Bandwidth Allocation with Preemption

Amotz Bar-Noy* Ran Canetti† Shay Kutten* Yishay Mansour‡ Baruch Schieber*

February 1, 1995

Abstract

Bandwidth allocation is a fundamental problem in the design of networks that allocate, in advance, bandwidth to connections. The problem is intensified when and acceptance/rejection decisions on connections have to be done on-line, without knowledge of future requests. We show that the ability to preempt (i.e., abort) connections in service in order to be able to schedule "more valuable" connections substantially improves the overall throughput of the network.

We concentrate on the case where any single call might occupy only a constant fraction of the bandwidth. We present bandwidth allocation strategies that use preemption and show that they achieve *constant competitiveness* with respect to the throughput. Our results should be contrasted with recent works showing that non-preemptive strategies have at most logarithmic competitiveness.

*IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10598.
Email: {amotz,kutten,sbar}@watson.ibm.com.

†Dept. of Applied Mathematics and Computer Science, Weizmann Institute, Israel.
Email: canetti@wisdom.weizmann.ac.il. Part of this research was done while visiting the IBM T.J. Watson Research Center.

‡Computer Science Department, Tel-Aviv University, Israel. This research 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. Email: mansour@cs.tau.ac.il

1 Introduction

Bandwidth allocation is one of the most important problems in the management of networks that have **guaranteed bandwidth** policy (e.g., ATM [ATM88], PARIS [CG88], IBM BBNS [CDG+]). 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., INTERNET), 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: (1) Bounded latency for real-time tasks. (2) Fairness, e.g., one user cannot overtake the entire network resources. (3) 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. In order for the bandwidth allocation strategy to achieve high performance it should be able to deal with competing requests for connections that vary significantly with respect to bandwidth requirement and duration.

In this context it is natural to consider the possibility to "soften" 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 have to be rejected. Preempting a connection has obvious disadvantages: all the work that was done so far may be lost, and the transmission has to start again later. However, the possibility 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 [RVG+93] for an implementation of a channel where this type of preemption is used.) 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 are studying 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). The requests have to be either served immediately or rejected. 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 simultaneously. The bandwidth of the virtual path has to be divided between 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 on-line 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), or equivalently as the integral over time of the bandwidth used by connections that were completed. 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 (off-line) 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. Also, our constants are fairly small, for example for a small δ we show an algorithm that is approximately $1/2$ competitive. Our algorithms are simple and efficient, however surprising and non-intuitive at first. We contrast our results with the fact that non-preemptive bandwidth allocation strategies have logarithmic competitiveness.

We show that our algorithms have optimal competitiveness, up to a small constant, for all values of δ. Furthermore, we show that: (1) Deterministic algorithms have very poor competitive ratio in case a single call may request the entire bandwidth (that is, if $\delta = 1$). (2) 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 [CI94].)

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. Even for this restricted model we show that any deterministic on-line algorithm has a competitive ratio of at most ≈ 0.66 (the bound holds *for all* k). This model is closely related to on-line preemptive task scheduling under overload [BKM+91b, BKM+91a, KS92, WM91, SWW91]. Our impossibility result applies to this problem as well.

Our work extends previous work of [GG92, GGK+93]. These papers also consider on-line bandwidth allocation with preemption on line networks. 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 on-line bandwidth allocation (also referred to as call control); we mention here only some of this work. In [AAP93] (and also [AAF+93]) the problem of non-preemptive on-line 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, they presented a strategy with logarithmic competitiveness, and proved that no on-line strategy has better performance. *Randomized* algorithms for non-preemptive call control on tree-like networks is given in [ABFR94], where logarithmic competitiveness is shown with matching impossibility results. Other topologies are considered in [AGLR94] with similar results. [LT94] considered non-preemptive on-line "interval scheduling" in a model similar to our line model. They achieved slightly worse than logarithmic competitive ratio and showed that no on-line 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.

We remark that [LPR94] study a related problem of managing a connectionless network (e.g.

TCP/IP) on a connection oriented network (e.g., X.25). Their main assumption is that the number of simultaneously open connections is limited, and their main task is to decide when and which connection to close. In their model they assume certain probabilistic behavior of the connection. They do not address the question of connections with different bandwidth.

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). 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 on-line fashion at a centralized location. A request for a call c can be either served immediately or rejected. The algorithm may **preempt** (i.e., stop) calls that where served. 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.

We now define the gain accrued from serving calls, bearing in mind that our goal is to maximize the throughput of the network. 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 the total throughput allocated for the call by the network.

We make use of 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 (off-line) algorithm for 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

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

Let $B_S(t) \stackrel{\text{def}}{=} \min\{\mathcal{B}, \tilde{B}_S(t)\}$. Since the bandwidth cannot be beyond $\mathcal{B}$, we define the **cover** of a sequence S be $V(S) \stackrel{\text{def}}{=} \sum_t B_S(t)$.

