Bandwidth Allocation with Preemption*

Amotz Bar-Noy† Ran Canetti‡ Shay Kutten§ Yishay Mansour¶ Baruch Schieber‖

April 13, 1997

Abstract

Bandwidth allocation is a fundamental problem in the design of networks where bandwidth has to be reserved for connections in advance. The problem is intensified when the overall requested bandwidth exceeds the capacity and not all requests can be served. Furthermore, acceptance/rejection decisions regarding connections have to be made online, without knowledge of future requests. We show that the ability to *preempt* (i.e., abort) connections while in service in order to schedule "more valuable" connections substantially improves the throughput of some networks. We present bandwidth allocation strategies that use preemption and show that they achieve *constant competitiveness* with respect to the throughput, given that any single call requests at most a constant fraction of the bandwidth. Our results should be contrasted with recent works showing that non-preemptive strategies have at most inverse logarithmic competitiveness.

---

*An extended summary of this work appears in the proceedings of the 27th ACM Symp. on Theory of Computing, pages 616–625, 1995.

†Electrical Engineering Department, Tel Aviv University, Tel Aviv, Israel. Email: amotz@eng.tau.ac.il.
Research of this author was supported in part by a grant of the Israeli Ministry of Science and Technology. Part of this work was done while the author was at IBM Research Division, T.J. Watson Research Center.

‡IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY.
Email: canetti@watson.ibm.com. Part of this work was done while the author was at Laboratory of Computer Science, MIT, and supported by the American-Israeli Binational Science Foundation grant 92-00226.

§Dept. of Industrial Engineering, Technion, Haifa, Israel. Email: kutten@ie.technion.ac.il. Part of this work was done while the author was at IBM Research Division, T.J. Watson Research Center.

¶Computer Science Department, Tel Aviv University, Israel. Email: mansour@cs.tau.ac.il. Research of this author was supported in part by The Israel Science Foundation administered by The Israel Academy of Science and Humanities and by a grant of the Israeli Ministry of Science and Technology.

‖IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY
Email: sbar@watson.ibm.com.

1 Introduction

Bandwidth allocation is one of the most important problems in the management of networks that have guaranteed bandwidth policy (e.g., ATM [5], PARIS [9], IBM BBNS [10]). In such networks the application has to reserve in advance sufficient bandwidth for its communication. The guaranteed bandwidth policy is contrasted with the more traditional policy (e.g., TCP/IP), where information packets are routed as they come to the network without prior knowledge about the connections. The advantages of the guaranteed bandwidth policy are many. For instance: (i) Bounded latency for real-time tasks. (ii) Fairness, e.g., one user cannot overtake the entire network resources. (iii) Simple pricing; the application can be charged for the bandwidth it allocated. The major drawback of the guaranteed bandwidth policy is inefficiency: the communication links may be under-utilized. Thus, a "good" bandwidth allocation strategy is crucial for such networks.

The bandwidth allocation problem becomes more difficult in view of the large variety of different applications that use the network simultaneously. These applications have very different requirements in terms of bandwidth, duration, delay, information loss, *etc.* Furthermore, since the communication volume may be much larger than buffer space, decisions regarding current requests cannot be delayed and have to be made without knowledge of future requests. The problem is further intensified when the available bandwidth cannot accommodate all requests for bandwidth, and some have to be rejected.

In this context it is natural to consider the possibility of "softening" the rigidity of the guaranteed bandwidth policy by allowing preemption (i.e., abortion) of connections in service, in order to schedule "more valuable" connections that would otherwise be rejected. Preempting a connection has obvious disadvantages: all the work that was done so far may be lost, and the transmission, in some applications, has to start again later. However, the ability to preempt certain types of connections as a policy may greatly improve the performance of the network. Understanding the power of preemption in this context may shed new light on the value of the guaranteed bandwidth policy.

Indeed, some types of connections should never be preempted (e.g., phone calls). However, there exist other scenarios where preemption is acceptable, and even essential. For instance, a high priority (say, real-time) connection should be allowed to preempt a low priority connection. (See [18] for an implementation of a channel where this type of preemption is used.) Another case is preempting other connections of the same user [19] or of different users based on a pricing model [21]. The willingness of users to communicate over preemptable, low priority, connections would depend on the price discount they would receive, compared to a non-preemptable connection. Our work shows that preemptable connections

allow far better utilization of the network, and thus may be charged at a considerably lower rate.

In this work, we concentrate on preemption as a tool for enhancing the throughput, or the utilization, of the network. We develop various preemption strategies (specifying *when* and *which connections* to preempt) for maximizing the throughput *of connections that eventually complete.* Our strategies perform provably and significantly better (in terms of throughput) than any non-preemptive strategy for bandwidth allocation.

We study two models for bandwidth allocation. The first one is a single link, where requests for connections (or calls) arrive one by one as time proceeds; each call has duration and bandwidth requirements (specified in advance). Requests have to be either served immediately or rejected (for example, due to limited buffer space). This model is an abstraction of a single virtual path in an ATM network that has a single entry and exit. A virtual path in an ATM network serves as a "highway" that is used by many virtual circuits (i.e., connections) simultaneously. The bandwidth of the virtual path has to be divided among the various virtual circuits. The second model is a line of processors where each connection has source, destination and required bandwidth. Here we assume that all requests are for permanent connections (or, alternatively, that all calls arrive at the same time, in some arbitrary order, and have the same duration). This model can be viewed as an abstraction of a virtual path with multiple entries and exits.

As suggested by the online nature of the problem (decisions on current requests are made without knowledge of future requests), we use competitive analysis to measure the performance of bandwidth allocation algorithms. We define the performance of a bandwidth allocation algorithm on a sequence of requests as the throughput of completed calls (i.e., calls that are neither rejected or preempted). This throughput can be measured as the sum, over completed calls, of the duration times the bandwidth requirement; or equivalently as the integral over time of the bandwidth used by calls that eventually complete. The competitiveness of an algorithm is the *infimum* over all request sequences of the ratio of the performance of the algorithm to the performance of the best (offline) schedule for this sequence.

The competitiveness of our algorithms depends only on δ, the ratio between the largest bandwidth requested by a *single* call and the capacity of the line. We note that whereas the capacity of the network can be arbitrarily large, the ratio δ is typically a constant smaller than one. In both models we achieve constant competitiveness if δ is a constant smaller than one. We contrast our results with the fact that non-preemptive bandwidth allocation strategies have logarithmic competitiveness. (We elaborate on this point in the sequel.)

Our algorithms are simple and efficient, however surprising and non-intuitive at first.

They suggest the following approach to bandwidth allocation: in deciding which calls to reject or preempt, the algorithms consider only the duration of a call and the time which a call in service has been already running, *completely ignoring the throughput of calls.* In particular, a call with very large throughput may be preempted in order to make room for a call with longer duration and much smaller throughput. Still, our algorithms achieve constant competitiveness (if δ is bounded away from one), where more straightforward strategies fail.

We show that our algorithms have optimal competitiveness, up to a small constant, for all values of δ. Furthermore, we show that: (i) Deterministic algorithms have very poor competitive ratio if a single call may request the entire bandwidth (that is, if $\delta = 1$). (ii) In the line model, if we let calls have arbitrary duration, and let δ be more than half, then constant competitiveness cannot be achieved by any deterministic algorithm. (Bounds on the competitiveness of *randomized* preemptive bandwidth allocation algorithms in this and related models are shown in [8].)

We also consider a special case of the single link model where all calls have identical bandwidth, which is $1/k$ of the capacity of the link for some integer k. This model can be visualized as k parallel links, each of unit capacity. Even for this restricted model, called the parallel links model, we show that any deterministic online algorithm has a competitive ratio of at most 0.66 (the bound holds *for all* k). The parallel links model is closely related to online preemptive task scheduling under overload [7, 6, 14, 15, 22, 20]. Our impossibility result applies to this problem as well.

Our work extends previous work of Garay and Gopal [12], and Garay *et al.* [13]. These papers also consider online bandwidth allocation with preemption on networks with line topology. However, they greatly simplify the model by assuming that the bandwidth requirement of each call is equal to the bandwidth of the links (and thus, in particular, only one call can be served at a time).

Considerable work has been recently done on non-preemptive online bandwidth allocation (also referred to as call control); we mention here only some of this work. In [2] (and also [1]) the problem of non-preemptive online bandwidth allocation and virtual circuit routing on an arbitrary network was considered. Under the assumption that no call may request more than a logarithmic fraction of the bandwidth, Awerbuch, Azar and Plotkin [2] and Awerbuch *et al.* [1] presented a strategy with competitiveness inversely proportional to the logarithm of the size of the network times the ratio between the largest and smallest value of a call, and proved that no online strategy has better performance. *Randomized* algorithms for non-preemptive call control on tree-like networks are given in [3], where competitiveness inversely logarithmic in similar parameters is shown with matching impossibility results.

Other topologies are considered in [4] with similar results. Lipton and Tomkins [16] considered non-preemptive online "interval scheduling" in a model similar to our line model. They achieved slightly worse than logarithmic competitive ratio and showed that no online algorithm can achieve a logarithmic competitive ratio. In summary, whenever preemption is not allowed the "logarithmic barrier" seems to be unbreakable, even by randomized algorithms.

Finally, we remark that Faigle and Nawijn [11] also showed that preemption is a useful tool. They considered the special case in which all calls have identical bandwidth. However, their goal was to minimize the number of rejected calls (while our goal is to maximize the utilization). They described an optimal deterministic online algorithm (competitiveness 1) when preemption is allowed whereas it was known before that without preemption no algorithm with constant competitiveness exists .

Organization. In Section 2 we introduce the single link model and define bandwidth allocation algorithms in this model. In Section 3 we describe our algorithms for the single link model. In Section 4 we introduce the line model and show how our algorithms can be adapted to this model. In Section 5 we demonstrate the optimality of our algorithms.

2 The single link model

Consider a communication link with bandwidth capacity $\mathcal{B}$ (where $\mathcal{B}$ may be very large). A call is a connection established between the two endpoints of the link. Each call c is characterized by the bandwidth required b_c, the request issue time t_c, and the duration d_c (known in advance). Let $e_c \stackrel{\text{def}}{=} t_c + d_c$ be the ending time of a call c. For simplicity we assume that the time is discrete. Our convention is that the minimum bandwidth requested by a call is 1. Let δ denote the maximum fraction of the capacity used by a single call. We have $\frac{1}{\mathcal{B}} \leq \delta \leq 1$.

