Competitive Analysis of Call Admission Algorithms that Allow Delay

Anja Feldmann[1] Bruce Maggs[1] Jiří Sgall[2] Daniel D. Sleator[1] Andrew Tomkins[1]

email contact: anja@cs.cmu.edu

March 21, 1995

Abstract

This paper presents an analysis of several on-line algorithms, called call admission algorithms, for processing requests for connections in distributed networks. Each request comes with a source, a destination, and a bandwidth requirement. The call admission algorithm decides whether to accept the request, and if so, when to schedule it and which path the connection should use through the network. The duration of the request is unknown to the algorithm when the request is made. We analyze the performance of the algorithms on simple networks such as linear arrays, trees, networks with small separators, and meshes. We use three measures to quantify their performance: network utilization (makespan), call-admission ratio, and data-admission ratio. Our results include proofs that greedy algorithms are $\Theta(\log n)$-competitive with respect to makespan on n-node trees and $O(\log^2 n)$-competitive on n^2-node meshes for requests with arbitrary durations and bandwidth requirements, a proof that on an n-node tree no algorithm can be better than $\Omega(\log\log n/\log\log\log n)$-competitive with respect to makespan, and a proof that no algorithm can be better than $\Omega(\log n)$-competitive with respect to call-admission and data-admission ratio on an n-node linear array, if each request can be delayed for at most some constant times its (known) duration.

[1] School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A.
This research is sponsored in part by NSF National Young Investigator Award CCR-9457766, NSF Grant IRI-9314969, and Advanced Research Projects Agency (ARPA) Contract F33615-93-1-1330.
[2]Institute of Computer Science, Hebrew University, Jerusalem 91904; on leave from Mathematical Institute, AV ČR, Žitná 25, 115 67 Praha 1, Czech Republic.

1 Introduction

Recent advances in network technology suggest that, in the future, various classes of traffic — voice, video, and data — will be transmitted on the same high-speed digital network. A user of such a network will initiate a call by submitting a request to a service provider for a connection between two nodes of the network. For example, a user might submit a request to hold a video conference between Pittsburgh and San Francisco. In this request the user will specify the quality of service required, such as minimum bandwidth, but might not be able to specify the duration of the connection. The user hopes to obtain the desired connection within a reasonable amount of time.

The service provider operates a network in which nodes are connected with links of varying bandwidth capacities. For the service provider to guarantee service, shared resources, such as bandwidth on links and buffer capacity at nodes, must be reserved along some path between the terminal nodes of each connection. Since several calls might require the same network resources, the service provider uses a call admission algorithm to decide which requests to grant, when to schedule them, and which paths to use. Call admission is one of the congestion control mechanisms that guarantees the necessary resources to already established calls.

Unlike traditional voice circuits in which the bandwidth requirement is small, some applications, like video, can have high bandwidth requirements. Given such bandwidth requirements, one can expect to see only a small number of connections sharing the same physical link, and hence the traffic on such a link cannot be modeled accurately by statistical means. Traditional methods of analyzing congestion control schemes rely on statistical models to forecast future requests and therefore have restricted use if applied to high-speed networks. We use competitive analysis, i.e., we compare the solution that an algorithm produces on any sequence of requests to the best solution for that sequence of requests [ST85]. This method does not rely on forecasting future events and as such is a realistic alternative to statistical methods.

1.1 The model

The communication network is modeled as an undirected graph with capacities on its edges. *Requests* (we use the terms call and request interchangeably) arrive at various times. Each is specified by a pair of terminal nodes, and the bandwidth required. The job of the call admission algorithm is to decide if *and when* the call should be placed, and which path through the network should be used for the call. Note that we allow the call admission algorithm to delay the start of a call, rather than simply making a binary decision to accept or reject it. Obviously the total bandwidth of the calls on a link in the network cannot exceed its capacity. Once a call is accepted, it remains in place until completion — preemption is not allowed.

The performance of a call-admission algorithm is evaluated by the quality of the schedules that it produces. In this paper we consider the following measures of schedule quality. The *makespan* of a schedule is the total time required to complete all the calls. For call admission algorithms that reject some requests, we consider the *call-admission ratio*, which is the ratio of the number of requests scheduled to the total number of requests, and the *data-admission ratio*, which is the ratio of the amount of data (bandwidth times duration times distance between the terminal nodes) in the requests that are scheduled to the total amount of data in all of the requests. (For a discussion of the *maximum response time*, which is the maximum time between when a request arrives and when it is scheduled to begin, see Feldmann et al. [FMS+95].)

All of these performance measures quantify important aspects of the behavior of the algorithms. The goal of the service provider is to achieve the highest network utilization possible. If all requests

are satisfied, then the schedule with the shortest makespan has the highest network utilization. If, however, some requests are not satisfied, then either the call-admission or data-admission ratio captures the network utilization. The user, on the other hand, is probably most concerned with the maximum response time.

This paper analyzes several call admission algorithms that are on-line. An algorithm is called *on-line* if it is given its input one piece at a time, and must process each piece before seeing the rest of the input. (An algorithm is called *off-line* if it is given all of its input at once.) The algorithms in this paper are on-line either because the requests are not all given at once, or because the durations of the requests are not all given at once. We characterize the performance of an on-line call admission algorithm in terms of its competitive ratio with respect to some measure. The *competitive ratio* of an on-line algorithm is the maximum ratio, over all inputs, of the performance of a schedule computed by the algorithm to the performance of an optimum schedule. (Occasionally, e.g., for call-admission and data-admission ratio, the competitive ratio is defined as the maximum of the *reciprocal* of this ratio.) An algorithm is *k-competitive* if it has a competitive ratio of at most k.

One way in which our work differs from previous analyses is that we allow our call admission algorithms to delay calls. In practice, some delay is almost always acceptable, and in fact delay is routinely encountered when making phone calls, setting up telnet connections, or sending email messages. Also, some sources of delay, such as the latency, or transit time, between the terminals of a connection are outside of the control of a call admission algorithm. Thus, since some delay is likely to be inevitable, it makes sense to allow a call admission algorithm a little leeway. In addition, it is likely that rejected calls will be immediately resubmitted, especially if the calls are initiated by computers. Given that even a rejected call uses some network resources, a network may be swamped by resubmitted, previously rejected, requests. Therefore it seems reasonable to consider a model in which requests may be delayed by the network, which upon establishing the connection notifies both the caller and callee of the request.