We say that S is **feasible** if $\tilde{B}_S(t) \leq \mathcal{B}$ for all times t. If S is feasible, then $V(S) = \sum_{c \in S} v_c$. In this case we say that $V(S)$ is the **throughput** of S.

For an algorithm A and sequence S, let $A(S) \subseteq S$ be the sequence of calls completed by A on input S. (We use the notation $S' \subseteq S$ to denote that S' is a subsequence of S). Algorithm A is a **valid bandwidth allocation** algorithm if for any sequence of requests S, $A(S)$ is feasible. Algorithm 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 to one the better the algorithm performs.

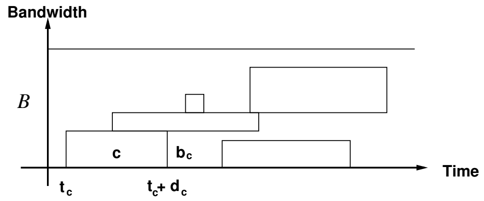

Figure 1: A geometric representation of calls on a single link.

Algorithm A is **strongly ρ-competitive** if

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

The cover of a sequence is an upper bound on the possible throughput of this sequence, and in particular it equals the throughput of any feasible subsequence. Therefore, if an algorithm is strongly ρ-competitive, then it 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 about 0.66.) We stress that although $\mathcal{B}$ can be arbitrarily large, the 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.)

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 the calls that are neither in L nor in R.

Figure 2: Algorithm LR

3.1 First tries

We first show that basing a bandwidth allocation algorithm on the throughput of the calls is a bad idea. This applies both to a greedy strategy (i.e., always prefer calls with larger throughput), and to a "double the gain" strategy (i.e., 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

This "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 *Algorithm* LR *is a* $\frac{1-2\delta}{2}$*-competitive bandwidth allocation algorithm for* $\delta < \frac{1}{2}$.

Proof: First, note that the total bandwidth required by the calls in service (that is, 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.

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 $E \stackrel{\rm def}{=} \mathrm{LR}(S)$ 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. Note that E_i is the union of two sets: (i) the set of calls completed up to time t_{c_i} (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 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 $(\frac{1}{2} - \delta)V(S)$. The theorem follows.

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

Case 1: The ith request, c_i, was served without preempting any previous calls. Let t_{c_i} (resp., e_{c_i}) be the beginning (resp., ending) time of call c_i. For all $t \notin [t_{c_i}..e_{c_i}]$, $B_{S_i}(t) = B_{S_{i-1}}(t)$. Since no call is preempted, $B_{E_i}(t) = B_{E_{i-1}}(t)$ as well. It now follows from the inductive assumption that for all $t \notin [t_{c_i}..e_{c_i}]$, $B_{E_i}(t) \geq \min\{B_{S_i}(t), (\frac{1}{2} - \delta)\mathcal{B}\}$. For $t \in [t_{c_i}..e_{c_i}]$, we have $B_{E_i}(t) - B_{E_{i-1}}(t) = b_{c_i} = B_{S_i}(t) - B_{S_{i-1}}(t)$, thus the inductive claim holds.

Case 2: The ith request was rejected. Let t_l be the latest starting time of a call in L, and let e_r be the earliest ending time of a call in R. From the algorithm, all calls that are preempted or rejected at time t_{c_i} are not in $L \cup R$. Thus, these calls are "contained" in the interval $[t_l..e_r]$. (That is, for each such call f, $t_l \leq t_f$ and $e_f \leq e_r$.) In particular, $t_l \leq t_{c_i} \leq e_{c_i} \leq e_r$. Since no call may request more than $\delta\mathcal{B}$ bandwidth and call c_i is rejected, we have $B_L(t) \geq (\frac{1}{2} - \delta)\mathcal{B}$ for all times $t \in [t_l..t_{c_i}]$ and $B_R(t) \geq (\frac{1}{2} - \delta)\mathcal{B}$ for all times $t \in [t_{c_i}..e_r]$. Thus, $B_{E_i}(t) = B_{L\cup R}(t) \geq (\frac{1}{2} - \delta)\mathcal{B}$, for all times $t \in [t_l..e_r]$. For all $t \notin [t_l..e_r]$, $B_{E_i}(t) = B_{E_{i-1}}(t)$ and $B_{S_i}(t) = B_{S_{i-1}}(t)$, and the claim is implied by the induction hypothesis.