The requests arrive one by one in an online fashion. A request for a call c can be either served immediately or rejected. The algorithm may preempt (i.e., stop, or abort) calls during service. The operation of the algorithm run by the control center can thus be described as follows. Upon a request for a call c, if serving c does not violate the bandwidth capacity of the link, then serve c. Otherwise, either reject the request, or preempt some calls that are currently being served so that call c can be served within the capacity of the link.

Let the throughput of a call c, denoted by v_c, be its bandwidth times its duration, i.e. $v_c = b_c \cdot d_c$. Once a call is completed, a value equal to the throughput of the call is gained. No gain is accrued for preempted calls. Note that the throughput of a completed call is a

measure for the amount of information contained in it.

We use competitive analysis to measure the performance of our bandwidth allocation algorithms. The competitive ratio of an algorithm is the infimum over all possible request sequences, of the total throughput of the algorithm on a sequence divided by the total throughput of the best (offline) algorithm on this sequence. More formally, for a sequence $S = c_1, \ldots, c_n$ of call requests, let the bandwidth requested by S at time t be

$$\tilde{B}_S(t) \stackrel{\text{def}}{=} \sum_{\{c \in S | t \in [t_c \ldots t_c + d_c]\}} b_c.$$

We say that S is feasible if $\tilde{B}_S(t) \leq \mathcal{B}$ for all times t. Let the cover of a sequence S be $V(S) \stackrel{\text{def}}{=} \sum_t B_S(t)$, where $B_S(t) \stackrel{\text{def}}{=} \min\{\mathcal{B}, \tilde{B}_S(t)\}$. If S is feasible then we say that $V(S)$ is the throughput of S (that is, the cover equals the throughput). Note that if S is feasible, then $V(S) = \sum_{c \in S} v_c$.

For an algorithm $\mathcal{A}$ and sequence S, let $A(S) \subseteq S$ be the sequence of calls completed by $\mathcal{A}$ on input S. (We use $S' \subseteq S$ to denote that S' is a subsequence of S). Algorithm $\mathcal{A}$ is a valid bandwidth allocation algorithm if for any sequence of requests S, $A(S)$ is feasible. Algorithm $\mathcal{A}$ is ρ-competitive if

$$\rho \leq \inf_S \min_{\{S' \subseteq S | S' \text{ feasible}\}} \frac{V(A(S))}{V(S')}.$$

Note that $0 \leq \rho \leq 1$, and the closer ρ is to one the better the algorithm performs.

Remark: Say that algorithm $\mathcal{A}$ is strongly ρ-competitive if

$$\rho \leq \inf_S \frac{V(A(S))}{V(S)}.$$

Since the link cannot serve more than $\mathcal{B}$ bandwidth at any given time, the cover is an upper bound on the throughput of any feasible subsequence of S. Thus, any strongly ρ-competitive algorithm is also ρ-competitive. We show that our algorithms are strongly competitive.

3 Bandwidth allocation on a single link

We present two algorithms for bandwidth allocation on a single link. The first algorithm, called the "left-right" (LR) algorithm is $\frac{1-2\delta}{2}$-competitive for $\delta < \frac{1}{2}$. The second algorithm, called the "effective time" (EFT) algorithm, is $\frac{1-\delta}{4}$-competitive for all $\delta < 1$. For $\delta > \frac{1}{3}$ algorithm EFT has better competitiveness, whereas for $\delta < \frac{1}{3}$ algorithm LR has better competitiveness. In particular, for very small values of δ, the LR algorithm is about $\frac{1}{2}$-competitive. (In Section 5 it is shown that for small δ no algorithm can have a better competitive ratio than 0.66.) We stress that although $\mathcal{B}$ can be arbitrarily large, the

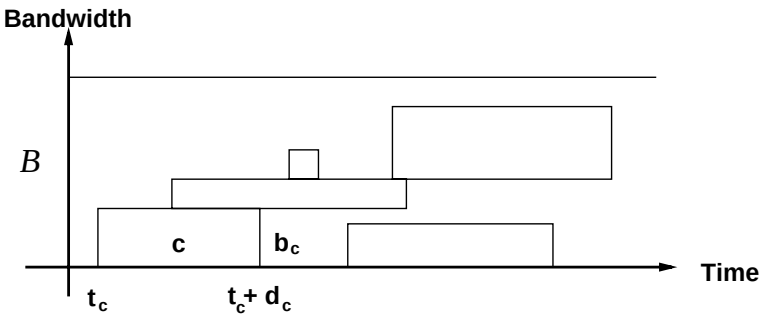

Figure 1: A geometric representation of calls on a single link.

fraction δ is typically a constant smaller than one. Thus our algorithms typically have constant competitiveness. Furthermore, our algorithms do not depend on δ. Therefore, their performance on any request sequence depends on the δ fraction *of this particular sequence.*

Our algorithms, however simple, may seem surprising and non-intuitive at first. In particular, they consider only the duration of a call, and the time "invested" in a call so far (i.e., the time for which a call in service has been running). The algorithms completely ignore the throughput of a call, even though the gain accrued is the sum of the throughputs of completed calls; thus, a call with very large throughput may be preempted in order to make room for a longer duration call with much smaller throughput. In Subsection 3.1 we briefly and informally explain why some straightforward strategies for bandwidth allocation fail. We present our algorithms in the two subsections that follow.

The following geometric representation of the scenario may be helpful (see Figure 1). Let the x axis represent time and the y axis represent bandwidth. Each call c is a rectangle of length d_c and height b_c. We have to fit the rectangles above the x axis, and below the line $y = \mathcal{B}$, under the constraint that the rectangle of call c has to start at $x = t_c$. Note, however, that a call need not use the same "bandwidth pieces" for all its duration. Thus we are allowed to "break" the rectangles vertically, as long as enough bandwidth is allocated at all times. (This distinguishes our problem from many other problems, e.g., memory allocation.)

3.1 First tries

We first show that basing a bandwidth allocation strategy on the throughput of the calls is a bad idea, even though we want to maximize the total throughput of completed calls. This applies both to a greedy strategy (e.g., always prefer calls with larger throughput), and to a "double the gain" strategy (e.g., serve an incoming call if enough bandwidth

can be freed by only preempting calls whose combined throughput is at most half of the throughput of the incoming call). Consider the following request sequence. First, $\sqrt{\mathcal{B}}$ calls are requested at time 0, each of bandwidth $\sqrt{\mathcal{B}}$ and duration $\sqrt{\mathcal{B}}$. Next, still at time 0, $\mathcal{B}$ calls, each of bandwidth 1 and duration $\frac{\mathcal{B}}{2}$ are requested. Serving any of the last $\mathcal{B}$ calls requires preempting one of the first $\sqrt{\mathcal{B}}$ calls. However, each of the last calls has smaller throughput than each of the first calls. Thus, both the greedy and the "double the gain" algorithms serve only the first $\sqrt{\mathcal{B}}$ calls, gaining $\mathcal{B}^{3/2}$. The best schedule is the last $\mathcal{B}$ calls with throughput $\frac{\mathcal{B}^2}{2}$, thus the competitiveness is at most $\frac{2}{\sqrt{\mathcal{B}}}$. The "moral" is that calls with longer duration are preferable *even if the longer calls have smaller throughput,* as there may be many similar calls coming in the future.

An alternative strategy may thus be to consider the duration of calls (say, use a "double the duration" algorithm). It turns out that considering only the duration is also not good enough (we omit further counter-examples). An additional parameter should be considered, namely the amount of time a call in service has been running (or, alternatively, the amount of time it will run in the future). Each of our algorithms considers a different combination of these parameters.

3.2 The "left-right" algorithm

The "left-right" algorithm implements a compromise between the need to hold on to jobs that have been running for the longest time (thus capitalizing on work done) and the need to hold on to jobs that will run for the longest time in the future (thus guaranteeing future work). The compromise is simple: half of the capacity is dedicated to each of these two classes of jobs. Surprisingly, this simple compromise yields a good competitive ratio. The algorithm, denoted LR, is described in Figure 2.

Theorem 1 *For $\delta < \frac{1}{2}$, algorithm* LR *is $(\frac{1-2\delta}{2})$-competitive.*

Proof: First, note that the total bandwidth required by the calls in $L \cup R$ at any time t is at most $B_L(t) + B_R(t) \leq \frac{\mathcal{B}}{2} + \frac{\mathcal{B}}{2} = \mathcal{B}$. Thus the algorithm is valid. Next we show its competitiveness.

Consider an input sequence $S = c_1, \ldots, c_n$ of call requests and assume that $\delta \stackrel{\text{def}}{=} \max_i\{\frac{b_{c_i}}{\mathcal{B}}\}$ is at most $\frac{1}{2}$ (otherwise the competitive claim in the theorem is vacuous). Let $E \stackrel{\text{def}}{=} \text{LR}(S)$ be the set of calls completed by LR on input S. Let S_i be the prefix of S composed of the first i calls in S, and let E_i be the set of calls completed by the algorithm assuming that the input sequence is only S_i. Note that E_i is the union of two sets: (i) the set of calls completed up to time t_{c_i} (that is, up to the time call c_i is requested), and (ii) the set of calls being served at time t_{c_i}. Below we show, by induction on i, that for

Let F be the set of calls currently in service. Upon the request of a call c do:

1. Add c to F.
2. Find the following two sets of calls, L and R:
   (a) Sort the calls by *increasing* order of *starting* time, and let L be the maximal set of calls at the top of the list (i.e., earliest starting times) such that the total bandwidth required by the calls in L is at most $\frac{\mathcal{B}}{2}$.
   (b) Sort the calls by *decreasing* order of *ending* time, and let R be the maximal set of calls at the top of the list (i.e., latest ending times) such that the total bandwidth required by the calls in L is at most $\frac{\mathcal{B}}{2}$.
3. Preempt/reject calls that are neither in L nor in R to fit in the link capacity.

Figure 2: Algorithm LR

all i and for all times t, $B_{E_i}(t) \geq \min\{B_{S_i}(t), (\frac{1}{2}-\delta)\mathcal{B}\}$. Since $S = S_n$ and $E = E_n$, we have $B_E(t) \geq (\frac{1}{2}-\delta)B_S(t)$ for all t. (The worst case is when $B_S(t) = \mathcal{B}$.) Thus the total throughput of the LR algorithm, $V(E)$, is at least $(\frac{1}{2}-\delta)V(S)$, and the theorem follows.

