_2 \geq 3p_1/2 + m_3p_3 + m_4p_4/2$ . Thus, the flow time of each request in  $Q_2$  is  $F_2 = C_2 - p_1/2$ . Based on this,  $F^{(p)}(T)$  is at least  $(k_2F_2^p)^{1/p} \geq (k_2(m_4p_4/2)^p)^{1/p} = \Omega(k_2^{1/p}m_4p_4)$ . Schedule  $T_3$  in Figure 3 is a representative for this case.

**Case 2.2:** In this subcase we assume that  $C_2 < 3p_1/2 + m_3p_3 + m_4p_4/2$ . Let  $W_3$  be the set of documents in  $Q_3$  whose broadcasts end at or after time  $3p_1/2 + m_3p_3 + m_4p_4/2$ . We again break this subcase into two further subcases.

**Case 2.2.1:**  $|W_3| \geq p_1/(4p_3)$ . In this case the contribution of the requests in  $W_3$  to  $F^{(p)}(T)$  is at least

$$\left( k_3 \frac{p_1}{4p_3} \left( \frac{m_4p_4}{2} \right)^p \right)^{\frac{1}{p}} = \Omega \left( \left( \frac{k_3p_1}{p_3} \right)^{\frac{1}{p}} m_4p_4 \right)$$

Schedule  $T_4$  in Figure 3 is a representative for this case.

**Case 2.2.2:** In the final subcase we assume that  $|W_3| < p_1/(4p_3)$ . Let  $Q_{123} = Q_1 \cup Q_2 \cup Q_3 - W_3$ . The total time needed to satisfy the requests in  $Q_{123}$  is at least  $2p_1 + m_3p_3 - |W_3|p_3 \geq 2p_1 + m_3p_3 - p_1/4 = 3p_1/2 + m_3p_3 + p_1/4$ . At time  $3p_1/2 + m_3p_3$ , requests in  $Q_4$  begin to arrive, while at least  $p_1/4$  time units more is needed to broadcast documents to satisfy the requests in  $Q_{123}$ . Since all requests in  $Q_{123}$  are satisfied by time  $3p_1/2 + m_3p_3 + m_4p_4/2$  which is the time the second half of requests in  $Q_4$  begin to arrive, some requests in  $Q_4$  will be delayed. The order that the requests in  $Q_4$  are serviced doesn't affect the flow time for these documents since each document has the same length and the same number of outstanding requests. Thus we can assume, without loss of generality, that requests in  $Q_4$  are serviced in  $T$  in the order that they arrive. and there are an equal number of requests to each document. Thus, there are at least  $k_4m_4/2$  requests in  $Q_4$ , each of which has flow time at least  $p_1/4$ . Based on the flow time of these requests,  $F^{(p)}(T)$  is at least  $((k_4m_4/2)(p_1/4)^p)^{1/p} = (k_4m_4/2)^{1/p}(p_1/4) = \Omega((k_4m_4)^{1/p}p_1)$ . Schedule  $T_5$  in Figure 3 is a representative for this case.

Since  $F^{(p)}(T)$  is asymptotically lower bounded by one of the cases considered above, the result follows.

The argument for stretch is basically the same. Observe that  $ST_i(T) = F_i(T)/p_j$  where  $p_j$  is the length of the document corresponding to request  $R_i$ . Notice that each term in the statement of this theorem is obtained by dividing the corresponding term by the number of segments of an appropriate document. The lengths are  $p_1, p_3, p_1, p_3$ , and  $p_4$ , respectively. ■

We now apply this lemma to get our lower bounds for flow time.

**Theorem 12** For any integer  $p \geq 1$ , there exists an input instance  $I_p$  such that

$$\frac{F^{(p)}(T)}{F^{(p)}(S')} = \Omega(m^{1/3}) = \begin{cases} \Omega(n^{1/3}) & p = 1 \\ \Omega(n^{1/2p}) & p \geq 2 \end{cases}$$

where  $T$  is the optimal unbuffered broadcast schedule for  $I_p$  and  $S'$  is the optimal buffered broadcast schedule for  $I_p$ .