Case 3: The ith request was served while some other calls were preempted. Recall that all the preempted calls are "contained" in the interval $[t_l..e_r]$. Thus, similar to case 2, for all times $t \in [t_l..e_r]$, $B_{E_i}(t) = B_{L\cup R}(t) \geq (\frac{1}{2} - \delta)\mathcal{B}$. For all times $t \in [e_r..e_{c_i}]$, the claim follows since similarly to Case 1, $B_{E_i}(t) - B_{E_{i-1}}(t) = b_{c_i} = B_{S_i}(t) - B_{S_{i-1}}(t)$. For all $t \notin [t_l..e_{c_i}]$, $B_{E_i}(t) = B_{E_{i-1}}(t)$ and $B_{S_i}(t) = B_{S_{i-1}}(t)$, and the claim is implied by the induction hypothesis. □

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 of a call c is $\tau_c \stackrel{\rm def}{=} t_c - d_c$. 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.

The effective time strategy may be reasoned as follows. We employ a "doubling strategy" with respect to the following parameter. Let $e_f = t_f + d_f$ be the ending time of a call f. Let r be a requested call and c be a call in service. By preempting c and serving r, we "gain" the time interval $[e_c..e_r]$, and "lose" the time interval $[t_c..t_r]$. 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 *Algorithm* EFT *is a* $\frac{1-\delta}{4}$*-competitive bandwidth allocation algorithm for all* $\delta < 1$.

The proof of Theorem 2 is delicate. The reader is encouraged to read this proof in Appendix A.

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

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 can be generalized 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 let δ be more than half, then constant competitiveness cannot be achieved by any deterministic algorithm. (We note that this holds even if randomization is allowed [CI94].)

More formally, 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 total network resources that is 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 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 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 and resolve remaining conflicts on this link by rejecting/preempting calls according to a scheme similar to the original one (where "time" is replaced by "location").

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 4: Algorithm ALR

The adapted left-right algorithm requires no further modifications. The adapted effective time algorithm encounters an additional difficulty, described below.

Algorithm ALR. The algorithm is described in Figure 4.

Theorem 3 *Algorithm* ALR *is a* $\frac{1-2\delta}{2}$*-competitive bandwidth allocation algorithm in the line model for* $\delta < \frac{1}{2}$.

Proof (sketch): The proof is similar to the proof of Theorem 1. Note that here an induction step consists of resolving the conflict on a single link (i.e., a single application of Step 2). Thus, several induction steps may be needed to show correctness after each new call. □

Algorithm AEFT. 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, namely that 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 seem to loose their meaning. Consequently, the adapted algorithm will use "effective endpoints" both to the left and to the right. Namely, 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 $s_c = [l_c - g \cdot h_c..r_c + g \cdot h_c]$, where g is some constant computed in the sequel. For calls a and b let "$a > b$" denote the statement "the effective span of a contains the effective span of b" (i.e. $s_b \subset s_a$). We extend this order to a complete order such that $a \succ b$ if $[a > b]$ or $[b \not> a$ and a is requested before $b]$. Algorithm AEFT is described in Figure 5.

Theorem 4 *Algorithm* AEFT *is a* $\frac{1-\delta}{2m+1}$*-competitive bandwidth allocation algorithm for all* $\delta < 1$*, where* $m = \max\{g, 1 + \frac{1}{g}\}$.

Corollary 5 *Set* $g = \phi$*, where* $\phi = 1 + \frac{1}{\phi}$ *is the golden ratio (that is,* $\phi = \frac{1+\sqrt{5}}{2} \approx 1.6$*). Then, algorithm AEFT is* $\frac{1-\delta}{2\phi+1}$*-competitive.*

Proof of Theorem 4 (sketch): We follow the outline of the proof of theorem 2. Let $m = \max\{g, 1 + \frac{1}{g}\}$. Define virtual calls as follows. The virtual call c' of 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 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.

As in Lemma 7, we have to show that: (1) If an incoming call c is rejected when considering some link e in Step 2 above, then $l_{f'} \leq l_c$ and $r_c \leq r_{f'}$ for all remaining calls $f \in F_e$. This is

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 5: Algorithm AEFT

shown as follows. If c is rejected when considering some link e in Step 2 above, then $c \not\succ f$ for all $f \in F_e$. W.l.o.g. assume that $r_f > r_c > l_f > l_c$. Since $c \not\succ f$ then $r_f + gh_f > r_c + gh_c$, and since $r_f - r_c < h_f$ then $gh_c < (1+g)h_f$, which implies that $h_c < mh_f$, since $m \geq 1 + \frac{1}{g}$. Therefore, $l_{f'} \leq l_c$ and $r_c \leq r_{f'}$ for all remaining calls $f \in F_e$. (2) If a call p is preempted then the requested call c satisfies $l_{c'} \leq l_{p'}$ and $r_{p'} \leq r_{c'}$. This is shown as follows. If a call p is preempted then the requested call c satisfies $c > p$. Since $m \geq g$ then $l_{c'} \leq l_{p'}$ and $r_{p'} \leq r_{c'}$. $\square$