In order to investigate possible advantages of delaying calls, we study two variations of the basic model: the first places no limit on the delay allowed before a call is scheduled, while the second limits the maximum allowed delay for each request to some multiple of its own duration. Except for the case when the delay is bounded, in our model the duration of a request is unknown when it is initiated, and can only be determined by scheduling the request and observing how long it takes to complete. This model differs from previously considered models for the on-line call admission problem, which were typically of the following form: the existence of a request is unknown until a certain arrival time, at which point all parameters of the request are completely specified.

1.2 Our results

In Section 2 we study whether the competitive ratio with respect to the call-admission ratio or data-admission ratio improves if a request can be delayed by an amount of time that is proportional to its duration. We allow a request of known duration d to be scheduled within time cd after its arrival, for some constant $c > 0$. We show that for any constant $c \geq 0$, the competitive ratio for both the call-admission ratio and the data-admission ratio of any on-line algorithm (deterministic or randomized) is at least $\Omega(\log T)$, where T is the ratio of the longest duration to the shortest duration. The lower bound on the call-admission ratio holds for a network consisting of a single edge. For the data-admission ratio, the network is an n-node linear array, where $n = \Theta(T)$. This lower bound generalizes the $\Omega(\log n \log T))$ lower bound on the call-admission ratio of [ABFR94, LT94], which consider the case in which no delay is allowed (i.e., $c = 0$). In fact, our lower bound holds even if the on-line schedule is compared to an off-line schedule that is not allowed to delay requests.

In view of this lower bound, in the rest of the paper we allow a request to be delayed for an arbitrary period of time. We consider the performance measure of makespan which is inversely proportional to the network utilization. When the network is overloaded, maximizing the network utilization by minimizing the makespan has the benefit of keeping periods of overload as short as possible and therefore helps the network to return to normal operation as quickly as possible.

We note that algorithms that are competitive with respect to makespan when all requests appear at once, called *batch-style algorithms*, are also competitive when requests arrive over time, called *fully on-line* algorithms, with at most a factor of 2 change in the competitive ratio. This result follows pretty much directly from a theorem of Shmoys, Wein, and Williamson [SWW91]. In Section 5, we strengthen this result by showing that with some mild restrictions on the type of algorithms allowed, a k-competitive batch-style algorithm can be converted into a $(k+1)$-competitive fully on-line algorithm. An algorithm that satisfies these restrictions is called *mergeable*.

The most basic call admission algorithm simply schedules each request as soon as its requirements can be satisfied. Algorithm GREEDY (described and analyzed in Section 3) is the batch-style version of this. (Note that although GREEDY is not a fully on-line algorithm, it is still an on-line algorithm because the durations of the requests in a batch are not known.) In addition, GREEDY is mergeable, so any bound on its competitive ratio with respect to makespan immediately translates to the fully on-line variant of it, which is the natural greedy algorithm for call admission. We analyze the competitive ratio of GREEDY with respect to makespan on linear arrays, trees, networks with small separators, and meshes.

GREEDY is $O(\log n)$-competitive on any n-node tree with unit edge capacity, provided that there is an upper bound of $1-1/c$ (or a lower bound of $1/c$) on the bandwidth of any call, for some fixed $c > 1$. We give a simple modification of GREEDY that is $O(\log n)$-competitive on any n-node tree for arbitrary bandwidth requirements. This modification is unavoidable, since, as we show in Appendix B, GREEDY may perform arbitrarily badly if arbitrary bandwidth requirements are allowed. In the same appendix we show that our analysis of GREEDY is tight since it cannot be better than $\Omega(\log n)$-competitive on n-node complete binary trees even if all link capacities and bandwidth requirements are 1.

Addressing the routing problem, we show how to generalize GREEDY to networks with small separators by providing a way to chose paths for the requests that are reasonable with respect to the separators. In Appendix 3.2 we give the generalized upper bound on the competitive ratio. In Section 3.3 we address the problem of routing in mesh networks and give a variant of GREEDY which is $O((\log n)^2)$-competitive for n^2-node mesh networks.

Considering several restricted cases provides useful insight into the combinatorial structure of the call admission problem. Using the relationship of call admission to on-line coloring and to dynamic storage allocation we show (Section 3.4) that the competitive ratio of GREEDY on a linear array network is between 4.4 and 25.8 if each edge has unit capacity, and each request requires unit bandwidth.

In Section 4 we prove an $\Omega(\log\log n/\log\log\log n)$ lower bound for any deterministic on-line call admission algorithm on an n-node complete binary tree network with respect to makespan, even if all link capacities and bandwidth requirements are 1. This non-constant lower bound confirms that there is a fundamental difference in call admission on a linear array network and a tree network.

All of our competitive ratios are independent of the maximum duration and the ratio of the largest to the smallest bandwidth requirement. All our algorithms are deterministic and do not reject any calls.

1.3 Related work

Awerbuch et al. [ABFR94] present on-line call-admission algorithms for trees. In their model, requests include known bandwidth, duration and benefit, and must be scheduled immediately or rejected. They

present a general technique called "classify and randomly select", in which the algorithm randomly selects a bandwidth f, duration t, and benefit m, all of which are powers of 2. The algorithm then restricts itself to calls that have bandwidth between f and $2f$, duration between t and $2t$, and benefit between m and $2m$, rejecting all other calls. Using tree separators, they apply a similar classify and select technique to schedule the restricted set of calls with an $O(\log n)$ competitive ratio, with respect to total benefit, where n is the number of nodes in the tree. The total competitive ratio is therefore $O(\log n \log M \log T \log F)$, where M is the ratio of the largest benefit to the smallest benefit, T is the ratio of longest duration to the shortest, and F is the ratio of the largest bandwidth requirement to the smallest. They also give a lower bound of $\Omega(\log n \log M \log T)$. Yet, all algorithms designed using the "classify and randomly select" paradigm seem unsatisfactory to real network users, even though they are theoretically competitive.

Another possible problem with such a model is that the sequence of requests presented to a call admission algorithm is highly dependent on the way that the algorithm responds to requests. For example, whenever a call admission algorithm immediately rejects a request, it is likely to see that request immediately reissued by a user. Although we may be able to use competitive analysis to prove that such an algorithm performs close to optimally on such a sequence, and indeed on any sequence presented to it, if the algorithm itself encourages worst-case sequences (with respect to, say, the achievable call-admission ratio), then the use of competitive analysis is counterproductive. Therefore another reason to consider algorithms that may delay calls is that in practice, call admission algorithms that reject fewer (if any) requests seem less likely to encourage worst-case sequences of requests and are more likely to be acceptable to the user of the network.