The inductive claim trivially holds for $i = 0$. Let $i > 0$, and fix some $t' \geq 0$. We distinguish two cases.

Case 1: No call p that requested bandwidth for time t' was rejected or preempted in the ith step (that is, when processing the ith request). In this case, if the ith request, c_i, requests bandwidth for time t' then $B_{E_i}(t') - B_{E_{i-1}}(t') = b_c \geq B_{S_i}(t') - B_{S_{i-1}}(t')$. If c_i does not request bandwidth for time t' then $B_{S_i}(t') - B_{S_{i-1}}(t') = 0 = B_{E_i}(t') - B_{E_{i-1}}(t')$. Thus the inductive claim holds.

Case 2: There exist calls that requested bandwidth for time t' and were rejected or preempted in the ith step. We show that in this case, $B_{E_i}(t') \geq (\frac{1}{2}-\delta)\mathcal{B}$. Let p be a call that was rejected or preempted in the ith step, for which $t' \in [t_p \ldots e_p]$. (Otherwise, $B_{E_i}(t')$ is not affected by call p.) Since the bandwidth requested by p is at most $\delta\mathcal{B}$, both $B_L(t_{c_i})$ and $B_R(t_{c_i})$ must be at least $(\frac{1}{2}-\delta)\mathcal{B}$ (otherwise, p would be in either L or R). Furthermore, $p \notin L$, thus we must have $B_L(t) \geq (\frac{1}{2}-\delta)\mathcal{B}$ for all times $t \in [t_p \ldots t_{c_i}]$ (in other words all the calls in L started before t_p, and are still running at t_{c_i}). Similarly, since $p \notin R$, we have $B_R(t) \geq (\frac{1}{2}-\delta)\mathcal{B}$ for all times $t \in [t_{c_i} \ldots e_p]$. (in other words all the calls in R end after e_p, and started running before t_{c_i}). Thus, $B_{E_i}(t') = B_{L\cup R}(t') \geq (\frac{1}{2}-\delta)\mathcal{B}$. □

3.3 The "effective time" algorithm

The "effective time" (EFT) algorithm implements a different compromise between banking on past profit and insuring future profit. Rather than dividing the bandwidth between the two classes, this algorithm attaches a time-value, called the effective time, to each single call. Calls with later effective times are preempted first. The effective time τ_c of a call c is its arrival time minus its duration, i.e. $\tau_c \stackrel{\text{def}}{=} t_c - d_c$. This strategy reflects (in a way described below) the idea that work that is done is worth twice as much as work to be done. Algorithm EFT is described in Figure 3.

Let the effective time of a call c be $\tau_c \stackrel{\text{def}}{=} t_c - d_c$.
Maintain a list F of the calls in service, sorted by increasing order of effective time.
Upon the request of a call c, do:

1. Add c to F in place.
2. Reject/preempt calls from the end of the list (i.e., "latest" effective times), to fit in the link capacity.

Figure 3: Algorithm EFT

The effective time strategy may be viewed as a variant of a "doubling strategy" as follows. Let r be a requested call and c be a call in service. By preempting c and serving r, we "gain" the time interval $[e_c \ldots e_r]$, and "lose" the time interval $[t_c \ldots t_r]$. We will preempt c to make room for r if the time-gain is more than twice the loss, that is if $e_r - e_c > 2(t_r - t_c)$. This condition is equivalent to $t_r - d_r < t_c - d_c$, or $\tau_r < \tau_c$.

Theorem 2 *For $\delta < 1$, algorithm* EFT *is $(\frac{1-\delta}{4})$-competitive.*

Proof: The validity of algorithm EFT is immediate from the description. We show its competitiveness. Unlike algorithm LR, here there may be times where the optimal schedule has the link used to capacity while algorithm EFT has almost no bandwidth allocated to calls that eventually complete. Therefore, we use a different "book-keeping" method for proving the competitiveness of EFT.

Consider some input sequence $S = c_1, \ldots, c_n$ of call requests, and let $\delta \stackrel{\text{def}}{=} \max\{\frac{b_{c_i}}{\mathcal{B}}\}$. Let $E \stackrel{\text{def}}{=} \text{EFT}(S)$ be the set of calls completed by algorithm EFT on input S. We introduce a sequence, E', of "virtual calls", and show that (i) $V(E) \geq \frac{1-\delta}{4} \cdot V(E')$, and (ii) $V(E') \geq V(S)$. Therefore, the total throughput of the algorithm (i.e., $V(E)$) is at least $\frac{1-\delta}{4} \cdot V(S)$.

We define virtual calls as follows. For each call c, let the virtual call c' have arrival time $t_{c'} = \tau_c = t_c - d_c$, ending time $e_{c'} = e_c + 2d_c$, and bandwidth $b_{c'} = \frac{1}{1-\delta} b_c$. For a set A of

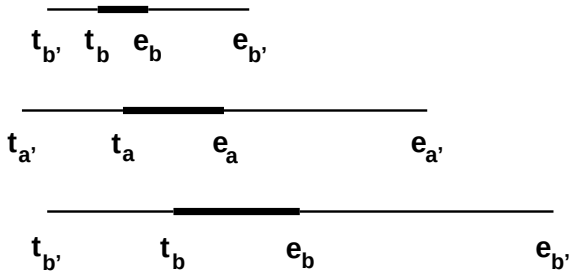

Figure 4: The proof of the technical claim.

calls, let $A' \stackrel{\text{def}}{=} \{c' : c \in A\}$. Note that the throughput of each virtual call c' is $\frac{4}{1-\delta}$ times the throughput of the corresponding real call c. Thus, $V(E) \geq \frac{1-\delta}{4} \cdot V(E')$. In Lemma 4 below we show that $B_{E'}(t) \geq B_S(t)$ for all t. Thus, $V(E') \geq V(S)$, and the theorem follows. □

Let us first prove a technical claim. (See Figure 4.)

Claim 3 *Let a and b be two calls scheduled by the algorithm at time t.*
(1) *If $t_{a'} \leq t_{b'}$ and $t_a \geq t_b$, then $e_{a'} \geq e_{b'}$.*
(2) *If $t_{a'} \leq t_{b'}$, then $e_{a'} \geq e_b$.*

Proof:
Part **(1).** If $t_a \geq t_b$ and $t_{a'} \leq t_{b'}$, then it must be that $d_a \geq d_b$. Thus,

$$e_{a'} = t_a + 3d_a \geq t_b + 3d_b = e_{b'} \ .$$

Part **(2).** Consider the case not covered in Part (1), i.e, $t_a < t_b$ and $t_{a'} \leq t_{b'}$. Call a has not ended by time t_b, thus $d_a \geq t_b - t_a$. Also, it follows from $t_{a'} \leq t_{b'}$ that $d_a \geq d_b - (t_b - t_a)$. Thus,

$$\begin{aligned} e_{a'} &= t_a + d_a + 2d_a \\ &\geq t_a + d_b - (t_b - t_a) + 2(t_b - t_a) \\ &= t_b + d_b = e_b \ . \end{aligned}$$

□

Lemma 4 *For any request sequence S and for all times t we have $B_{E'}(t) \geq B_S(t)$.*

Proof: Say that a time t is i-quiet if, up to and including the ith request, all the calls that requested bandwidth for time t (that is, calls c such that $t \in [t_c \ldots e_c]$) were completed by the algorithm. Note that if time t is i-quiet, it is also j-quiet for all $1 \leq j \leq i$. A time t is quiet if it is i-quiet for all i. If a time t is quiet then certainly $B_{E'}(t) > B_E(t) = B_S(t)$.

It remains to deal with times that are not quiet (i.e, times that are not i-quiet for some request i). Define S_i and E_i as in the proof of Theorem 1. (That is, S_i is the prefix of S composed of the first i calls in S, and E_i is the set of calls completed by the algorithm assuming that the input sequence is only S_i.) We show, by induction on i, that $B_{E'_i}(t) = \mathcal{B}$, for all times t that are not i-quiet. Since $E = E_n$ and $S = S_n$ we get $B_{E'}(t) \geq B_S(t)$ for all times t.

The inductive claim holds vacuously for $i = 0$. For $i > 0$ we distinguish three cases.

Case 1: The ith request, c_i, was served without preempting other calls. In this case, any time that is not i-quiet is also not $(i-1)$-quiet, and $B_{E'_i}(t) \geq B_{E'_{i-1}}(t)$ for all times t. Thus the claim follows from the induction hypothesis.

Case 2: The ith request was rejected without preempting other calls. In this case, the inductive hypothesis holds for any time t that is not $(i-1)$-quiet. Also, all times that are $(i-1)$-quiet and are not i-quiet are in the time range $[t_{c_i} \ldots e_{c_i}]$. Thus it is enough to show that $B_{E'_i}(t) = \mathcal{B}$ for all $t \in [t_{c_i} \ldots e_{c_i}]$. Let F be the set of calls being served at time t_{c_i}. Since c_i is rejected, and $b_{c_i} \leq \delta\mathcal{B}$, the total bandwidth of the calls in F is at least $(1-\delta)\mathcal{B}$. We have $t_{f'} \leq t_{c'_i}$ for all $f \in F$, thus by Claim 3 Part (2) we have $e_{c_i} \leq e_{f'}$. It follows that $B_{E'_i}(t) \geq B_{F'}(t) \geq \mathcal{B}$ for all times $t \in [t_{c_i} \ldots e_{c_i}]$. (In fact this holds for all times $t \in [t_{c'_i} \ldots e_{c_i}]$.)

Case 3: Calls were preempted while processing request i. Let P_i be the set of calls that were preempted (or rejected) while processing request i.[1] For a set A of calls let t_A (resp., e_A) denote the earliest starting time (resp., latest ending time) of a call in A. Here we have to consider the time range $[t_{P'_i} \ldots e_{P'_i}]$ (rather than only $[t_{P_i} \ldots e_{P_i}]$) since calls in P_k that contributed to $B_{E'_{i-1}}(t)$ are now preempted.

It can be seen, similarly to the proof of Case 2, that $B_{E'_i}(t) \geq \mathcal{B}$ for all times $t \in [t_{P'_i} \ldots e_{P_i}]$. It is left to show that $B_{E'_i}(t) \geq \mathcal{B}$ for all times in $[e_{P_i} \ldots e_{P'_i}]$ that are not i-quiet.