5 Optimality of the algorithms

We show impossibility results for the competitive ratio of any *deterministic* on-line 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 $\mathcal{B}$ can be unboundedly large). Next we show that if $\delta > \frac{1}{2}$ then no algorithm can be more than $(1-\delta)$-competitive and 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 while for $\delta > \frac{1}{2}$ is at most a factor of 4 from optimality. For $\delta < \frac{1}{2}$ the competitiveness of any algorithm is shown to be at most 0.66, therefore our algorithm is at most a factor of 5.3 from optimal. Finally, we consider the more general case of calls with arbitrary durations on a general linear network. We show that any deterministic on-line algorithm for this case is at most $\frac{1}{\mathcal{B}}$-competitive, if $\delta > \frac{1}{2}$. 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 (off-line) schedule.

The case $\delta = 1$: We show that if $\delta = 1$ then for any algorithm 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 gain $V(S') = \mathcal{B}^3$. Consequently, the competitiveness of 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 A cannot serve both a 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 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 A serves a square call then at each time unit a slice call is requested. (b) As long as A serves a slice call, then every $\mathcal{B}$ time units a square call is requested.

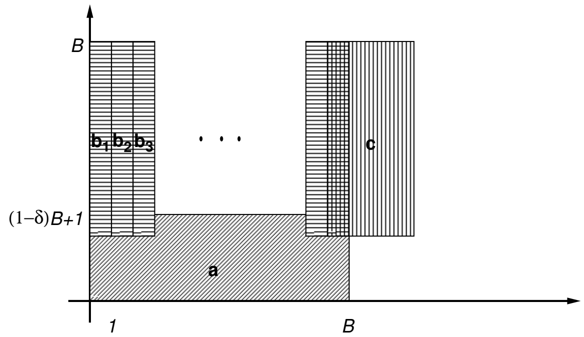

Figure 6: The impossibility result for the case $\frac{1}{2} < \delta < 1$. Here, $\mathcal{B} = 12$ and $\delta = \frac{3}{4}$.

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 $\frac{1}{2} < \delta < 1$: We show that any algorithm A is at most $(1 - \delta + o(1))$-competitive, using the following sequence of calls. (See Figure 6.)

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 A 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 $\frac{1-\delta}{\delta}\mathcal{B}$ and bandwidth $\delta\mathcal{B}$.

Analysis: Assume call a is served (otherwise A is 0-competitive). If A preempts a to serve some b_i then A is at most $\frac{1}{\mathcal{B}}$-competitive. Assume all the b_i calls are rejected. Calls a and c cannot be both served. Thus, the value gained by A (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 $V(S') = \mathcal{B}^2 - \delta\mathcal{B}$. Hence, the competitive ratio of A is at most $\frac{(1-\delta)\mathcal{B}^2+\mathcal{B}}{\mathcal{B}^2-\delta\mathcal{B}} = (1 - \delta) + o(1)$.

The case $\delta = \frac{1}{2}$: See Appendix B.

The case $\delta < \frac{1}{2}$: The impossibility result for this case is shown for a special case of the single link model, called the **parallel links** model. In this model, the bandwidth of all calls is exactly $\frac{\mathcal{B}}{k}$ for some integer $k > 1$. The model and impossibility result are described in Appendix C.

General linear networks and arbitrary duration: We consider the more general case of calls with arbitrary durations on a general linear network. We show that any deterministic on-line algorithm for this case is at most $\frac{1}{\mathcal{B}}$-competitive, if $\delta > \frac{1}{2}$. The proof of this bound is similar to that of the bound for $\delta = 1$, with the exception that "bandwidth" is translated to "time". Details are omitted from this extended abstract.

Acknowledgments

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

References

[AAF+93] B. Awerbuch, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proc. of the 23rd ACM Symp. on Theory of Computing*, pages 623–631, May 1993.

[AAP93] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive of on-line routing. In *Proc. 30th IEEE Symp. on Foundations of Computer Science*, pages 32–40, October 1993.

[ABFR94] 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, Washington, DC, January 1994.

[AGLR94] B. Awerbuch, R. Gawlick, T. Leighton and Y. Rabani. On-line Admission Control and Circuit Routing for High Performance Computing and Communication. In *Proc. 35th IEEE Symp. on Foundations of Computer Science*, November 1994.

[ATM88] Special Issue on Asynchronous Transfer Mode. Int. Journal of Digital and Analog Cabled Systems, 1(4), 1988.

[CDG+] 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. an IBM publication, IBM, Networking Systems Architecture, P.O. Box 12195, Triangle Research Park, NC 27709.