Lipton and Tomkins [LT94] study the problem of scheduling a single resource (e.g., a single network edge) for requests with known duration. As in [ABFR94], requests must be scheduled immediately or rejected. The goal is to keep the resource busy for as much time as possible. They show a $\log^{1+\epsilon}(T)$-competitive algorithm, in which T is the ratio between longest and shortest requests, and need not be known in advance. They also show an $\Omega(\log T)$ lower bound on the competitive ratio of any algorithm.

Awerbuch, Gawlick, Leighton, and Rabani [AGLR94] focus (mainly) on the problem of designing a call admission algorithm for the case in which network links have unit capacity, and requests have unit bandwidth requirements and infinite durations. They present, among other things, an algorithm that is $\Theta(\log d)$-competitive, with respect to the call-admission ratio, for tree networks of diameter d, and an algorithm that is $O(\log n \log\log n)$-competitive for n^2-node meshes.

Garay et al. [GGK+93] give algorithms for linear array networks, under various benefit models. They assume that durations of calls are known in advance, and in most of their models they assume that all bandwidths are 1. Awerbuch et al. [AAPW94] study calls of unknown durations, but do so in the context of relative load on the network, in a model that allows calls to be rerouted a limited number of times. Awerbuch et al. [AAP93] give $O(\log n)$-competitive algorithms for n-node networks with arbitrary topologies, but do not allow calls to require more than a $1/\log n$ fraction of the available bandwidth of a single link.

2 Limiting the maximum delay

In this section we allow a request to be delayed for some bounded time, but not indefinitely. We suppose that a request of duration d has to be scheduled within time cd after its arrival for some constant $c \geq 0$; if it is not scheduled, it is rejected and we do not succeed in sending the data associated with it. The duration of a request is known as soon as it arrives.

In this case it is not interesting to consider makespan as a performance measure, since it is obviously

bounded in terms of the arrival times of jobs, their durations, and their maximal allowed delay. On the other hand, not all calls are scheduled, and thus it is important to consider both the call-admission ratio (the number of scheduled requests divided by the total number of requests) and the data-admission ratio (the total amount of data in the scheduled requests divided by the total amount of data in all requests).

Theorem 2.1 *Suppose that R is a randomized call admission algorithm for a linear array network such that each request is either scheduled between t and $t + cd$, where t is its arrival time, d is its duration, and $c \geq 0$ is a constant, or not scheduled at all. Suppose also that all link capacities and bandwidth requirements are 1. Let T denote the ratio between the longest and shortest durations. Then*

- *the competitive ratio of R with respect to call-admission ratio is at least $\Omega(\log T)$ for a 2-node linear array network, and*
- *the competitive ratio of R with respect to data-admission ratio is at least $\Omega(\log T)$ for an $n + 1 = \Theta(T)$-node linear array network,*

even if the off-line schedule has to satisfy every request immediately or reject it.

Proof. Without loss of generality, we can assume that c is an even integer (we can always round it up). We denote the nodes of the linear array by $v_0, \ldots, v_n$. The notation (t, u, v, b, d) $(b = 1)$ denotes a request that arrives at time t for a connection between nodes u and v with bandwidth requirement b and duration d.

We first construct a large set of requests and point out its important properties; later we choose a random subset of the large set of requests (according to a specific distribution) that we use as an actual input for the call admission algorithm.

Let $\bar{c} = 3c/2 + 1$, $\hat{c} = 2\bar{c} + 1 = 3c + 3$, and $l = \lfloor \log_{\hat{c}} T \rfloor$.

The requests are indexed by sequences $\alpha = (\alpha_1, \ldots, \alpha_{|\alpha|})$ of odd integers from $[1, \hat{c} - 1]$ of length $|\alpha| \leq l$. The request indexed by α arrives at time $\mathrm{Arr}(\alpha)$ which is defined as $(\alpha_1 \hat{c}^{l-1} + \alpha_2 \hat{c}^{l-2} + \ldots + \alpha_{|\alpha|} \hat{c}^{l-|\alpha|})(c + 1)$, i.e., as if we interpret the sequence as a number with l digits written in base $\hat{c}$, which is then multiplied by $c + 1$. Let the request indexed by α be $r_\alpha = (\mathrm{Arr}(\alpha), v_0, v_{\hat{c}|\alpha|}, 1, \hat{c}^{l-|\alpha|})$, see Figure 1. Note that the time interval during which the request r_α may be running starts at $\mathrm{Arr}(\alpha)$ and is $(c + 1)\hat{c}^{l-|\alpha|}$ long.

We have made the duration of each request inversely proportional to the length of the path between its terminals so that the amount of data of each request is the same. As a consequence, the data-admission ratio is equal to the call-admission ratio. Since all requests require full bandwidth on the edge between v_0 and v_1, the time intervals in which the satisfied requests are scheduled have to be pairwise disjoint.

The requests are naturally arranged in a $\bar{c}$-ary tree, with the smallest element $r_\emptyset$ at the root, and $r_{\alpha'}$ being the descendant of r_α if α is a prefix of α'. The following are the only two properties of intervals we need for the rest of the proof; both are easy to check.

(1) Whenever the on-line algorithm schedules (within the appropriate interval) a request r that is not on the last level of the tree, there is a child r' of r in the tree such that r' has not arrived yet and no request in the subtree of r' can be scheduled (without a collision with r).

(2) If the off-line algorithm schedules (with no delay) a request r_α, it does not collide with any request in the tree below r except for the requests in the tree below $r_{\alpha 1}$.

Now we define the random subset $\mathbf{R}$ of requests that we use as an input; in fact it is a subtree of the tree defined above. It is defined recursively as follows:

- $r_\emptyset \in \mathbf{R}_0$;
- for any $r_\alpha \in \mathbf{R}_i$, $|\alpha| < l$ and β an odd number from $[1, \hat{c} - 1]$, we put $r_{\alpha\beta}$ in $\mathbf{R}_{i+1}$ with probability $1/\bar{c}$.