Let t be a time in $[e_{P_i} \ldots e_{P'_i}]$ that is not i-quiet. Thus there exists a request $j \leq i$ that caused either rejection or preemption of a call that requested bandwidth at time t. We complete the proof by showing, for each request $j \leq k \leq i$, a set $G_k \subseteq E_k$ of calls such that $B_{G'_k}(t) \geq \mathcal{B}$. These sets are defined inductively, as follows. Let F_k denote the set of calls in service immediately after request k was processed. Let $G_j \stackrel{\text{def}}{=} F_j$. For $k > j$, if c_k is rejected, or if the effective time of c_k is greater than the effective times of all the calls in G_{k-1}, then $G_k = G_{k-1} - P_k$. Else, $G_k = G_{k-1} \cup \{c_k\} - P_k$. Certainly, $G_k \subseteq E_k$. We

[1] It is possible that calls are preempted and also the incoming call is rejected. For simplicity we include the rejected call with the preempted calls.

show by induction on k that the following three properties hold for all $j \leq k \leq i$. (We are interested in Property 3.)

Property 1: All calls in $G_k \cap F_k$ have earlier effective times than all calls in $F_k - G_k$.

Property 2: For all calls $g \in G_k$, $t \in [t_{g'} \ldots e_{g'}]$.

Property 3: $B_{G'_k}(t) = \mathcal{B}$.

Consider the case $k = j$. Property 1 holds since $G_j = F_j$. When request j was processed a call asking bandwidth for time t was rejected or preempted, therefore by Claim 3 all calls in $F_j = G_j$ have Property 2. Furthermore, the total bandwidth of all calls in G_j must be at least $(1 - \delta)\mathcal{B}$, otherwise one of the calls that was either rejected or preempted while processing request j would have been kept by the algorithm. Property 3 now follows from Property 2.

Now fix some $k > j$, and assume that all three properties hold for G_{k-1}. Note that c_k is added to G_k if and only if it is in F_k and $\tau_g > \tau_{c_k}$ for some $g \in G_{k-1}$. Thus Property 1 holds for G_k. To show Properties 2 and 3 we distinguish three subcases:

Case 3.1: Call c_k is rejected. In this case $G_k = G_{k-1} - P_k$. Property 2 holds since $G_k \subseteq G_{k-1}$. Property 3 is shown as in the base case $k = j$.

Case 3.2: Call c_k is served and $\tau_g \leq \tau_{c_k}$ for all $g \in G_{k-1}$. Also here $G_k = G_{k-1} - P_k$. Property 2 holds since $G_k \subseteq G_{k-1}$. Property 3 holds since no call in G_{k-1} is preempted (i.e., $G_k = G_{k-1}$).

Case 3.3: Call c_k is served and $\tau_g > \tau_{c_k}$ for some $g \in G_{k-1}$. Here, $G_k = G_{k-1} \cup \{c_k\} - P_k$. We first show Property 2. From the induction hypothesis, Property 2 holds with respect to all calls in G_k other than c_k. Note that $t_g \leq t_{c_k}$. By our assumption also $\tau_{c_k} \leq \tau_g$, and thus it follows from Claim 3 Part (1) that $t_{c'_k} \leq t_{g'}$ and $e_{g'} \leq e_{c'_k}$. Since $t \in [t_{g'} \ldots e_{g'}]$, we have $t \in [t_{c'_k} \ldots e_{c'_k}]$, which proves Property 2 for call c_k.

Next we show that G_k satisfies Property 3. If $P_k \cap G_{k-1} = \emptyset$ then $G_{k-1} \subseteq G_k$ and Property 3 holds. Otherwise, a call in G_{k-1} was preempted while processing request k. From Property 1 (for $k - 1$) it follows that all calls not in G_{k-1} that were in service upon the arrival of request k must have been preempted as well. Consequently, $F_k = G_k$. However, the total bandwidth of calls in F_k must be at least $(1 - \delta)\mathcal{B}$, otherwise one of the calls that was either rejected or preempted while request k was processed would have been kept by the algorithm. Thus, the total bandwidth of calls in G_k must be at least $(1 - \delta)\mathcal{B}$. By Property 2, all calls in G_k request bandwidth for time t. Property 3 follows. □

4 Bandwidth allocation on a line within a single time slot

In this section we consider bandwidth allocation in the line model, defined as follows. A sequence of stations are connected on a line by communication links. For simplicity we assume that all links have unit length and the same bandwidth capacity, $\mathcal{B}$. Our results generalize to networks with individual capacities and lengths of the links. A call is a connection between two stations on the line. We consider the case where all calls arrive at the same time unit (in some arbitrary order), and have the same duration. In the next section we show that if we let calls have arbitrary duration, and if δ is more than half, then constant competitiveness cannot be achieved by any deterministic algorithm. (We note that this holds even if randomization is allowed [8].)

More formally, the line model is defined as follows. Assume the stations are labeled by consecutive integers increasing from left to right. A call c is characterized by its left endpoint l_c, its right endpoint r_c and its bandwidth requirement b_c. Let the length of call c be $h_c \stackrel{\text{def}}{=} r_c - l_c$. The throughput of c is now $v_c = h_c \cdot b_c$. Again, bearing in mind that our goal is to maximize the throughput of the network, we define the additive throughput accrued from serving a completed call c to be its throughput v_c. Here the throughput can also be regarded as the amount of network resources allocated for the call. The definitions of the cover, feasibility and throughput of a request sequence, as well as the competitiveness and validity of bandwidth allocation algorithms are similar to those of the single link model (Section 2), where time t is replaced, in the natural way, by location (link) e on the line, and earlier times are translated to locations with smaller label. This model is a generalization of the single link model, in the sense that any algorithm for the line model is valid, and has the same performance, in the single link model (when "location" is translated to "time"). The converse is not true, due to reasons described in the sequel.

We adapt our algorithms to this more general scenario. The adapted LR algorithm, called ALR, is still $\frac{1-2\delta}{2}$-competitive, for $\delta < \frac{1}{2}$. The adapted EFT algorithm, called AEFT, is $\frac{1-\delta}{2\phi+1}$-competitive for all $\delta < 1$, where $\phi = 1 + \frac{1}{\phi} = \frac{1+\sqrt{5}}{2} \approx 1.6$ is the golden ratio. The first difficulty in adapting our algorithms to this model is as follows. In the single link model there is a "current time" at which all calls in service require bandwidth and where the new call must start; thus all bandwidth conflicts are at the current time. In the line model bandwidth conflicts upon the arrival of a new call are not limited to a single location; at some locations, it may seem beneficial to accept the incoming call, where at other locations it may seem beneficial to reject. It turns out that the following technique provides a sufficient solution for this difficulty in both adapted algorithms. Upon the arrival of a request, we first add the requested call to the list of calls in service. Next, we go over

the links one by one, in an arbitrary order, and resolve remaining conflicts on each link by rejecting/preempting calls according to a scheme similar to the original one (where "time" is replaced by "location"). The adapted left-right algorithm requires no further modifications. The adapted effective time algorithm encounters an additional difficulty, described in the sequel.

4.1 The adapted left-right algorithm

The adapted left-right algorithm is described in Figure 5.

Let LR* be the same algorithm as LR, where time is translated to location.
Let F be the set of calls currently in service. Upon the request of a call c do:

1. Add c to F.
2. While there are links e with a conflict (i.e., $\tilde{B}_F(e) > \mathcal{B}$) do:

   Let e be a link with a conflict, and let $F_e \subseteq F$ be the set of calls using e. Run algorithm LR* on F_e and reject/preempt calls as instructed.

Figure 5: Algorithm ALR

Theorem 5 *For $\delta < \frac{1}{2}$, algorithm* ALR *is $(\frac{1-2\delta}{2})$-competitive.*

Proof: The proof is similar to the proof of Theorem 1. Consider a link e with a conflict. Let $L_e \subseteq F_e$ and $R_e \subseteq F_e$ be the two sets of calls computed by LR* when run on F_e. First, note that the total bandwidth required by the calls in $L_e \cup R_e$ is at most $\frac{\mathcal{B}}{2} + \frac{\mathcal{B}}{2} = \mathcal{B}$. Thus the algorithm is valid. Next we show competitiveness.

Consider an input sequence $S = c_1, \ldots, c_n$ of call requests and assume that $\delta = \max_i\{\frac{b_{c_i}}{\mathcal{B}}\}$ is at most $\frac{1}{2}$. Let $E \stackrel{\text{def}}{=} \text{LR}^*(S)$ be the set of calls completed by LR*. Let S_i be the prefix of S composed of the first i calls in S, and let E_i be the set of calls completed by the algorithm assuming that the input sequence is only S_i. Below we show, by induction on i, that for all i and for all links e, $B_{E_i}(e) \geq \min\{B_{S_i}(e), (\frac{1}{2} - \delta)\mathcal{B}\}$. Since $S = S_n$ and $E = E_n$, we have $B_E(e) \geq (\frac{1}{2} - \delta)B_S(e)$ for all e. Thus the total throughput of the ALR algorithm, $V(E)$, is at least $(\frac{1}{2} - \delta)V(S)$. The theorem follows.

The inductive claim trivially holds for $i = 0$. For $i > 0$ we distinguish two cases. Fix some link e.

Case 1: (Identical to the proof of Theorem 1.) No call p that requested bandwidth for link e (i.e., $e \in [l_p \ldots r_p]$) was rejected or preempted in the ith step (that is, when processing the ith request). In this case, if the ith request, c_i, requests bandwidth for link e then

$B_{E_i}(e) - B_{E_{i-1}}(e) = b_c \geq B_{S_i}(e) - B_{S_{i-1}}(e)$. If c_i does not request bandwidth for link e then $B_{S_i}(e) - B_{S_{i-1}}(e) = 0 = B_{E_i}(e) - B_{E_{i-1}}(e)$. Thus the inductive claim holds.