[BKM+91a] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang. On the competitiveness of on-line real-time task scheduling. In *Proc. 32nd IEEE Symp. on Real Time Systems*, pages 106–115, San Antonio, TX, December 1991.

[BKM+91b] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha. On-line scheduling in the presence of overload. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 101–110, San Juan, Puerto Rico, October 1991.

[CG88] 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.

[CI94] R. Canetti and S. Irani. On the Power of Preemption in Randomized Scheduling. Submitted to these proceedings.

[FT94] M. Furst and A. Tomkins. Some lower bounds for call control algorithms. Manuscript, 1994.

[GG92] J.A. Garay and I.S. Gopal. Call preemption in communication networks. In *Proc. INFOCOM 92*, volume 44, pages 1043–1050, Florence, Italy, 1992.

[GGK+93] J.A. Garay, I.S. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithms. In *Proc. 2nd Israel Conf. on Theory of Computing and Systems*, pages 285–293, Netanya, Israel, 1993.

[KS92] G. Koren and D. Shasha. D^{over}: An optimal on-line scheduling algorithm for overloaded real-time systems. Technical Report 594, Courant Institute, New York University, 1992.

[LT94] R.J. Lipton and A. Tomkins. On-line interval scheduling. In *Proc. of the 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 302–311, Washington, DC, January 1994.

[LPR94] 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*, November 1994.

[RVG+93] 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. Network and Operating System Support for Digital Audio and Video: Fourth International Workshop, Lancaster, UK. Springer Verlag, pages 225–236, Nov. 1993.

[ST85] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.

[SWW91] D.B. Shmoys, J. Wein, and D.P. Williamson. Scheduling parallel machines on-line. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 131–140, San Juan, Puerto Rico, October 1991.

[T88] A. S. Tanenbaum. Computer Networks. Prentice Hall, Englewood, NJ, USA, 1988.

[WM91] F. Wang and D. Mao. Worst case analysis for on-line scheduling in real-time systems. Technical Report 91-54, Dept. and Computer and Information Science, University of Massachusetts at Amherst, 1991.

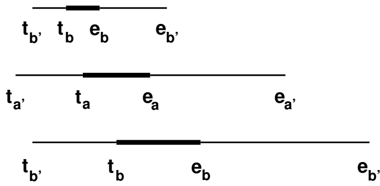

Figure 7: The proof of the technical claim.

A Correctness of algorithm EFT

Theorem 2 *Algorithm* EFT *is a* $\frac{1-\delta}{4}$*-competitive bandwidth allocation algorithm for all* $\delta < 1$.

Proof: The validity of algorithm EFT is immediate from the description. 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 $E \stackrel{\text{def}}{=} \mathrm{EFT}(S)$ be the set of calls completed by algorithm EFT. 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 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 7 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 7).

Claim 6 *Let a and b be two calls scheduled by the algorithm at time t. If $\tau_a \leq \tau_b$, then $e_{a'} \geq e_b$. If in addition $t_a \geq t_b$, then $e_{a'} \geq e_{b'}$.*

Proof: Assume first that $t_a \geq t_b$ (and $\tau_a \leq \tau_b$). Consequently, we have $d_a \geq d_b$. Thus, $e_{a'} = t_a + 3d_a \geq t_b + 3d_b = e_{b'}$. Assume now that $t_a < t_b$. Call a has not ended by time t_b, thus $d_a \geq t_b - t_a$. Also, it follows from $\tau_a \leq \tau_b$ that $d_a \geq d_b + t_a - t_b$. Thus:

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

□

Lemma 7 *For any request sequence S and for all times t, $B_{E'}(t) \geq B_S(t)$.*

Proof: Say that a time t is **quiet** if all the calls that requested bandwidth for time t (that is, calls c such that $t \in [t_c..e_c]$) were completed by the algorithm. If t is quiet then $B_{E'}(t) > B_E(t) = B_S(t)$. It remains to deal with non-quiet times. Define S_i and E_i as in the proof of Theorem 1. We show, by induction on i, that $B_{E'_i}(t) = \mathcal{B}$, for all non-quiet times t. Since $E = E_n$ and $S = S_n$ we get also $B_{E'}(t) \geq B_S(t)$, for all non-quiet times t.

The claim vacuously holds 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, there are no new non-quiet times and the claim follows from the induction hypothesis.

Case 2: The ith request was rejected. To prove this case it is enough to show that $B_{E'_i}(t) = \mathcal{B}$ for all $t \in [t_{c_i}..e_{c_i}]$. This is because all new non-quiet times are in this range, and thus for all other ranges the claim follows from the induction hypothesis. Let F be the set of calls served at time t_{c_i}. Since c_i is rejected, the total bandwidth of the calls in F is at least $(1-\delta)\mathcal{B}$. Certainly, $t_{f'} \leq t_{c_i}$

for all $f \in F$, and by Claim 6 we have $e_{f'} \geq e_{c_i}$. It follows that $B_{E'_i}(t) \geq B_{F'}(t) \geq \mathcal{B}$ for all times $t \in [t_{c_i}..e_{c_i}]$.