All random choices are made independently. Now $\mathbf{R} = \cup_{\mathbf{i=0}}^{\mathbf{l}} \mathbf{R_i}$. By $\#(\mathbf{R}_i)$ we denote the number of children of nodes in R_i that are placed in R_{i+1}. From the definition of $\mathbf{R}_0$ it follows that the expected number of children of the root, $\mathrm{E}[\#(\mathbf{R}_0)]$, is 1 for $i = 0$. Recursively, the expected number of children of nodes on level i of the tree, $\mathrm{E}[\#(\mathbf{R}_i)]$, is 1, for all i.

First we compute the expected number of requests in $\cup_{j=i}^{l} \mathbf{R_j}$ that the optimal schedule can satisfy. We denote this number $\mathrm{OPT}(i)$. For $i = l$, $\mathrm{OPT}(i) = 1$, since the optimal schedule will schedule the single request. Now we compute $\mathrm{OPT}(i-1)$ as a function of $\mathrm{OPT}(i)$. The optimal strategy optimally satisfies the requests of the subtrees rooted at all the existing children, and then also schedules the root request if possible, which is if and only if the first child is not present (recall condition (2)). Thus the expected number of satisfied requests is $\mathrm{E}[\#(\mathbf{R}_{i-1})]\mathrm{OPT}(i) + \mathrm{E}[\#(\mathbf{R}_{i-2})](1 - 1/\bar{c}) = \mathrm{OPT}(i+1) + (1 - 1/\bar{c})$, and hence $\mathrm{OPT}(i) = \Theta(l - i)$. Hence, the expected number of requests that an optimal schedule can satisfy is $\mathrm{OPT}(0) = \Theta(l)$.

Now we bound the maximal (expected) number $F(i)$ of requests in $\cup_{j=i}^{l} \mathbf{R_j}$ satisfied by any on-line algorithm. We prove by induction on i that $F(i) \leq \bar{c}$. For $i = l$, $F(l) = 1 \leq \bar{c}$. To bound $F(i-1)$ we distinguish two cases. First, suppose that on some partially known input the algorithm decides to schedule a node R in $\mathbf{R_{i-1}}$. Then (according to condition (1)) the whole subtree of the first child is blocked; since it is before the arrival time of the requests in this subtree, the on-line algorithm has no possible way of knowing if they are in the random set $\mathbf{R}$. Thus, averaging over the instances consistent with this partial input, we find that the expected number of subtrees of R that are blocked is at least $1/\bar{c}$, and the expected number of requests that are scheduled is $1 + (\mathrm{E}[\#(R)] - 1/\bar{c})F(i) \leq \bar{c}\mathrm{E}[\#(R)]$. Second, if for some node R in $\mathbf{R}_{i-1}$ the on-line algorithm does not schedule the corresponding request, then the expected number of satisfied requests is bounded by $\mathrm{E}[\#(R)]F(i) \leq \bar{c}\mathrm{E}[\#(R)]$. Averaging over all instances we get that the average number of satisfied requests $F(i-1)$ is bounded by $\bar{c}\mathrm{E}[\#(\mathbf{R}_i)] = \bar{c}$. Note that the fact that the algorithm can be randomized does not change the analysis above.

Suppose that we have a randomized σ-competitive algorithm. By averaging over all instances from our distribution and the definition of competitiveness, it has to satisfy at least $\mathrm{OPT}(0)/\sigma$ requests. Since by the above analysis it can satisfy only a constant number, $\bar{c}$, of the requests, the competitive ratio is at least $\Omega(\mathrm{OPT}(0)) = \Theta(l) = \Theta(\log T)$. □

3 The GREEDY algorithm

For the rest of this paper there is no limit on the delay allowed before a call is scheduled and the duration of a request is unknown to the on-line algorithms until the connection is terminated. We also assume that all requests are available at time $t = 0$, i.e., our algorithms are batch-style. All our algorithms can be converted to fully on-line algorithms as shown in Section 5.

The algorithm GREEDY schedules each request r at the first time t for which enough bandwidth is available for r. In this section we analyze its performance with respect to makespan on simple networks such as trees, networks with small separators, and meshes.

3.1 GREEDY for tree networks

A formal description of the GREEDY algorithm for tree networks is presented below. The tuple (u, v, b, d) denotes a request for a connection between nodes u and v requiring bandwidth b and duration

d (d is not given in the input).

Algorithm GREEDY for trees

Given a batch of requests $r_0, r_1, \ldots, r_m$ do:

repeat until all requests have been scheduled

for $i := 0$ **to** m **do**

let $r = r_i = (u, v, b, d)$ be the i-th request in the batch;

if r has not yet been scheduled **and** bandwidth $b' \geq b$ is available along a path (u, v) from u to v

schedule r along path (u, v) at time t;

wait until some call finishes;

GREEDY is $O(\log n)$ competitive with respect to makespan for tree networks if the bandwidth requirements are either not too large or not too small. This condition is unavoidable because GREEDY may perform arbitrarily badly otherwise, see Appendix B. In practice this is hardly a restriction at all, since the price of a connection requiring almost the whole bandwidth will be huge and only a few applications need such a large percentage of the link bandwidth (cf. [AW92]).

To remove the restriction on the bandwidth, we define $\overline{\text{GREEDY}}$ to be the following algorithm. First, use GREEDY to schedule all the requests with bandwidth at most $1/2$. After all these calls are finished, use GREEDY to schedule all the requests with bandwidth more than $1/2$.

The following theorem follows from more the general Theorem 3.5 which is proved in Appendix A.

Theorem 3.1 *Given an n-node tree network G with maximum degree 3 (e.g. any binary tree) and unit edge capacities then (i) for any $c > 1$, GREEDY is $2(c+1)(\log_2 n)$-competitive with respect to makespan if the bandwidth requirement of each request is at most $1 - 1/c$ or the bandwidth requirement of each request is at least $1/c$; (ii) $\overline{GREEDY}$ is $(12 \log_2 n)$-competitive with respect to makespan.*

3.2 GREEDY for networks with small separators

In order for GREEDY to achieve similar performance bounds on any network with small edge separator trees as on tree networks we need to restrict the possible set of paths GREEDY is allowed to use. This restriction is necessary because of the following example. Suppose that we have a 1-dimensional ring network of n nodes with edge capacities 1, and a list of requests $L = r_0 = (0, 1, 2/3, 1); r_1 = (1, 2, 2/3, 1), \ldots, r_{n-1}(n-2, n-1, 2/3, 1)$. Imagine that GREEDY would route r_0 the long way. Then no other request can be scheduled during time step 0. If GREEDY does the same with all other requests the length of the schedule will be $n - 1$ time steps long while the optimal schedule has length 1.