Case 2: There exist calls that requested bandwidth for link e and were rejected or preempted in the ith step. We show that in this case, $B_{E_i}(e) \geq (\frac{1}{2} - \delta)\mathcal{B}$. Let $p_1, \ldots, p_a$ be the calls that were rejected or preempted in the ith step, sequenced by the order of their rejection/preemption. We prove, by induction on $1 \leq j \leq a$, that when p_j is either rejected or preempted, the total bandwidth of the calls in service on e is at least $(\frac{1}{2} - \delta)\mathcal{B}$. Suppose that this was the case before the rejection or preemption of p_j. If $e \notin [l_{p_j} \ldots r_{p_j}]$, then this is clearly the case also after the rejection or preemption of p_j. Otherwise, let e' be the link such that p_j was rejected/preempted while LR* was run on $F_{e'}$, and let $L_{e'}$ (resp., $R_{e'}$) be the set L (resp., R) found by LR* when operating on e'. Assume e is to the left of e' (i.e., $e \in [l_{p_j} \ldots e']$). Since $p_j \notin L_{e'}$, and the bandwidth requested by p_j is at most $\delta\mathcal{B}$, then $B_L(e)$ must be no less than $(\frac{1}{2} - \delta)\mathcal{B}$. Similarly, if e is to the right of e' (i.e., $e \in [e' \ldots r_{p_j}]$), then $B_R(e)$ must be no less than $(\frac{1}{2} - \delta)\mathcal{B}$. □

4.2 The adapted effective-time algorithm

The following difficulty is encountered in adapting the effective time method to the line model. The original effective time algorithm weighed differently past work and future work, relying on the fact that time is directional (i.e., the starting time of a new call is no earlier than the starting time of the calls in service). In the line model past and future lose their meaning: the left endpoint of a new call is not necessarily "more to the right" than the left endpoints of the calls in service. Instead, the adapted algorithm will use "effective endpoints" both to the left and to the right, as follows. For a call c with left endpoint l_c, right endpoint r_c, and length $h_c = l_c - r_c$, let the effective span be the range $s_c = [l_c - g \cdot h_c \ldots r_c + g \cdot h_c]$, where g is some constant to be computed in the sequel. We define a complete order, denote $\succ$, on the calls. For calls a and b, $a \succ b$ either if the effective span of a contains the effective span of b (i.e., $s_b \subset s_a$), or if the effective spans of a and b are not contained in each other and a is requested before b. Algorithm AEFT is described in Figure 6.

Theorem 6 *For $\delta < 1$, algorithm* AEFT *is $(\frac{1-\delta}{2m+1})$-competitive, where $m = \max\{g, 1 + \frac{1}{g}\}$, and g is the constant used for determining effective spans.*

Corollary 7 *Let $g = \phi$ be the golden ratio (that is, $\phi = 1 + \frac{1}{\phi}$ and $\phi = \frac{1+\sqrt{5}}{2} \approx 1.6$). For $\delta < 1$, algorithm AEFT is $\frac{1-\delta}{2\phi+1}$-competitive.*

Proof of Theorem 6: We follow the outline of the proof of Theorem 2. (We also use the notation of Theorem 2.) Let $m = \max\{g, 1 + \frac{1}{g}\}$. Define virtual calls as follows.

Let F be the list of calls currently in service. Upon the request of a call c do:

1. Add c to F.
2. While there are links e with a conflict (i.e., $\tilde{B}_F(e) > \mathcal{B}$) do:

   Let e be a link with a conflict, and let $F_e \subseteq F$ be the list of calls using e, sorted so that if $a \succ b$ then a is ahead of b in the list.
   Reject/preempt calls from the end of F_e to fit in the link capacity.

Figure 6: Algorithm AEFT

The virtual call c' that corresponds to a call c has left endpoint $l_{c'} = l_c - m \cdot h_c$, right endpoint $r_{c'} = r_c + m \cdot h_c$ and requires bandwidth $b_{c'} = \frac{1}{1-\delta} b_c$. Consider a sequence S of incoming requests and let $E \stackrel{\text{def}}{=} AEFT(S)$. Clearly, $V(E) \geq \frac{1-\delta}{2m+1} V(E')$. We show below that $B_{E'}(e) \geq B_S(e)$ for all links e. Consequently $V(E') \geq V(S)$, and the algorithm is $\frac{1-\delta}{2m+1}$-competitive.

We show, by induction on the number of calls in the input sequence, that $B_{E'}(e) \geq \mathcal{B}$ for each link e where a call requested bandwidth and was rejected or preempted. Similarly to the proof of theorem 2, the inductive claim is shown by proving that: **(1)** Whenever a call c is *rejected* (when considering a link e), its span is contained in the spans of all the virtual calls that correspond to the calls that remain in service and use link e. (The span of a call c is the range $[l_c \ldots r_c]$.) **(2)** Whenever a call c is *preempted* (when considering a link e), the span of the virtual call c' is contained in the spans of all the virtual calls that correspond to the calls that remain in service and use link e.

Case 1: Suppose that an incoming call c is rejected while considering some link e in Step 2 of algorithm AEFT (given in Figure 6). Here it suffices to show that $l_{f'} \leq l_c$ and $r_c \leq r_{f'}$ for all remaining calls $f \in F_e$. This clearly holds for all calls in F_e whose effective span contains the effective span of c. Consider a call f that remained in F_e whose effective span does not contain the effective span of c. Without loss of generality assume that f is to the right of c (that is, $l_c < l_f < r_c < r_f$). Since c was rejected, we have $f \succ c$, thus the effective span of c does not contain the effective span of call f. Observe that $l_c + gh_c < l_f + gh_f$ (otherwise $r_f + gh_f \geq r_c + gh_c$, implying that the effective span of f contains the effective span of c). This implies that $r_c + gh_c \leq r_f + gh_f$, or

$$gh_c \leq r_f - r_c + gh_f. \tag{1}$$

By our assumption that $l_f < r_c$, we have $r_f - r_c < h_f$. Substituting this inequality in (1) we get $gh_c < (1+g)h_f$, or equivalently

$$h_c < \left(1 + \frac{1}{g}\right) h_f \leq mh_f.$$

Therefore, $l_{f'} = l_f - mh_f \leq l_f - h_c \leq l_c$, and $r_{f'} = r_f + mh_f \geq r_f + h_c \geq r_c$.

Case 2: Suppose that a call p is preempted when a call c is requested (and call c is accepted). We show that in this case $l_{c'} \leq l_{p'}$ and $r_{p'} \leq r_{c'}$. The proof that the spans of the virtual calls that correspond to the rest of the calls that remain in service contain the span of call c' is the same as in Case 1. We have $c \succ p$, since call c is accepted. Since call c was requested after call p the only way for the relation $c \succ p$ to hold is if the effective span of c contains the effective span of p; that is, $l_c - gh_c \leq l_p - gh_p$ and $r_c + gh_c \geq r_p + gh_p$ (and at least one of these inequalities is strict). Since $m \geq g$ then $l_{c'} \leq l_{p'}$ and $r_{p'} \leq r_{c'}$. □

5 Optimality of the algorithms

We prove impossibility results for the competitive ratio of any *deterministic* online bandwidth allocation algorithm *on a single link*, demonstrating that our algorithms have optimal competitiveness for all values of δ, up to a small multiplicative constant. First, we show that if a single call may require the entire bandwidth (i.e., $\delta = 1$), then no algorithm can be more than $\frac{1}{\mathcal{B}}$-competitive. (Note that $\mathcal{B}$ may be unboundedly large.) Next we show that if $\delta \geq \frac{1}{2}$ then no algorithm can be more than $\min\{\frac{1}{8}, (1-\delta)\}$-competitive. In particular, if $\delta = \frac{1}{2}$ then no algorithm can be more than $\frac{1}{8}$-competitive. Recall that algorithm EFT is $\frac{1-\delta}{4}$-competitive, therefore for $\delta = \frac{1}{2}$ EFT is optimal, for $\frac{1}{2} < \delta \leq \frac{7}{8}$ it is at most a factor of $\frac{1}{2(1-\delta)}$ from optimality, and for $\frac{7}{8} \leq \delta < 1$ it is at most a factor of 4 from optimality. For $\frac{1}{3} < \delta < \frac{1}{2}$ the competitiveness of any algorithm is shown to be at most $\frac{1}{2}$, therefore our algorithm is at most a factor of $\frac{2}{1-\delta}$ from optimality. Finally, for $0 < \delta \leq \frac{1}{3}$ the competitiveness of any algorithm is shown to be at most $\frac{2}{3}$, therefore our algorithm is at most a factor of $\frac{4}{3(1-2\delta)}$ from optimality. We summarize our results for the different ranges in the table in Figure 7 and the graph in Figure 8.

We show the impossibility results by demonstrating, for any algorithm, an input sequence of call requests which forces any algorithm to perform poorly comparing to the best (offline) schedule. We describe input sequences as if they may change "on the fly", depending on the choices of the algorithm so far. Such a description is valid since the behavior of a deterministic algorithm on any prefix of the input sequence can be thought of as known beforehand. We sometimes call the sequence creator the *adversary*.

Range	Lower Bound (Algorithm)	Upper Bound (Impossibility Result)	Optimality Ratio $\left(\frac{\text{Upper Bound}}{\text{Lower Bound}}\right)$
$\delta = 1$	$\frac{1}{\mathcal{B}}$	$\frac{1}{\mathcal{B}}$	1
$\frac{7}{8} \leq \delta < 1$	$\frac{1-\delta}{4}$	$1-\delta$	4
$\frac{1}{2} \leq \delta \leq \frac{4}{5}$	$\frac{1-\delta}{4}$	$\frac{1}{8}$	$\frac{1}{2(1-\delta)} \leq 2.5$
$\frac{1}{3} \leq \delta \leq \frac{1}{2}$	$\frac{1-\delta}{4}$	$\frac{1}{2}$	$\frac{2}{1-\delta} \leq 4$
$0 < \delta \leq \frac{1}{3}$	$\frac{1-2\delta}{2}$	$\frac{2}{3}$	$\frac{4}{3(1-2\delta)} \leq 4$
$\delta \to 0$	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{4}{3}$

Figure 7: Table of all bounds in the different ranges of $0 < \delta \leq 1$.

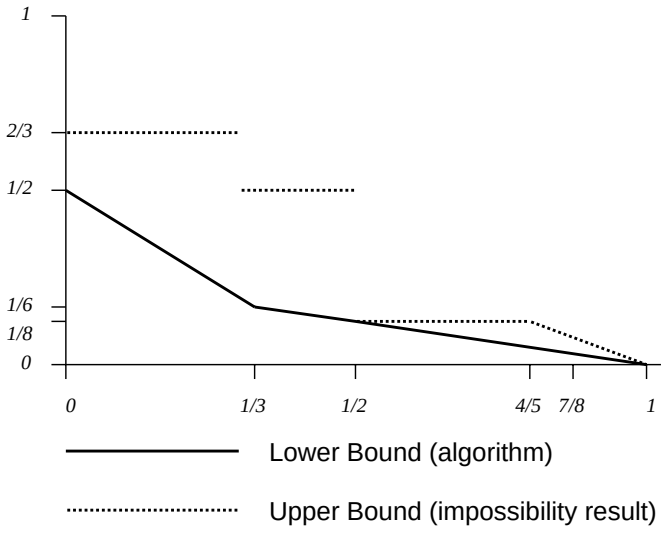

Figure 8: Graph of bounds in the different ranges of $0 < \delta \leq 1$.