Case 3: The call c_i as served while preempting other calls. Let P be the set of calls that were preempted to make room for call c_i. Let $t_P = \min_{p \in P}\{t_p\}$, let $t_{P'} = \min_{p \in P}\{t_{p'}\}$, let $e_P = \max_{p \in P}\{e_p\}$, and let $e_{P'} = \max_{p \in P}\{e_{p'}\}$. Since c_i is served, it follows that $t_{c'_i} \leq t_{P'}$. Since $t_{c_i} \geq t_p$, for every $p \in P$, Claim 6 implies that $e_{P'} \leq e_{c'_i}$. All together we get:

$$t_{c'_i} \leq t_{P'} \leq t_P \leq e_P \leq e_{P'} \leq e_{c'_i} \ .$$

For $t \leq t_{P'}$ and $t \geq e_{P'}$, $B_{E'_i}(t) \geq B_{E'_{i-1}}(t)$, and no new non-quiet times were created. Thus, the claim is implied by the induction hypothesis. For all non-quiet times $t \in [t_{P'}..e_P]$, it can be shown that $B_{E'_i}(t) = \mathcal{B}$, similarly to Case 2.

To complete the proof we show that $B_{E'_i}(t) = \mathcal{B}$, for all non-quiet times $t \in [e_P..e_{P'}]$. Let t be a non-quiet time in $[e_P..e_{P'}]$. Let $j < i$ be such that t_{c_j} is the latest time in which a request asking bandwidth at time t was either rejected or preempted. (Such a j exists since t is non-quiet.) Inductively define subsets $F_k \subseteq E_k$ for $k \geq j$, as follows. For $k = j$, let $F_j = A_j$, where A_k is defined to be the set of served calls at time t_{c_k}. For $k > j$, if c_k was rejected then $F_k = F_{k-1}$. If c_k was served, then there are two possibilities. If the effective time of c_k is greater than the effective times of all the calls in F_{k-1}, then $F_k = F_{k-1}$. Otherwise, $F_k = F_{k-1} \cup \{c_k\} - P$, where P is the set of preempted calls at t_{c_k}. Certainly, $F_k \subseteq E_k$.

We show by induction that the following three properties hold for all F_k, $j \leq k \leq i$. In particular, the third property applied to $k = i$ completes the proof.

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

Property 2: For all calls $f \in F_k$, $t \in [t_{f'}..e_{f'}]$.

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

Assume first that $k = j$. Property 1 holds since $F_j = A_j$. At time t_{c_j} a call asking bandwidth at time t was rejected or preempted, therefore all calls in $A_j = F_j$ have Property 2. Furthermore, the total bandwidth of all calls in F_j must be at least $(1 - \delta)\mathcal{B}$, implying Property 3.

Now fix some $k > j$, and assume that all properties hold for F_{k-1}. If c_k was rejected then $F_k = F_{k-1}$. Therefore, Properties 2 and 3 are implied by the induction hypothesis. Property 1 holds since $A_k \subseteq A_{k-1}$. If c_k was served and its effective time is greater than the effective times of all the calls in F_{k-1}, then again $F_k = F_{k-1}$, implying Properties 2 and 3. Property 1 holds since $c_k \notin F_k$.

The remaining case is when c_k was served and its effective time was smaller than the effective time of at least one call in F_{k-1}, denoted by f. Property 1 holds because of the induction hypothesis for f and because c_k was added to F_k. Note that $F_k \subseteq F_{k-1} \cup \{c_k\}$. Property 2 holds since $t_{c'_k} \leq t_{f'} \leq t$, and since $t_{c_k} \geq t_f$ and thus by Claim 6, $t \leq e_{f'} \leq e_{c'_k}$. Property 3 clearly holds if $P \cap F_{k-1} = \emptyset$. Otherwise, a call in F_{k-1} was rejected to make room for c_k. In this case $A_k \subseteq F_k$ and the total bandwidth of calls in A_k must be at least $(1 - \delta)\mathcal{B}$, implying Property 3. □

B An impossibility result for $\delta = \frac{1}{2}$

We show that if $\delta = \frac{1}{2}$ then for any algorithm A there exists an input sequence S such that $V(A(S)) = \frac{1}{8}\mathcal{B}^3 + \mathcal{B}^2$, whereas some feasible subsequence $S' \subseteq S$ has value $V(S') = \mathcal{B}^3$. Thus A is at most $(\frac{1}{8} + o(1))$-competitive.