We start the discussion on the restriction of the set of paths by defining edge separator trees. Recall that a graph (V', E') is a vertex-induced subgraph of (V, E) if $V' \subset V$ and $E' = E \cap (V' \times V')$.

Definition 3.2 *A set of edges S* separates *a graph (V, E) into graphs (V_1, E_1), ..., (V_a, E_a) if every (V_b, E_b), $1 \leq b \leq a$, is a disjoint connected vertex-induced subgraph of (V, E) and E is a disjoint union of S, E_1, ..., E_a.*

An edge separator tree *ST for a graph G is a rooted tree in which each vertex w is labeled by a pair (G_w, S_w), so that for every vertex w and its children $w_1, \ldots, w_a$ in ST, S_w separates G_w into $G_{w_1}, \ldots, G_{w_a}$; moreover, if w is the root of ST then $G_w = G$.*

Let the size *of the separator tree ST, denoted size(ST), be the maximum sum of sizes (number of edges) of the separators S_w along a path from the root to a leaf in a separator tree ST.*

Note that it follows from the definition that every G_w in ST is a connected vertex-induced subgraph of G, and for each leaf of ST w, S_w contains all edges of G_w.

The algorithm GREEDY for networks with edge separator tree ST is allowed to route requests only along reasonable paths with respect to ST, which are defined below.

Definition 3.3 *For a request $r = (u, v, b, d)$, let $z_r \in ST$ be the (unique) vertex such that both u and v are in G_{z_r}, but for each child w of z_r, G_w contains at most one of u and v. A path from u to v is* reasonable *with respect to ST if it is contained in G_{z_r}.*

It is easy to check that for any separator tree and any request there exists at least one reasonable path connecting them (the fact that graphs G_w are connected is needed here). Note that the length of a reasonable path and the size, $size(ST)$, of the separator tree ST are not necessarily related and that the unique path connecting u and v in a tree is always a reasonable path.

To get a formal description of GREEDY on a network with a small edge separator tree ST, replace "along a path" with "along a reasonable path" in the description of GREEDY for tree networks.

The GREEDY algorithm cannot delay a request without a good excuse, meaning there has to be a set of requests scheduled instead of the delayed one such that all of these requests conflict with the delayed request and the sum of their bandwidth requirements is sufficient to prohibit the delayed request from being scheduled. This GREEDY property is stated more formally in Lemma 3.4.

Lemma 3.4 (GREEDY property) *If request $r = (u, v, b, d)$ is scheduled at time t then for every reasonable path p for r and for every time $t' < t$ there exists an edge e with capacity $c(e)$ on the path p such that the total bandwidth of all the requests using e at time t' exceeds $c(e) - b$.*

The analysis of GREEDY is given in Appendix A, and proves the following theorem.

Theorem 3.5 *Given a network G of n nodes with unit edge capacities and an edge separator tree ST for G, then (i) for any $c > 1$, GREEDY is $(c+1)size(ST)$-competitive with respect to makespan if the bandwidth requirement of each request is at most $1 - 1/c$ or the bandwidth requirement of each request is at least $1/c$; (ii) $\overline{GREEDY}$ is $(6 \cdot size(ST))$-competitive with respect to makespan.*

The results for GREEDY on tree networks are a special instance of Theorem 3.5. For every n-node tree there is a node v whose removal separates the tree into components such that each contains at most $n/2$ nodes of the tree. Since in our case the tree has maximum degree 3, the removal of the two edges leading to the two larger components separates the tree into at most three components, each containing at most $n/2$ nodes. Using this fact recursively, we can construct a separator tree ST with depth $\log_2 n$ and a size of each separator 2; hence $size(ST) = 2\log_2 n$. Theorem 3.1 now follows by observing that the unique path connecting u and v in the tree is always reasonable.

We can use Theorem 3.5 on any planar network with bounded node degree, which includes two-dimensional mesh networks. Lipton and Tarjan [LT79] showed that any planar graph has a vertex separator of size $O(\sqrt{n})$ such that the size of each component is a fixed fraction of the size of the graph. If the degree is bounded, it follows that we have an edge separator with the same parameters as well. If we apply this recursively, we get a separator tree ST with $size(ST) = O(\sqrt{n})$.

Theorem 3.6 *Given any planar network with bounded degree nodes and unit edge capacities then (i) for any $c > 1$, GREEDY is $O(c\sqrt{n})$-competitive with respect to makespan if the bandwidth requirement of each request is at most $1 - 1/c$ or the bandwidth requirement of each request is at least $1/c$; (ii) $\overline{GREEDY}$ is $O(\sqrt{n})$-competitive with respect to makespan.*

3.3 GREEDY for mesh networks

For the important special case of meshes we can obtain a much stronger result by combining the GREEDY algorithm with the idea of routing the requests along an embedded tree of meshes from [AGLR94].

Theorem 3.7 *Given a two-dimensional $n \times n$-mesh with unit edge capacities a variant of GREEDY is $O((\log n)^2)$-competitive with respect to makespan.*

Proof. We sketch the proof for an $O(n \log n) \times O(n \log n)$-mesh, where n is a power of 2 (this does not change the asymptotics). The $n \times n$ tree of meshes has $2 \log n$ levels. The root is an $n \times n$ mesh. It is connected to two children, which are $n \times \frac{n}{2}$-meshes, by n wires (going from the nodes on its right edge to the nodes on the left edge of one of the smaller meshes and symmetrically on the other side). These meshes are connected each to two children, $\frac{n}{2} \times \frac{n}{2}$ meshes, by $n/2$ wires from top and bottom edges, and these children are roots of $\frac{n}{2} \times \frac{n}{2}$ trees of meshes constructed recursively. For sufficiently large k ($k = O(\log n)$), the $n \times n$ tree of meshes can be embedded in a $kn \times kn$ mesh so that (1) the mesh is partitioned into $k \times k$ submeshes containing one leaf of the tree each, (2) each $a \times b$ mesh from the tree is embedded so that it together with the whole subtree are contained in some $ka \times kb$ submesh, and no other leaves of the tree are embedded in this submesh, and (3) only wires in the even rows and even columns of the mesh are used for the embedding of the tree. We omit the easy technical details (cf. [AGLR94]).