This section is organized as follows. First, we describe the known impossibility results for the case in which calls always request the whole bandwidth. Next we prove our impossibility result for the case $\delta = 1$. Using the techniques of these two bounds we show the $\frac{1}{8}$ bound for the case $\frac{1}{2} \leq \delta < 1$. Next we prove the $1 - \delta$ bound for the case $\frac{1}{2} < \delta < 1$. We proceed for the case $0 \leq \delta < \frac{1}{2}$ and prove our bounds using the more restricted parallel links model (to be defined there). At the end of the section, we consider the more general case of calls with arbitrary durations on a general linear network. We show that any deterministic online algorithm for this case is at most $\frac{1}{\mathcal{B}}$-competitive, if $\delta > \frac{1}{2}$.

The whole bandwidth case:

The sequence S is an adaptation of a sequence demonstrated in [13] to show that no algorithm can be more than $\frac{1}{4}$-competitive for the case where all calls require bandwidth *exactly* $\mathcal{B}$. (The sequence in [13] is an adaptation of a bound for scheduling algorithms in the presence of overload, shown in [6].) Let us shortly review their construction. For any $\alpha > \frac{1}{4}$, construct a sequence $C = c_1, \ldots, c_n$ of calls where each call c_i arrives "just before call c_{i-1} ends" (i.e., $t_{c_i} = e_{c_{i-1}} - 1$, where $e_{c_i} \stackrel{\text{def}}{=} t_{c_i} + d_{c_i}$ is the ending time of c_i), and: (a) $d_{c_{i-1}} < \alpha e_{c_i}$ for all i, (b) $d_{c_n} \leq d_{c_{n-1}}$. The offline algorithm is able to schedule calls that cover the entire duration $0 \leq t \leq e_i$, in a way described below. Since $d_{c_{i-1}} < \alpha e_{c_i}$ for all i, the online algorithm is always forced to preempt the single call in service (that is, c_{i-1}) when a new call arrives, otherwise it is not α-competitive. Thus the algorithm completes either only c_n or only c_{n-1} with value at most $d_{c_{n-1}}$ whereas the offline can cover the entire duration $0 \leq t \leq e_{c_n}$. Consequently, once the call c_n is requested the online algorithm can no longer be α-competitive. To complete the construction set $d_{c_1} = \mathcal{B}$ and interleave the c_i calls with many additional service calls, each of duration 1. The service calls cannot be served by the online algorithm since once a service call is served the sequence stops and the algorithm is no longer competitive. The offline algorithm serves all service calls and accrues throughput e_{c_n}.

The case $\delta = 1$:

We show that if $\delta = 1$ then for any online algorithm $\mathcal{A}$ there exists an input sequence S such that the throughput $V(A(S)) = \mathcal{B}^2$, whereas some feasible subsequence $S' \subseteq S$ has a gain $V(S') = \mathcal{B}^3$. Consequently, the competitiveness of $\mathcal{A}$ is at most $\frac{1}{\mathcal{B}}$.

The sequence S consists of two types of calls: squares and slices. A square call has duration $\mathcal{B}$ time units and bandwidth $\mathcal{B}$. A slice call has duration $\mathcal{B}^2$ time units and bandwidth 1. Both have throughput $\mathcal{B}^2$. Observe that no algorithm can serve both a

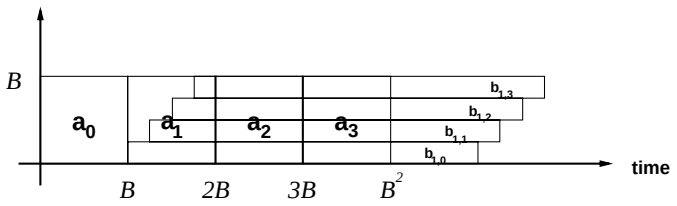

Figure 9: The impossibility result for the case $\delta = 1$.

square and a slice call that intersect (that is, require bandwidth for the same time). The sequence S starts with a square call at time 0. The algorithm $\mathcal{A}$ must serve this call, otherwise it is 0-competitive. Next, the following procedure is repeated until S contains either $\mathcal{B}$ squares calls or $\mathcal{B}$ slice calls:
(a) As long as $\mathcal{A}$ serves a square call then at each time unit a slice call is requested.
(b) As long as $\mathcal{A}$ serves a slice call, then every $\mathcal{B}$ time units a square call is requested.
(See Figure 9.)
Analysis: At the end of the process A has completed at most one call, hence $V(A(S)) \leq \mathcal{B}^2$. On the other hand, a feasible $S' \subseteq S$ may contain $\mathcal{B}$ calls of the same type (either squares or slices) and be of value $V(S') = \mathcal{B} \cdot \mathcal{B}^2 = \mathcal{B}^3$.

The case $\delta = \frac{1}{2}$:

Consider the sequence $C = c_1, \ldots, c_n$ as described above for a given $\alpha > \frac{1}{4}$. In the sequel we will omit references to α, and think of the sequence which is associated with $\alpha = \frac{1}{4} + \varepsilon$ where ε is negligible. Rigorous proof follows in a straightforward way. Although we present the bound only for $\delta = \frac{1}{2}$, it will be clear from the presentation that the bound holds for any $\delta > \frac{1}{2}$.

The basic idea behind our proof is to force the algorithm to serve c_i with bandwidth $\mathcal{B}/2$ while the offline can serve calls that cover the whole bandwidth until e_{i+1}. This will add an extra factor of 2 and therefore the 1/8 bound.

In the sequence C, let the length of call c_i be γ_i and let its beginning time and ending time be τ_i and ϵ_i respectively. Define a new call c_{n+1} with parameters $\tau_{n+1} = \tau_n$, $\gamma_{n+1} = \gamma_n$ and $\epsilon_{n+1} = \epsilon_n$, and choose $\mathcal{B} \gg n \cdot \epsilon_n$. We construct a sequence S that contains three types of calls. First, for each i there are two calls f_i with bandwidth $\frac{1}{2}\mathcal{B}$, arrival time $t_i = \mathcal{B}^3\tau_i$, ending time $e_i = \mathcal{B}^3\epsilon_i$ and duration $d_i = \mathcal{B}^3\gamma_i$. Next, many service calls of type a_i with duration $\mathcal{B}^3(2\epsilon_{i+1})$ and bandwidth 1, and many service calls of type b with duration $\mathcal{B}$ and bandwidth $\frac{1}{2}\mathcal{B}$. The starting time of the service call, would depend on the behavior of the algorithm, as explained later.

Recall that v_c is the throughput of call c. We get that, $v_b = \frac{\mathcal{B}^2}{2}$, $v_{a_i} = \mathcal{B}^3(2\epsilon_{i+1}) = o(\mathcal{B}^4)$, and $v_{f_i} = \frac{1}{2}\mathcal{B}d_i = \frac{\mathcal{B}^4}{2}\gamma_i = O(\mathcal{B}^4)$. Note also that since $\epsilon_n \cdot n \ll \mathcal{B}$ it follows that $\sum_{j=1}^{n} v_{a_j} = o(\mathcal{B}^4)$. These equations and the fact that

$$\frac{d_i}{e_{i+1}} = \frac{\gamma_i}{\epsilon_{i+1}} \to \frac{1}{4},$$

imply the following proposition.

Proposition 1

$$\frac{v_b + \sum_{j=1}^{i} v_{a_j}}{v_{f_i}} \ll \frac{1}{8}. \tag{1}$$

$$\frac{v_b + v_{f_i} + \sum_{j=1}^{i} v_{a_j}}{\mathcal{B}v_{a_i}} \leq \frac{2v_{f_i} + \sum_{j=1}^{i} v_{a_j}}{\mathcal{B}v_{a_i}} = \frac{\gamma_i}{2\epsilon_{i+1}} + \frac{\sum_{j=1}^{i} v_{a_j}}{\mathcal{B}^4\epsilon_{i+1}} \to \frac{1}{8}. \tag{2}$$

$$\frac{v_{f_i} + \sum_{j=1}^{i+1} v_{a_j}}{\mathcal{B}e_{i+1}} = \frac{\gamma_i}{2\epsilon_{i+1}} + \frac{\sum_{j=1}^{i+1} v_{a_j}}{\mathcal{B}^4\epsilon_{i+1}} \to \frac{1}{8}. \tag{3}$$

Say that we are in the ith phase if the following three conditions hold.

Condition 1. The algorithm has not completed any calls of type b or f_j for $1 \leq j < i$ before time t_i.

Condition 2. The algorithm serves one call of type f_i and maybe also one call of type a_j for $1 \leq j \leq i$.

Condition 3. There exists a feasible subsequence $S' \subseteq S$, consisting only of calls of type b, that covers the entire bandwidth for times in $[0 \ldots t_i]$.

The sequence S starts by requesting an f_1 call and an a_1 call, both at time 0. The algorithm must serve the f_1 call in order to be competitive. Hence, we are now in the first phase. We show below how to force the algorithm to reach the $(i+1)$st phase from the ith phase. The bound will follow once the nth phase is reached. At that stage, the gain of the online algorithm is at most $v_{f_n} + \sum_{j=1}^{n+1} v_{a_j} = v_{f_{n-1}} + \sum_{j=1}^{n+1} v_{a_j}$ while the gain of the offline is $\mathcal{B}e_n$. Equation (3) of Proposition 1 proves the $\frac{1}{8}$ bound.

Once in the ith phase, the $(i+1)$st phase is reached as follows.

1. At each time t, $t_i \leq t \leq e_i - \mathcal{B}$, request two calls of type b. In order to serve any of these calls, the algorithm has to preempt either the f_i call or the a_i call currently in service.

   (a) Suppose that the call f_i is preempted. In this case we stop. The algorithm serves a call of type b and calls of type a_j for $1 \leq j \leq i$, whereas the offline can serve an f_i call. The bound is yielded by Equation (1) of Proposition 1.