*Proof:* Input instance  $I_p$  will be in the form described in Definition 10.  $F^{(1)}(S')$  is upper bounded by  $F^{(1)}(S)$  where  $S$  is the buffered schedule given in Figure 3. Let  $t$  be a large positive integer. For  $p = 1$ , the lower bound instance  $I_1$  is defined as follows;

$$I = (t^2, 1, t^2; t, 1, t^2; 1, t^3, 1; 1, t^3, t).$$

From Equalities (1) and (2),  $m = 1 + t^3 + t^3 = \Theta(t^3)$  and  $n = t^2 \cdot 1 + t \cdot 1 + 1 \cdot t^3 + 1 \cdot t^3 = \Theta(t^3)$ . From Equation (3),  $F^{(1)}(S) = \Theta(t^4)$ . From Lemma 11,  $F^{(1)}(T) = \Omega(t^5)$ . Thus,  $F^{(1)}(T)/F^{(1)}(S) = \Omega(t^5/t^4) = \Omega(t) = \Omega(m^{1/3}) = \Omega(n^{1/3})$ .

For  $p \geq 2$ , an instance  $I_p$  is defined as follows;

$$I_p = (t^p, 1, t^2; 1, 1, t^2; t^{2p-3}, t^3, 1; t^{2p-3}, t^3, t).$$

From Equalities (1) and (2),  $m = 1 + t^3 + t^3 = \Theta(t^3)$  and  $n = t^p \cdot 1 + 1 \cdot 1 + t^{2p-3} \cdot t^3 + t^{2p-3} \cdot t^3 = \Theta(t^{2p})$ . From Equation (3),

$$\begin{aligned} F^{(p)}(S) &= (k_1 p_1^p + k_2 (p_1 + m_3 p_3)^p + k_3 m_3 p_3^p + k_4 m_4 p_4^p)^{1/p} \\ &= (t^p t^{2p} + (t^2 + t^3)^p + t^{2p-3} t^3 + t^{2p-3} t^3 t^p)^{1/p} \\ &= \Theta(t^3) \end{aligned}$$

From Lemma 11,  $F^{(p)}(T) = \Omega(t^4)$ . Thus,  $F^{(p)}(T)/F^{(p)}(S) = \Omega(t^4/t^3) = \Omega(t) = \Omega(m^{1/3}) = \Omega(n^{1/2p})$  for  $p \geq 2$ . ■

We now turn our attention to lower bounds for stretch QoS measures.

**Theorem 13** For any integer  $p \geq 1$ , there exists a input instance  $I_p$  such that

$$\frac{ST^p(T)}{ST^p(S)} = \Omega(m^{1/3}) = \begin{cases} \Omega(n^{1/3}) & p = 1, 2 \\ \Omega(n^{1/p}) & p \geq 3 \end{cases}$$

where  $T$  is the optimal unbuffered broadcast schedule for  $I_p$  and  $S'$  is the optimal buffered broadcast schedule for  $I_p$ .

*Proof:* Input instance  $I_p$  will be in the form described in Definition 10.  $ST^{(1)}(S')$  is upper bounded by  $ST^{(1)}(S)$  where  $S$  is the buffered schedule given in Figure 3. Let  $t$  be a large positive integer. For  $p = 1$ , the lower bound instance  $I_1$  is defined as follows;

$$I = (t^6, 1, t^3 ; t^4, 1, t^3 ; 1, t^5, 1 ; 1, t^6, t).$$

From Equalities (1) and (2),  $m = \Theta(t^6)$  and  $n = \Theta(t^6)$ . From Equation (4),  $ST^{(1)}(S) = \Theta(t^6)$ . From Lemma 11,  $ST^{(1)}(T) = \Omega(t^8)$ . Thus,  $ST^{(1)}(T)/ST^{(1)}(S) = \Omega(t^8/t^6) = \Omega(t^2) = \Omega(m^{1/3}) = \Omega(n^{1/3})$ .