We partition the requests into the following sets. The first set consists of local requests, i.e., those which have both endpoints within one $k \times k$ submesh (from (1)). The remaining, long-distance, requests are divided into 4 sets according to the parity of the row of one of the endpoints and the parity of the column of the other endpoint in the mesh (e.g. odd-row to even-column requests). Each of the five sets is further divided into the set of calls requesting bandwidth at most 1/2 and the set of calls requesting bandwidth more than 1/2. This results in 10 sets, which we schedule one after the other. Next we describe how to schedule each set.

The local calls are routed using GREEDY which is $O(\sqrt{k})$-competitive by Theorem 3.6.

Now we schedule the set of odd-row to odd-column long-distance requests (the other cases are obtained similarly, by shifting the embedding of the tree of meshes by one along one or both coordinates), using the following greedy rule. Whenever a call can be routed from both endpoints along odd rows and odd columns to the leaf of the tree of meshes embedded in both of its $k \times k$ submeshes, and then along (one of the) shortest paths in the tree of meshes, we route it. Consider the call which was the last one to be scheduled, at time t. At any time before t there has to be some $a \times b$ mesh in the tree of meshes which this call has to use and which is used by calls of bandwidth $\Omega(a)$. (If there is no such congested mesh, the call would be scheduled, using the fact that either all the calls have bandwidth at most 1/2 or more than 1/2.) Choose the mesh for which this happens for the longest total time T. Obviously $T \geq \Omega(t/\log n)$. Even in the optimal schedule the calls congesting the chosen $a \times b$-mesh have to leave this mesh or its child. Since there are only $O(ak)$ wires to do this and the total amount of data which has to get through the mesh is $\Omega(aT)$, this takes $\Omega(T/k)$ time. Hence we are $O(k \log n) = O((\log n)^2)$ competitive for this set of requests (using also the fact that the optimal schedule is longer than the duration of the last job), and, since there are only 10 sets, also for the whole schedule. □

3.4 GREEDY for linear array networks

If each request specifies not only its terminal nodes but also the path that its connection will take through the network, and all link capacities, bandwidth requirements, and durations are 1, then the

call admission problem is equivalent to coloring the nodes of a graph that we call the *conflict graph*. In the conflict graph there is a node for each request and an edge between two nodes if the corresponding requests share a link of the network. A coloring of the nodes of the conflict graph corresponds to a schedule of the requests. Each color corresponds to a time step and the total number of colors used corresponds to the makespan of the schedule.

The simple correspondence between call admission algorithms and coloring algorithms is lost once we introduce arbitrary durations for requests. We can nevertheless use the relationship between coloring and call admission to show that GREEDY is constant competitive on a linear array if each edge has unit capacity and each request requires unit bandwidth. The proof shows that the schedule generated by the GREEDY algorithm corresponds to a coloring produced by a particular *on-line coloring algorithm* even with calls of arbitrary duration. An on-line coloring algorithm is one that is given the nodes of a graph one-by-one according to a specific sequence. As each node is revealed, the edges connecting that node to previously seen nodes are also revealed. The on-line coloring algorithm must assign a color to each node immediately after seeing it, and once assigned, the color cannot be changed.

Given a linear array network the conflict graph is an *interval graph*. In an interval graph the nodes can be represented by nonempty intervals on the real axis, and an edge exists between each pair of nodes whose intervals intersect. The simplest on-line coloring algorithm assigns each node u the color α which is the least positive integer not already assigned to a neighbor of u. This algorithm is called *First-Fit*. For interval graphs, Kierstead [Kie88] showed that First-Fit is 40-competitive with respect to the number of different colors used, and later Kierstead and Qin [KQ92] improved this bound to 25.8. Chrobak and Slusarek [CS88] give an example that proves that First-Fit cannot be better than 4.4-competitive.

Theorem 3.8 *Given a linear array network with unit capacity edges, and requests with unit bandwidth requirements and unknown durations, GREEDY is* 25.8*-competitive with respect to makespan and cannot be better than* 4.4*-competitive.*

Proof. See Feldmann et al. [FMS+95]. □

4 Lower bound for call admission on a tree network

In this section we prove that no deterministic on-line call admission algorithm on a tree of $n = 2^{d+1} - 1$ nodes can achieve a competitive ratio better than $\frac{\log d}{\log \log d}$ with respect to makespan.

In Appendix B, we give an example that proves that GREEDY cannot be better than $(1/2)\log n$-competitive with respect to makespan on n-node binary trees. For the lower bound of this section, the adversary will try to restrict the possibilities of the call admission algorithm so that the algorithm must act similarly to GREEDY in the example. The main difference between the two lower bounds is that the adversary can force GREEDY to schedule a specific subset of the requests, but it does not seem possible to force any on-line algorithm to do so. Thus, this lower bound is exponentially weaker.

The adversary proceeds either by setting the durations of a small number of jobs so that the on-line algorithm makes little progress during some period of time, or by finding an unused part of the network. Simultaneously, the adversary finds a subtree of the network and a subset of unscheduled requests that constitute a smaller size problem of the same kind.

Theorem 4.1 *Consider a network that is a complete binary tree T of depth d with $n = 2^{d+1} - 1$ nodes and unit edge capacities. Suppose we have a deterministic call admission algorithm for requests with*

unit bandwidth requirements on such a network. Then the competitive ratio with respect to makespan is at least $\Omega(\frac{\log\log n}{\log\log\log n}) = \Omega(\frac{\log d}{\log\log d})$.

Proof. See Appendix C. □

5 Conversion of batch-style algorithms to on-line algorithms

In this section we show that any algorithm for processing a single batch of requests that is k-competitive with respect to makespan can be converted into an algorithm that can process requests that arrive while others are being serviced that is $2k$-competitive with respect to makespan. We call the former algorithm a *batch-style* algorithm and the latter a *fully on-line* algorithm. This technique allows us to employ any batch-style algorithm as a black box to generate a fully on-line algorithm. We also show that in certain cases a k-competitive batch-style algorithm may have an internal structure that allows us to convert it into a fully on-line algorithm that is $(k+1)$-competitive. Algorithms that allow this more efficient conversion can incorporate new requests immediately into the schedule under construction. We refer to these algorithms as *mergeable* algorithms.

The first result follows (pretty much immediately) from a theorem of Shmoys, Wein, and Williamson [SWW91].

Theorem 5.1 (SWW) *If $\mathcal{A}$ is a batch-style algorithm that is k-competitive with respect to makespan, then $\mathcal{A}'$ is an fully on-line algorithm that is $2k$-competitive with respect to makespan.*

A mergeable algorithm is defined as follows.