(b) Suppose that the call a_i is preempted. Request calls of type a_i at times $t+1, t+2, \ldots$ until either the algorithm preempts the f_i call, or preempts the b type call, or rejects $\mathcal{B}$ calls of type a_i. The first case is the same as the previous case (a). In the second case we are back in the ith phase and we continue offering the calls of type b. In the third case we stop. The algorithm serves a call of type b, a call of type f_i, and some calls of type a_j for $1 \leq j < i$, whereas the offline can serve $\mathcal{B}$ calls of type a_i. The bound is yielded by Equation (2) of Proposition 1.

We thus assume that none of these b calls are served. Note that now there exists a feasible subsequence $S' \subseteq S$, consisting only of b calls, which covers the entire bandwidth for times in $[0 \ldots e_i]$.

2. At time $e_i - \mathcal{B}$, when the f_i call is about to end, request two calls of type f_{i+1}. Now, the algorithm has several options:

   (a) The algorithm rejects both calls. In this case we stop. The offline can serve b type calls to cover the whole bandwidth up to t_{i+1} and the two f_{i+1} calls to cover the whole bandwidth up to e_{i+1}, whereas the algorithm can serve one f_i call and some a_j type calls for $1 \leq j \leq i$. The bound follows from Equation (3) of Proposition 1.

   (b) The algorithm serves one of the f_{i+1} calls and preempts the f_i call. In this case we reached the $(i+1)$st phase.

   (c) The algorithm serves one of the f_{i+1} calls and preempts the a_i call. This means that at this stage, the algorithm serves a call of type f_i, and a call of type f_{i+1}. In this case, request calls of type a_{i+1} at times $e_i - \mathcal{B}, e_i - \mathcal{B} + 1, \ldots$ until either the algorithm preempts the f_i call, or preempts the f_{i+1} call, or rejects $\mathcal{B}$ calls of type a_{i+1}. In the first case we reach the $(i+1)$st phase. The second case is the same as the previous case (a) in which the algorithm rejects the two f_{i+1} calls. In the third case we stop. Since $v_{f_i} < v_{f_{i+1}}$ it follows that the throughput of the algorithm is at most $2v_{f_{i+1}} + \sum_{j=1}^{i} v_{a_j}$, whereas the throughput of the offline can be $\mathcal{B}v_{a_{i+1}}$. The bound is yielded by Equation (2) of Proposition 1.

   (d) The algorithm serves both f_{i+1} calls. Again we request calls of type a_{i+1} at times $e_i - \mathcal{B}, e_i - \mathcal{B} + 1, \ldots$ until either the algorithm preempts one of the f_{i+1} calls or rejects $\mathcal{B}$ calls of type a_{i+1}. Both cases appeared in the previous case (c).

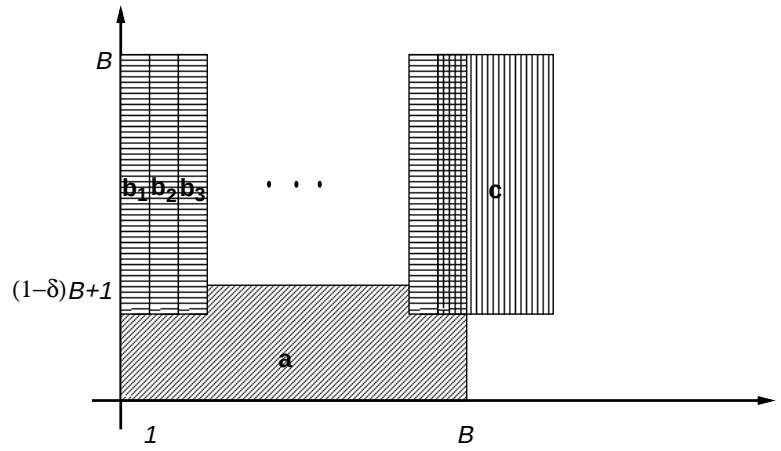

Figure 10: The impossibility result for the case $\frac{1}{2} < \delta < 1$. Here, $\mathcal{B} = 12$ and $\delta = \frac{3}{4}$.

The case $\frac{1}{2} < \delta < 1$

We show that in this case any online algorithm is at most $(1 - \delta + o(1))$-competitive. Note that for $\frac{1}{2} \leq \delta \leq \frac{7}{8}$ the previous bound (of $\frac{1}{8}$) is better. Consider the following request sequence to an online algorithm. (See Figure 10.)

1. Request a call a at time 0, with duration $\mathcal{B}$ time units and bandwidth $(1 - \delta)\mathcal{B} + 1$. If the algorithm does not serve this call, then end the sequence.

2. At each time $0 \leq i < \mathcal{B}$ request a call b_i of duration one and bandwidth $\delta\mathcal{B}$. No b_i call can be served together with a. If the algorithm preempts a to serve one of the b_i calls, then end the sequence.

3. Request a call c at time $\mathcal{B} - 1$ with duration $1 + \frac{1-\delta}{\delta}\mathcal{B}$ and bandwidth $\delta\mathcal{B}$.

Analysis: Note that the value of a is $(1 - \delta)\mathcal{B}^2 + \mathcal{B}$, the value of c is $(1 - \delta)\mathcal{B}^2 + \delta\mathcal{B}$, and the value of any b_i is $\delta\mathcal{B}$. Assume call a is served (otherwise the algorithm is 0-competitive). If the algorithm preempts a to serve some b_i then the algorithm is at most $\frac{\delta\mathcal{B}}{(1-\delta)\mathcal{B}^2+\mathcal{B}} = O\left(\frac{1}{\mathcal{B}}\right)$-competitive. Therefore, we can assume that all the b_i calls are rejected. Calls a and c cannot be both served. Thus, the value gained by the algorithm (by serving either a or c) is at most $(1 - \delta)\mathcal{B}^2 + \mathcal{B}$. The subsequence $S' = \{b_0, \ldots, b_{\mathcal{B}-2}, c\}$ is feasible with $V(S') = \mathcal{B}^2$. Hence, the competitive ratio of the algorithm is at most $\frac{(1-\delta)\mathcal{B}^2+\mathcal{B}}{\mathcal{B}^2} = (1 - \delta) + o(1)$.

The case $\delta < \frac{1}{2}$:

We show that for $\frac{1}{3} < \delta < \frac{1}{2}$ no algorithm can be more than 1/2-competitive, and that for $0 < \delta \leq \frac{1}{3}$ no algorithm can be more than 2/3-competitive. This impossibility result

is shown for a model called the parallel links model. In this model, the bandwidth of all calls is exactly $\frac{\mathcal{B}}{k}$ for some integer $k > 1$. This can be viewed as if the two stations are connected via k parallel links, and each call occupies exactly one link for its duration. (For convenience we assume that calls can transferred between lines during service at no cost.) Clearly, this model is a special case of our model to which we refer as the single link model.

Let r_k be an upper bound on the competitiveness of any online algorithm in the parallel links model. We show that in the single link model no online algorithm can be more than r_k competitive, in case a call may request at most $\delta\mathcal{B}$ bandwidth, for $\delta > \frac{1}{k+1}$. Consider any online algorithm $\mathcal{A}$ for the single link model. Let $\mathcal{A}'$ be the algorithm for the k parallel links model that corresponds to algorithm $\mathcal{A}$. Let $c_1, c_2, \ldots$ be a sequence of calls that causes $\mathcal{A}'$ to be at most r_k competitive in the k parallel links model. We claim that this sequence in which the bandwidth of each c_i is $\delta\mathcal{B}$ also causes $\mathcal{A}$ to be at most r_k competitive in the single link model. To see that note that since $\delta > \frac{1}{k+1}$, it follows that there cannot be $k+1$ calls that are served concurrently. Therefore, at any point of time, the calls in service can be viewed as arranged on k parallel links such that the width of each link is $\delta\mathcal{B}$.

It remains to bound r_k. We remark that there exist algorithms for the parallel links model that have slightly better competitiveness than our algorithms for the single link model. Algorithms for $k = 1$ and $k = 2$ were previously known [13, 6]. A simple extension of them yields algorithms for even k with a competitive ratio of $\frac{1}{2}$ and for odd k with competitive ratio $\frac{1}{2} - \frac{1}{4k}$.

We first describe a simple impossibility result for $k = 2$ (the case $k = 1$ was dealt with in [13], as outlined above). At time 0, request two calls of length x. If the algorithm serves only one call then we are done. At every time unit $0 \leq i \leq x - 1$, request two service calls of length 1. If the algorithm takes one of these service calls it must preempt one of the original calls. In this case, the sequence ends and the algorithm achieves competitiveness of $\frac{x+1}{2x} = \frac{1}{2} + o(1)$. Otherwise, at time $x - 1$ request two new requests of length x. In this case the competitiveness is $\frac{2x}{4x-2} = \frac{1}{2} + o(1)$. For the cases $k = 3$ and $k = 4$ we have an impossibility result of $1/2$ and $11/20$ respectively. However, the proofs are tedious and are omitted.

We now describe the impossibility result for the competitiveness of any bandwidth allocation algorithm in this model for all $k > 1$. We show by induction on k that the competitive ratio for k links is at most r where r satisfies $e^r = \frac{r}{1-r}$, in particular, $r \leq \frac{2}{3}$.

The basic strategy is an extension of the strategy for the two lines case. At time 0, request k calls of length x and at time $x - 1$ request another set of k calls of length x. We refer to these calls as long calls. During the times $[0 \ldots x - 1]$ many service calls (short calls) will be offered in a way that the offline algorithm will be able to cover this range and

therefore will have a throughput of $(2x-1)k$. We will choose $x \gg 1$ and therefore without loss of generality assume that the throughput of the offline is $2xk$. We do not know how to prevent the algorithm from serving all of the service calls and thus to achieve a $1/2$ bound. In what follows we describe how to prevent the algorithm from serving some of the service calls to achieve the $2/3$ bound.

A general scenario of the execution of the online algorithm can be described as follows. The online algorithm starts by serving $w \leq k$ out of the k long calls requested at time 0 (and all the short calls that can be served). In case the adversary does not stop the request sequence, then after t_1 units of time the online algorithm preempts one of the w calls to serve a service call. Again, in case the adversary does not stop the request sequence, then after an additional t_2 units of time it preempts another long call and so on, for $z-w$ times. Note that by definition of $t_1, \ldots, t_{z-w}$ we must have that $\sum_{j=1}^{w-z} t_j \leq x$.