For  $p = 2$ , the lower bound instance  $I_2$  is defined as follows;

$$I = (t^{18}, 1, t^8 ; t^6, 1, t^8 ; 1, t^{14}, 1 ; 1, t^{18}, t^2).$$

From Equalities (1) and (2),  $m = \Theta(t^{18})$  and  $n = \Theta(t^{18})$ . From Equation (4),  $ST^{(2)}(S) = \Theta(t^9)$ . From Lemma 11,  $ST^{(2)}(T) = \Omega(t^{15})$ . Thus,  $ST^{(2)}(T)/ST^{(2)}(S) = \Omega(t^{15}/t^9) = \Omega(t^6) = \Omega(m^{1/3}) = \Omega(n^{1/3})$ .

For  $p \geq 3$ , the lower bound instance  $I_p$  is defined as follows;

$$I = (t^{p^2+p}, 1, t^{2p-1} ; 1, 1, t^{2p-1} ; 1, t^{3p}, 1 ; t^{p^2-2p-3}, t^{3p+3}, t^{p-2}).$$

From Equalities (1) and (2),  $m = 1 + t^{3p} + t^{3p+3} = \Theta(t^{3p+3})$  and  $n = t^{p^2+p} \cdot 1 + 1 \cdot 1 + 1 \cdot t^{3p} + t^{p^2-2p-3} \cdot t^{3p+3} = \Theta(t^{p^2+p})$ . From Equation (4),

$$\begin{aligned} ST^{(p)}(S) &= \left( k_1 + k_2 \left( 1 + \frac{m_3 p_3}{p_1} \right)^p + k_3 m_3 + k_4 m_4 \right)^{1/p} \\ &= \left( t^{p^2+p} + 1 \cdot \left( 1 + \frac{t^{3p} \cdot 1}{t^{2p-1}} \right)^p + 1 \cdot t^{3p} + t^{p^2-2p-3} \cdot t^{3p+3} \right)^{1/p} \\ &= \left( t^{p^2+p} + (1 + t^{p+1})^p + t^{3p} + t^{p^2+p} \right)^{1/p} \\ &= \Theta(t^{p+1}) \end{aligned}$$

From Lemma 11,  $ST^{(p)}(T) = \Omega(t^{2p+2})$ . Thus,  $ST^{(p)}(T)/ST^{(p)}(S) = \Omega(t^{2p+2}/t^{p+1}) = \Omega(t^{p+1}) = \Omega(m^{1/3}) = \Omega(n^{1/p})$  for  $p \geq 3$ .  $\blacksquare$

## 4.2 Upper Bounds

**Theorem 14** For any integer  $p \geq 1$  and for any buffered broadcast schedule  $S$  for an input instance  $I$  with  $n$  requests, there exists an unbuffered broadcast schedule  $T$  for  $I$  such that

$$\frac{F^{(p)}(T)}{F^{(p)}(S)} = O(n^{1/p}) \quad \text{and} \quad \frac{ST^{(p)}(T)}{ST^{(p)}(S)} = O(n^{1/p})$$

*Proof:* The construction of  $T$  from  $S$  is the same as in Theorem 7. Suppose  $R_l$  and  $\mathcal{R}$  are as defined in Theorem 7. Define  $f(a) = l$  for each  $a \in \mathcal{R}$ . Let  $X$  be the set of all requests that are considered as  $R_l$  during the construction. Let  $Y$  be the set of all other requests. Thus, for any request  $R_x \in X$ ,

$F_x(T) = F_x(S)$ , and for any request  $R_y \in Y$ ,  $F_y(T) \leq F_y(S) + F_{f(y)}(S)$ , and  $f(y) \in X$ . Then,

$$\begin{aligned}
F^{(p)}(T) &= \left( \sum_{R_x \in X} F_x^p(T) + \sum_{R_y \in Y} F_y^p(T) \right)^{1/p} \leq \left( \sum_{R_x \in X} F_x^p(S) + \sum_{R_y \in Y} (F_y(S) + F_{f(y)}(S))^p \right)^{1/p} \\
&\leq \left( \sum_{R_x \in X} F_x^p(S) + \sum_{R_y \in Y} 2^{p-1}(F_y^p(S) + F_{f(y)}^p(S)) \right)^{1/p} \\
&\leq \left( n2^{p-1} \sum_{R_x \in X} F_x^p(S) + \sum_{R_y \in Y} 2^{p-1}F_y^p(S) \right)^{1/p} \leq \left( n2^{p-1} \sum_{1 \leq i \leq n} F_i^p(S) \right)^{1/p} \\
&= O(n^{1/p})F^{(p)}(S).
\end{aligned}$$