Definition 5.2 *Algorithm $\mathcal{A}'$ is a* mergeable variant *of $\mathcal{A}$ if*

1. *$\mathcal{A}'$ performs identically to $\mathcal{A}$ on batch-style request sequences (i.e., sequences in which all release times are 0).*
2. *Upon being presented with a set of new requests at time t_{new}, $\mathcal{A}'$ incorporates them into its schedule so that the set of requests running (or beginning) at t_{new} is identical to the set of requests that $\mathcal{A}$ would schedule to begin at time 0 if presented at time 0 with a batch $\mathcal{I}'$ of requests, where the sequence $\mathcal{I}'$ satisfies the following properties.*
   - *For each request r running at time t_{new} in the schedule produced by $\mathcal{A}'$, $\mathcal{I}'$ contains a request whose duration is $end(r) - t_{new}$, where $end(r)$ denotes the completion time of r.*
   - *$\mathcal{I}'$ also contains all requests that have not been scheduled to begin before time t_{new} in the schedule produced by $\mathcal{A}'$.*
   - *$\mathcal{I}'$ does not contain any requests that have completed by time t_{new} in the schedule produced by $\mathcal{A}'$.*

If $\mathcal{A}'$ exists then $\mathcal{A}$ is a mergeable *algorithm.*

The following theorem shows that mergeable algorithms can be converted to accept arbitrary release times with less overhead than general batch-style algorithms.

Theorem 5.3 *If $\mathcal{A}$ is a batch-style algorithm that is k-competitive with respect to makespan, and $\mathcal{A}'$ is a mergeable variant of $\mathcal{A}$ then $\mathcal{A}'$ is an fully on-line algorithm that is $(k+1)$-competitive with respect to makespan.*

Proof. Omitted □

6 Concluding remarks

In this paper we show that the ability to delay a request by a constant times its duration gives no advantage to a scheduling algorithm with respect to both call-admission ratio and data-admission ratio. It is an open question if there exists an alternative way to limit the delay that actually gives an advantage to the algorithm.

Our non-constant lower bound with respect to makespan for unit duration requests on tree networks shows that there is a significant difference between scheduling on trees and scheduling on linear array networks. The gap between our $\Omega(\log\log n/\log\log\log n)$ lower bound for all deterministic algorithms and the $\Theta(\log n)$ upper bound of GREEDY for n-node trees raises the following questions: are there better algorithms for trees, and can the lower bound be strengthened? Another point of interest is whether the lower bound can be extended to hold against randomized algorithms.

Our result that GREEDY is competitive for networks which can be decomposed using separators poses the question whether there are other network decompositions which lead to competitive algorithms.

References

[AAP93] Baruch Awerbuch, Yossi Azar, and Serge Plotkin. Throughput-competitive on-line routing. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, pages 32–41, Palo Alto, California, November 1993.

[AAPW94] Baruch Awerbuch, Yossi Azar, Serge Plotkin, and Orli Waarts. Competitive routing of virtual circuits with unknown durations. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 321–327, Arlington, Virginia, January 1994.

[ABFR94] Baruch Awerbuch, Yair Bartal, Amos Fiat, and Adi Rósen. Competitive non-preemptive call control. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 312–320, Arlington, Virginia, January 1994.

[AGLR94] Baruch Awerbuch, Rainer Gawlick, Tom Leighton, and Yuval Rabani. On-line admission control and circuit routing for high performance computing and communication. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 412–423, Santa Fe, New Mexico, November 1994.

[AW92] Heinrich Armbrüster and Klaus Wimmer. Broadband multimedia applications using ATM networks: High-performance computing, high-capacity storage, and high-speed communication. *IEEE Journal on selected areas in communications*, 10(9):1382–1396, December 1992.

[CS88] M. Chrobak and Maciej Slusarek. On some packing problems related to dynamic storage allocation. *RAIRO Informatique Thèorique et Applications*, 22(4):487–499, 1988.

[FMS+95] Anja Feldmann, Bruce M. Maggs, Jiří Sgall, Daniel D. Sleator, and Andrew Tomkins. Competitive analysis of call admission algorithms that allow delay. Technical Report CMU-CS-95-102, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, January 1995.

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

[Kie88] H.A. Kierstead. The linearity of first-fit for coloring interval graphs. *SIAM Journal of Discrete Mathematics*, 1(4):526–530, 1988.

[KQ92] H.A. Kierstead and Jun Qin. First-fit and interval graphs, 1992. Unpublished manuscript.

[LT79] Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[LT94] Richard J. Lipton and Andrew Tomkins. Online interval scheduling. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 302–311, Arlington, Virginia, January 1994.

[ST85] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[SWW91] David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 131–140, San Juan, Puerto Rico, October 1991.

A Analysis of GREEDY for networks with small separators

This section provides a proof of Theorem 3.5, which is restated below. First we prove an important property of the reasonable paths.

Lemma A.1 *If reasonable paths for two requests r and r' (with respect to the same separator tree ST) intersect then either z_r is a descendant of $z_{r'}$ in ST, or $z_{r'}$ is a descendant of z_r or $z_r = z_{r'}$.*

Proof. From the definition of a separator tree it follows that otherwise the graphs G_{z_r} and $G_{z_{r'}}$ are disjoint, and so are the reasonable paths, as they are contained in these graphs; this contradicts the assumption. □

Theorem A.2 *Given a network G of n nodes with unit edge capacities and an edge separator tree ST for G, then (i) for any $c > 1$, GREEDY is $(c+1)size(ST)$-competitive with respect to makespan if the bandwidth requirement of each request is at most $1 - 1/c$ or the bandwidth requirement of each request is at least $1/c$; (ii) $\overline{GREEDY}$ is $(6 \cdot size(ST))$-competitive with respect to makespan.*

Proof. (ii) is a trivial consequence of (i). Before proving (i), we introduce some notation. Let t_{OPT} be the makespan of an optimal schedule and let t_{GREEDY} be the makespan of the schedule generated by GREEDY. Let d_{max} denote the maximum duration of any request.

Suppose $w_0, w_1, \ldots, w_k$ is a path in the separator tree ST starting at the root. For any $i \leq k$, let R_i be the set of all requests r such that $z_r = w_i$ (i.e., the endpoints of r are both in G_{w_i} but not in $G_{w'}$ for any child w' of w_i). Let $R = \bigcup_{i=0}^{k} R_i$. For this set of requests R, let $den(R)$ denote their density, which is bandwidth times duration summed over all of the requests in R, and let $den_t(R)$ denote the density at time t, which is the sum of the bandwidth of all requests from R running at time t in the schedule generated by GREEDY. Obviously $den(R) = \sum_{t=1}^{t_{GREEDY}} den_t(R)$.