At time $x-1$ the online algorithm remains with z long calls. In case the adversary does not stop the request sequence, k long calls are requested at this time. In this case the throughput of the online algorithm will be xk plus the sum of the lengths of all the service calls it served before time $x-1$.

Assume first a very simple strategy for the adversary. The sequence will have only two types of calls: long calls of length x and short calls of unit length. Assume further that the algorithm is r-competitive. Then the following inequalities must hold:

$$\begin{aligned}
\frac{xw}{xk} &\geq r && (0)\\
\frac{t_1(k-w)+x(w-1)}{xk} &\geq r && (1)\\
\frac{t_1(k-w)+t_2(k-w+1)+x(w-2)}{xk} &\geq r && (2)\\
&\vdots\\
\frac{\sum_{j=1}^{i} t_j(k-w+j-1)+x(w-i)}{xk} &\geq r && (i)\\
&\vdots\\
\frac{\sum_{j=1}^{w-z} t_j(k-w+j-1)+xz}{xk} &\geq r && (w-z)\\
\frac{\sum_{j=1}^{w-z} t_j(k-w+j-1)+xk}{2xk} &\geq r && (k)
\end{aligned}$$

Inequality (0) is true since otherwise the adversary could stop immediately after presenting

the k long calls and thus preventing the algorithm from being r competitive. The rest of the inequalities are true since otherwise the adversary could stop after the jth preemption of a long call by the algorithm.

The value of r is maximized when all the inequalities become equalities. By comparing the $(j-1)$th equality with the jth equality we get that for $1 \leq j \leq w-z$

$$t_j(k-w+j-1) = x \ .$$

Thus Equality (k) is equivalent to

$$\frac{w-z+k}{2k} = r \ .$$

The above equality together with Equality (0) implies that

$$w+z = k \ .$$

In addition, plugging the value of t_j in the equality $\sum_{j=1}^{w-z} t_j = x$ we have

$$\sum_{j=1}^{w-z} \frac{x}{k-w+j-1} = x \ .$$

This is equivalent to

$$\frac{1}{k-w} + \frac{1}{k-w+1} + \cdots + \frac{1}{k-z-1} = 1 \ .$$

Using the estimation $\sum_{j=1}^{n} \frac{1}{j} = \ln n + O(1)$, we get that $\sum_{j=m}^{n} \frac{1}{j} > \sum_{j=m+1}^{n+1} \frac{1}{j} \approx \ln\left(\frac{n+1}{m}\right)$. Substituting $n = k-z-1$ and $m = k-w$, it follows that

$$\ln\left(\frac{k-z}{k-w}\right) < 1 \ .$$

This implies that $\frac{k-z}{k-w} < e$. since $w = k-z$ we get that $\frac{w}{k-w} < e$ and since $w = kr$ we get that $\frac{r}{1-r} < e$ which is equivalent to

$$r < \frac{e}{1+e} \approx 0.731059 < \frac{3}{4} \ .$$

Now, assume a more complicated strategy for the adversary. Note that in the simple strategy the adversary does not try to minimize the number of service calls served by the online algorithm; that is, whenever the online algorithm has a "free" line it can use it to serve service calls. In the more complicated strategy the adversary tries also to minimize the number of service calls served by the online algorithm. This is done by using the strategy "recursively". Whenever the algorithm is serving j long calls and has $k-j$ free lines (for

$w \leq j \leq z$), the adversary offers service calls according to the strategy for $k-j$ parallel links. Namely, instead of offering service calls of a fixed size, the service calls become the long calls for $k-j$ parallel links, while scaling down the length of the service calls for the $k-j$ lines accordingly. Furthermore, we assume by induction that the bound r can be achieved for any number of lines less than k. We have already seen that the inductive claim holds for $k=2$.

The analysis is almost the same as the one for the simpler adversary. Here, we add a factor of r in the inequalities as follows:

$$\frac{\sum_{j=1}^{i} t_j(k-w+j-1)r + x(w-i)}{xk} \geq r \ .$$

Again, the value of r is maximized when all the inequalities become equalities. This implies that

$$t_j(k-w+j-1)r = x$$

which yields as before the equality

$$w+z=k \ .$$

In addition, we get that

$$\frac{1}{k-w} + \frac{1}{k-w+1} + \cdots + \frac{1}{k-z-1} = r$$

and therefore

$$\ln\left(\frac{k-z}{k-w}\right) < r \ .$$

Using similar arguments as before, we get that

$$r < \frac{e^r}{1+e^r} \ .$$

This inequality does not hold for $r > 0.66$. Hence, we proved that

$$r < \frac{2}{3} \ .$$

Line networks with arbitrary durations:

Finally, we consider networks with a line topology in which the calls have arbitrary duration (rather than unit time duration). In this case the throughput of a call is the product of its length, bandwidth and duration. Suppose that the line consists of $n+1$ stations, all the links have the same bandwidth $\mathcal{B}$, and that the maximum allowed bandwidth of a single call is $\delta\mathcal{B}$, for $\delta > \frac{1}{2}$. We show that any deterministic online algorithm for this model is at most $1/n$-competitive. The sequence and the analysis is as in the case of the single link model

with $\delta = 1$, where instead of bandwidth $\mathcal{B}$ we consider a line of nodes $0, \ldots, n$. Specifically, we simulate the procedure of the adversary presented in the impossibility result for the single link model with $\delta = 1$. In the simulation, whenever a square call a_i is requested in the procedure for the single link model with $\delta = 1$, the adversary here requests a square call of bandwidth $\delta\mathcal{B}$ at time in that connects nodes 0 and n, for n time units. Note that the throughput of this call is $n^2\delta\mathcal{B}$. Similarly, instead of requesting the j-th slice call b that intersects the square call a_i as described in the impossibility result for the single link model with $\delta = 1$, the request made here is for a call at time $in + j$ that connects nodes j and $j + 1$ for n^2 time units. Again, the throughput of this call is also $n^2\delta\mathcal{B}$. Since $\delta > \frac{1}{2}$, a square call and a slice call that intersect cannot both be served. Thus, following the analysis for the single link case with $\delta = 1$ we get the $1/n$ upper bound.

Acknowledgments

We are indebted to Hugo Krawczyk for very helpful discussions at the early stages of this work.

References

[1] B. Awerbuch, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. Online load balancing with applications to machine scheduling and virtual circuit routing. In *Proc. of the 25th ACM Symp. on Theory of Computing*, pages pages 623–631, 1993.

[2] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive of online routing. In *Proc. 34th IEEE Symp. on Foundations of Computer Science*, pages 32–40, 1993.

[3] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén. Competitive non-preemptive call control. In *Proc. of the 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 312–320, 1994.

[4] B. Awerbuch, R. Gawlick, T. Leighton and Y. Rabani. Online Admission Control and Circuit Routing for High Performance Computing and Communication. In *Proc. 35th IEEE Symp. on Foundations of Computer Science*, pages 412–423, 1994.

[5] Special Issue on Asynchronous Transfer Mode. *Int. Journal of Digital and Analog Cabled Systems*, 1(4), 1988.

[6] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang. On the competitiveness of online real-time task scheduling. In *Proc. 32nd IEEE Symp. on Real Time Systems*, pages 106–115, 1991.

[7] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha. Online scheduling in the presence of overload. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 101–110, 1991.

[8] R. Canetti and S. Irani. On the Power of Preemption in Randomized Scheduling. In *Proc. of the 27th ACM Symp. on Theory of Computing*, pages 606–615, 1995. To appear in *SIAM Journal on Computing.*

[9] I. Cidon and I. Gopal. PARIS: An approach to integrated high-speed private networks. *Int. Journal of Digital and Analog Cabled Systems*, 1(2):77–86, 1988.

[10] P. F. Chimento, J. E. Drake, L. Gum, W. A. Hervatic, C. P. Immanuel, G. A. Marin, R. O. Onvural, S. A. Owen and T. E. Tedijanto. Broadband Network Services for High Speed Multimedia Networks. IBM Publication, IBM – Networking Systems Architecture, P.O. Box 12195, Triangle Research Park, NC 27709.

[11] U. Faigle and W. M. Nawijn, Note on scheduling intervals online. *Discrete Applied Mathematics,* Vol. 58, pages 13–17, 1995.

[12] J. A. Garay and I. S. Gopal. Call preemption in communication networks. In *Proc. INFOCOM 92*, pages 1043–1050, 1992.

[13] J. A. Garay, I. S. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient online call control algorithms. In *Proc. 2nd Israel Conf. on Theory of Computing and Systems*, pages 285–293, 1993.

[14] G. Koren and D. Shasha. D-over: An Optimal On-line Scheduling Algorithm for Overloaded Real-Time Systems. *SIAM Journal on Computing*, Vol. 24, pages 318–339, 1995.

[15] G. Koren and D. Shasha. MOCA: A Multiprocessor On-line Competitive Algorithm for Real-Time System Scheduling. *Theoretical Computer Science, Special Issue on Dependable Parallel Computing*, No. 128, pages 75–97, 1994.

[16] R. J. Lipton and A. Tomkins. On-line Interval Scheduling. In *Proc. of the 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 302–311, 1994.

[17] C. Lund, S. Phillips and N. Reingold. IP Over Connection-Oriented Networks and Distributional Paging. In *Proc. 35th IEEE Symp. on Foundations of Computer Science*, pages 424–434, 1994.

[18] K.K. Ramakrishnan, L. Vaitzblit, C. Gray, U. Vahalia, D. Ting, P. Tzelnic, S. Glaser, W. Duso. Operating System Support for a Video-On-Demand File Service. In D. Shepherd, G .Blair, G. Coulson, N. Davies, F. Garcia (eds.), *Proc. Network and Operating System Support for Digital Audio and Video: 4th International Workshop*, Springer-Verlag, Lecture Notes in Computer Science 846, pages 216–227, 1994.

[19] N. Shacham Preemption based admission control in multi media multi party communications. In *Proc. INFOCOM 95*, pages 827–834, 1995.

[20] D.B. Shmoys, J. Wein, and D. P. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, Vol. 24, pages 1313–1331, 1995.

[21] F. Toutain and O. Huber A general Preemption-based admission control policy using smart market approach. In *Proc. INFOCOM 96*, pages 794–801, 1996.

[22] F. Wang and D. Mao. Worst case analysis for on-line scheduling in real-time systems. TR 91-54, Dept. and Computer and Information Science, University of Massachusetts at Amherst, 1991.