where the second inequality follows from the following inequality  $(u + v)^p \leq 2^{p-1}(u^p + v^p)$ , which is a consequence of the Cauchy-Schwartz inequality. The third inequality follows from the fact that there are at most  $n - 1$  requests in  $Y$  and  $f(y) \in X$  for any  $R_y \in Y$ . The result for stretch follows the same argument.  $\blacksquare$

We can refine the result in the previous theorem for  $l_1$  norm of the flow time. Specifically, we show that client-side buffering can improve the average flow time by at most a factor of  $n^{1/2}$ .

**Theorem 15** For any buffered broadcast schedule  $S$  for an input instance  $I$  with  $n$  requests, there exists an unbuffered broadcast schedule  $T$  for  $I$  such that  $\frac{F(T)}{F(S)} \leq \sqrt{n} + 1$ .

*Proof:* Initially,  $T$  is set to equal to  $S$  and the flow time of each request is charged to itself, that is, request  $R_i$  is charged  $F_i(S)$ . Then  $T$  will be iteratively modified. A set of requests will be considered in an iteration, but each request will be considered in exactly one iteration. Each request  $R_i$  will be charged at most an extra  $\sqrt{n}F_i(S)$  in the iteration that it is considered. This implies the result. It remains to show how  $T$  is modified and how each request  $R_i$  is charged in each iteration.

We now describe the procedure in each iteration. Let  $R_l$  be the latest request that we have not previously considered. Suppose  $D_j$  is the document requested by  $R_l$ . Let  $U$  be the set of requests to document  $D_j$  that we have not previously considered, whose lifespan intersects the lifespan of  $R_l$ . At the end of the iteration, all requests in  $U \cup \{R_l\}$  are eliminated. Each iteration falls into one of the following 2 cases.

Suppose  $|U \cup \{R_l\}| < \sqrt{n}$ . In the interval  $[r_l, C_l(S)]$ ,  $S$  spends  $p_j$  time units on broadcasting document  $D_j$ . In  $T$  the document  $D_j$  is broadcast beginning with the first segment during the interval  $[r_l, C_l(S)]$  at the same times that  $D_j$  is broadcast in  $S$ . All request in  $U \cup \{R_l\}$  are satisfied by this broadcast. The flow time of request  $R_l$  does not increase by this modification. For any request  $R_i \in U$ , its flow time increases by at most  $F_l(S)$ . There are at most  $\sqrt{n}$  such requests. Thus, the total increase is at most  $\sqrt{n}F_l(S)$ . We charge the increase to  $R_l$ .

Now suppose  $|U \cup \{R_l\}| \geq \sqrt{n}$ . In  $T$  all broadcasts previously scheduled after time  $r_l$  are further delayed by  $p_j$  time units, that is, a broadcast at time  $t > r_l$  is rescheduled to time  $t + p_j$ . Then  $T$  broadcasts document  $D_j$  continuously from time  $r_l$  to time  $r_l + p_j$  beginning with the first segment. The flow time of request  $R_l$  does not increase by this modification. For any request  $R_i \in U$ , its flow time increases by at most  $p_j \leq F_i(S)$ . We charge this to itself. The flow time of each request that is pushed forward increases by at most  $p_j$ . There are at most  $n - |U \cup \{R_l\}| \leq n - \sqrt{n}$  such requests. Thus, the total increase in flow time for these requests is at most  $(n - \sqrt{n})p_j$ . We can charge this increase to the requests in  $U \cup \{R_l\}$ . Each request  $R_i$  in  $U \cup \{R_l\}$  is charged at most

$\frac{(n-\sqrt{n})p_j}{|U \cup \{R_i\}|} \leq \frac{(n-\sqrt{n})F_i(S)}{\sqrt{n}} = (\sqrt{n}-1)F_i(S)$ . Taking into account the increase of its own flow time, each request  $R_i \in U \cup \{R_i\}$  is charged at most by an extra  $\sqrt{n}F_i(S)$  in this iteration. ■

## 5 Resource Augmentation

In this section, we show that there is at most a 2-speed difference between any of the models.