The sequence S is an adaptation of a sequence demonstrated in [GGK+93] 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 [GGK+93] sequence is an adaptation of a bound for scheduling algorithms in the presence of overload, shown in [BKM+91a].) Let us shortly review their construction. For any $\alpha < \frac{1}{4}$, they 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 off-line 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 on-line 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 $d_{c_{n-1}}$. Thus the on-line algorithm is not 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 on-line algorithm since once a service call is served the sequence stops and the algorithm is no longer competitive. The off-line algorithm serves all service calls and accrues throughput e_n.

Given such a sequence $C = c_1, \ldots, c_n$ as described above, We construct a sequence S in our scenario as follows. First, for each call $c_i \in C$ we have two identical calls c_i' and c_i'' with bandwidth $\frac{1}{2}\mathcal{B}$, and arrival time and duration set to $\mathcal{B}^2$ times those of c_i. Next, we have many **service** calls of two types: calls of type a have duration $\mathcal{B}^2$ and bandwidth 1, and calls of type b have duration 1 and bandwidth $\frac{1}{2}\mathcal{B}$.

Say that we have reached the ith **breakpoint** at time $T_{c_i'}$ if: (a) the algorithm has completed no previous calls, (b) the algorithm serves call c_i' (or c_i'') and a call of type a, both starting at time $t_{c_i'}$, and (c) there exists a feasible subsequence $S' \subseteq S$, consisting only of calls of type b, that fills the entire bandwidth for times in $[0..t_{c_i'}]$ (and has, thus, value $V(S') = \mathcal{B} \cdot t_{c_i'}$).

The sequence S starts by requesting the call c_1' and an a call, both at time 0. We are now at the first breakpoint. Next, we show below how to reach the $(i+1)$st breakpoint from the ith breakpoint. The bound now follows from the following argument. At breakpoint n the online gain is $\frac{1}{2}d_{c_n'}\mathcal{B}^2 + \mathcal{B}^2$, while the offline has $e_{c_n'}\mathcal{B}$. Since $d_{c_n'}/e_{c_n'} = d_{c_n}/e_{c_n} \to \frac{1}{4}$, the competitive ratio is $\frac{1}{8} + o(1)$.

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

1. At each time $t_{c_i'} \ldots e_{c_i'}$, request two calls of type b. In order to serve any of these calls, the algorithm has to preempt either c_i' or the a call currently in service. If the algorithm preempts either c_i' or the a call then the request sequence will end at that point and the algorithm will be less than $\frac{1}{\mathcal{B}}$-competitive. We thus assume that none of these b calls is served. Note that now there exists a feasible subsequence $S' \subseteq S$, consisting only of b calls, which fills the entire bandwidth for times in $[0..e_{c_i'}]$.

2. When the call c_i' is about to be completed (i.e., at time $e_{c_i} - 1$), request the two calls c_{i+1}' and c_{i+1}''. Now, the algorithm has several options:

   (a) Serve c_{i+1}' (or c_{i+1}'') and preempt c_i'; the a call remains in service. We are now at breakpoint $i+1$. It remains to show that this is the only possibility in which the algorithm doesn't lose, but we generated a new breakpoint.

   (b) Serve c_{i+1}' and preempt the a call (c_i' remains in service). We now continue with many requests for a calls until either c_i' or c_{i+1}' is preempted:

i. If neither c'_i nor c'_{i+1} is preempted then the algorithm loses, since there are now enough a calls in S to fill the entire bandwidth for a very large duration ($\mathcal{B}^2$ time units).

ii. If the preempted call is c'_i then we're back in case (2a).

iii. If the preempted call is c'_{i+1}, then the algorithm is less than $\frac{1}{8}$-competitive: The algorithm now serves only c'_i and one a call, with total throughput around $d_{c'_i} \cdot \frac{\mathcal{B}}{2}$. The subsequence S' described in Step (1) above, together with c'_{i+1} and c''_{i+1} has value $e_{c'_{i+1}} \cdot \mathcal{B}$. It follows from the definition of sequence C that $d_{c'_i} < \frac{1}{4} e_{c'_{i+1}}$. Thus, the competitiveness is at most $\frac{\frac{1}{4} e_{c'_{i+1}} \cdot \frac{\mathcal{B}}{2}}{e_{c_{i+1}} \cdot \mathcal{B}} = \frac{1}{8}$.

(c) Reject both c'_{i+1} and c''_{i+1}. This case is similar to case (2(b)iii).

(d) Serve both c'_{i+1} and c''_{i+1}, and preempt both c'_i and the a call. In this case we request many a calls until c''_{i+1} is preempted. Now we are back in case (2a).

C An impossibility result for the parallel links model

In this section we consider a special case of the single link model, called the parallel links model. Here all calls require bandwidth exactly $\mathcal{B}/k$ for some integer k. 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. The throughput accrued when completing a call is its duration.