**Theorem 16** The optimal unbuffered schedule  $T$  with a 2-speed server dominates the optimal buffered schedule  $S$  for a 1-speed server.

*Proof:* We will assume that  $S$  broadcasts one segment every two time units, and  $T$  broadcasts one segment every time unit. We construct  $T$  from  $S$  in the following manner. At every time unit,  $T$  broadcasts the document that  $S$  is broadcasting. Since the server is broadcasting cyclically this fixes  $T$ . Note that even though  $S$  and  $T$  broadcast the same document at any time, they may be broadcasting different segments of the document since  $T$  has a faster server.

Consider any request  $R_i$  for a document  $D_j$  with  $p_j$  segments. During the lifespan  $L_i(S)$  of request  $R_i$  in schedule  $S$ , at least  $p_j$  segments of  $D_j$  must be broadcast in  $S$ . Thus, there must be at least  $2p_j$  segments of  $D_j$  in schedule  $T$  in  $L_i(S)$ . Since,  $D_j$  is broadcast cyclically in  $T$ , there must be  $p_j$  segments of  $D_j$  starting from the first segment to the last segment in order in this interval. Thus,  $C_i(T) \leq C_i(S)$ , and the result follows. ■

**Acknowledgments:** The origination of the questions considered in this paper arose from discussions with Jeff Edmonds. The observation that arbitrary server ordering is of no benefit when there is client-side buffering is due to Jiri Sgall. A preliminary version of the paper appeared in the 2002 European Symposium on Algorithms.

## References

- [1] S. Aacharya, and S. Muthukrishnan, “Scheduling on-demand broadcasts: new metrics and algorithms”, ACM/IEEE International Conference on Mobile Computing and Networking, 43 – 54, 1998.
- [2] A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieber, “Minimizing service and operation costs of periodic scheduling”, ACM/SIAM Symposium on Discrete Algorithms, 11 – 20, 1998.
- [3] Y. Bartal, and S. Muthukrishnan, “Minimizing maximum response time in scheduling broadcasts”, ACM/SIAM Symposium on Discrete Algorithms, 558 – 559, 2000.
- [4] Digital Fountain website, <http://www.digitalfountain.com>.
- [5] DirecPC website, <http://www.direcpc.com>.
- [6] J. Edmonds and K. Pruhs, “Broadcast scheduling: when fairness is fine”, ACM/SIAM Symposium on Discrete Algorithms. 421 – 430, 2002.
- [7] T. Erlebach and A. Hall, “Hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow”, ACM/SIAM Symposium on Discrete Algorithms, 194 – 202, 2002.

- [8] R. Gandhi, S. Khuller, Y. Kim and Y-C. Wan, “Approximation algorithms for broadcast scheduling”, International Integer Programming and Combinatorial Optimization Conference, 425- 438, 2002.
- [9] B. Kalyanasundaram, and K. Pruhs, “Speed is as powerful as clairvoyance”, *Journal of the ACM*, **47**(4), 617 – 643, 2000.
- [10] B. Kalyanasundaram, K. Pruhs, and M. Velauthapillai, “Scheduling broadcasts in wireless networks”, *Journal of Scheduling*, **4**(6), 339 – 354, 2000.
- [11] C. Kenyon, N. Schabanel and N. Young, “Polynomial-time approximation schemes for data broadcast”, ACM Symposium on Theory of Computing, 659-666, 2000.
- [12] J. Kurose and K. Ross, *Computer Networking: A Top Down Approach Featuring the Internet*, Addison Wesley Longman, 2001.
- [13] C. Phillips, C. Stein, E. Torng, and J. Wein “Optimal time-critical scheduling via resource augmentation”, *Algorimica*, **32**(2), 163 – 200, 2002.