We remark that we have 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 [GGK+93] [BKM+91a]. We extended them to all even k with a competitive ratio of $\frac{1}{2}$ and to all odd k with competitive ratio $\frac{1}{2} - \frac{1}{4k}$. (We omit the description of our algorithms from this extended abstract.)

We first describe a simple impossibility result for $k = 2$ (the case $k = 1$ was dealt with in [GGK+93], as outlined in Appendix B). At time 0 the adversary (constructing the sequence of requests) requests two calls of length x. If the algorithm serves only one call then we are done. Otherwise, at time $x - 1$ the adversary requests two new requests of length x. At every time unit the adversary also requests many service calls of length 1. It is not hard to verify that the throughput for the adversary will be $2x - 2$ and for the algorithm no more than x. For large x, the algorithm is at most $(\frac{1}{2} + o(1))$-competitive. For the cases $k = 3$ and $k = 4$ we have an impossibility result of 1/2 and 11/20 respectively. (The proofs are omitted.)

We sketch 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 $e^r = \frac{r}{1-r}$, hence, $0.65 \leq r \leq 0.66$.

We have seen that the inductive claim holds for $k = 2$. For $k > 2$, we first present an impossibility result for the throughput of the algorithm with respect to the cover of the input sequence, (as defined in Section 2). Next we describe an off-line strategy that utilizes all the available bandwidth, thus providing the impossibility result for the competitiveness. For this description assume that the duration of a time unit is $\epsilon \ll 1$.

Assume an algorithm with competitiveness r. The adversary requests k calls of length 1 at time 0. The algorithm serves w, where $\frac{w}{k} \geq r$. We apply the adversary strategy for $k - w$ links on a smaller scale, until either we reach time $1 - \epsilon$ or the algorithm preempts one of the initial requests. In the latter case we apply the strategy for $k - w + 1$ links and so on. In general, in the

ith stage, $0 \leq i \leq w$, we apply the adversary strategy for $k - w + i$ links until either it reaches time $1 - \epsilon$ or the algorithm preempts one of the initial requests. In the former case the adversary requests k calls of length 1. In the latter case the adversary goes to stage $i + 1$.

We assume that the algorithm preempts an initial request whenever it has satisfied requests with total duration 1. In other words, the throughput covered by the algorithm at any time is between w and $w+1$. It can be seen that this is the best strategy for the algorithm with respect to this particular sequence. Let z be the number of initial requests that remain when the algorithm reaches time $1 - \epsilon$. It follows that at time $1 - \epsilon$ the algorithm can satisfy $k - z$ new requests. Therefore the throughput covered by the algorithm is $w + k - z$. Thus, the utilization is roughly $\frac{w+k-z}{2k}$ Consequently, $\frac{w}{k} \leq \frac{w+k-z}{2k}$ and

$$w \leq k - z \ .$$

Let α_i be the time passed between stage i and stage $i + 1$ (or the end of the sequence for the last stage). We have $\alpha_0(k - w)r = 1$ since the algorithm needs to cover an throughput equal to 1 and could cover only a fraction r of them by the induction hypothesis. In general, $\alpha_i(k-w+i)r = 1$, or $\alpha_i = \frac{1}{(k-w+i)\cdot r}$. Since $\sum_{i=0}^{w-z+1} \alpha_i = 1$ it follows that $1 = \frac{1}{r}\sum_{i=0}^{w-z+1} \frac{1}{k-w+i} \approx \ln\left(\frac{k-z}{k-w}\right)$. Therefore, $r = \ln\left(\frac{k-z}{k-w}\right)$ which implies that

$$e^r = \frac{k-z}{k-w} \ .$$

Since $k - z \geq w$ it follows that $e^r \geq \frac{w}{k-w}$. Finally, since $r = \frac{w}{k}$ it follows that

$$e^r = \frac{r}{1-r} \ .$$

We conclude by showing how the off-line algorithm can utilize all links for the entire duration of the input sequence. To that end we add the following requests. Whenever we apply the adversary for $k-w+i$ links we request very small service calls that the algorithm will never take. However, the off-line algorithm can serve all of these requests and therefore can utilize all the links. Furthermore, when the off-line reaches time $1 - \epsilon$ it can satisfy all the k new requests.

We remark that slightly better bounds can be achieved when applying this bound to the more general single link model. That is, the technique described in Section 5 for transforming the bound for $k = 1$ to the single link model with $\delta = \frac{1}{2}$ can be generalized to obtain, for all $k \in \mathbf{N}$, an impossibility result of $\frac{(k-1)r}{k} + o(1)$ in the single link model with $\delta = \frac{1}{k}$. (We omit the details.)