First we construct a path $w_0, w_1, \ldots, w_k$, such that for all but $(k+1) * d_{max}$ time steps t, $den_t(R) \geq 1/c$. We construct the path from the root w_0 down. In order to determine which child of vertex w_i on the path will be w_{i+1}, we check the children w'_b of w_i in ST to see if there is some request with both endpoints in the corresponding subgraph $G_{w'_b}$. If no subgraph has such a request, the path terminates at w_i. Otherwise, if only one child's subgraph has such a request, then that child becomes w_{i+1}. If several children's subgraphs have such requests, then we choose the child whose subgraph contains a request with the latest finishing time in the schedule (breaking ties arbitrarily).

We define r_i to be the request scheduled by GREEDY on a path contained in G_{w_i} that finishes latest of all such requests (ties broken arbitrarily). Note that r_i's are not necessarily distinct. Let t_i be the time when r_i finishes in the schedule and b_i the bandwidth requirement of r_i. Since $G_{w_{i+1}}$ is a subgraph of G_{w_i} and $G_{w_0} = G$, we have $t_{i+1} \leq t_i \leq t_0 = t_{GREEDY}$.

First consider a time step t such that some r_i is running at time t. For such a time step $den_t(R)$ might be small (i.e., less than $1/c$). However, since there are at most $k+1$ requests, r_0 through r_k, and each has duration at most d_{max}, there are at most $(k+1) * d_{max}$ such time steps.

Second, suppose that no request r_i is running at time t. In this case we prove that $den_t(R) \geq 1/c$. Let $l \leq k$ be the largest number such that $t < t_l$ (since $t_0 = t_{GREEDY}$, such an l exists). It follows that the request r_l was not yet scheduled at time t. The GREEDY property (Lemma 3.4) guarantees that there exists a set of requests R' running at time t with density $den_t(R') > 1 - b_l$ all using an edge e on the path used by r_l. From the assumption about the bandwidths it follows that $den_t(R') \geq 1/c$: either it follows since $b_l \leq 1 - 1/c$; or it follows since R' is nonempty and the bandwidth of any request is at least $1/c$. Now we prove that $R' \subseteq R$. Fix a request $r \in R'$ and look at the vertex z_r in ST. By the choice

of l and the construction of w_{l+1} (or the fact that $k = l$), z_r is not a descendant of w_l. Since the paths used by r_l and r are reasonable, it follows by Lemma A.1 that z_r is on the path from the root to w_l, and thus $z_r = w_{l'}$ for some $l' \leq l$. It follows that $r \in R$ and hence $R' \subseteq R$ and $den_t(R) \geq den_t(R') \geq 1/c$.

We have proved that for all but $(k+1) * d_{max}$ time steps t, $den_t(R) \geq 1/c$. Therefore, summing over all t,

$$den(R) \quad \geq \quad (t_{GREEDY} - (k+1) * d_{max})/c,$$

and

$$t_{GREEDY} \quad \leq \quad c * den(R) + (k+1) * d_{max}.$$

In the optimal schedule, any request from R_i must be scheduled using an edge from a separator $S_{w_{i'}}$ for some $i' \leq i$, Therefore for each of the $den(R)$ bits of data there exist some separator S_{w_j}, $j \leq k$ which this bit has to cross in time at most t_{OPT}. Since the total capacity of all these separators is at most $size(ST)$, we have $den(R) \leq t_{OPT} * size(ST)$. Thus

$$\begin{aligned} t_{GREEDY} &\leq c * den(R) + (k+1) * d_{max} \\ &\leq c * t_{OPT} * size(ST) + (k+1) * d_{max} \quad \leq \quad (c+1) size(ST) * t_{OPT} \end{aligned}$$

since $k + 1 \leq size(ST)$ and $d_{max} \leq t_{OPT}$. Therefore GREEDY is $(c+1)size(ST)$-competitive. □

B Lower bounds on GREEDY's performance

In this section, we present two simple examples that show that

1. the competitive ratio of GREEDY, with respect to makespan, can be arbitrarily poor if the bandwidth of the requests are allowed to vary from between 0 and 1 on a linear array network with unit capacity edges, and
2. the competitive ratio of GREEDY, with respect to makespan, is $\Omega(\log n)$ on an n-node tree, even if all edge capacities, bandwidth requirements, and durations are 1.

While GREEDY is constant-competitive given unit bandwidth requirements, GREEDY can perform arbitrarily badly if the bandwidth requirements are not restricted, even on the linear array network. For example, since preemption is not allowed, a request requiring only a little bandwidth can block a request that arrives a little later that requires almost all of the bandwidth. This can result in a schedule as shown in Figure 2, proving that there is a lower bound of $n/2$ on the competitive ratio of GREEDY for a linear array network of n nodes even if all capacities are 1 and all durations are 1.

The next example, Figure 3, proves that GREEDY cannot be better than $(1/2)\log n$-competitive with respect to makespan on n-node binary trees with unit capacity edges, even if all requests have unit duration and unit bandwidth. In contrast, GREEDY is constant competitive on linear arrays with unit capacity edges, when all requests have unit bandwidth (and arbitrary durations).

C Lower bound for call admission on a tree network

This section provides a proof of Theorem 4.1, which is restated below.

Theorem C.1 *Consider a network that is a complete binary tree T of depth d with $n = 2^{d+1} - 1$ nodes and unit edge capacities. Suppose we have a deterministic call admission algorithm for requests with unit bandwidth requirements on such a network. Then the competitive ratio with respect to makespan is at least $\Omega(\frac{\log\log n}{\log\log\log n}) = \Omega(\frac{\log d}{\log\log d})$.*

Proof. The proof has the following outline. We first construct a set of requests which we use as input to the call admission algorithm. Second we describe the strategy the adversary will use to determine the duration of the requests. Next we provide a lower bound on the time the on-line algorithm needs to schedule this set of requests and finish by providing a better solution and thus bounding the competitive ratio.

The requests

As a naming convention for the nodes of the network each node $v = v_\alpha$ is indexed by a bit string α. The length of the string $|\alpha|$ is the depth of the node in the tree and its left and right sons are $v_{\alpha 0}$ and $v_{\alpha 1}$. A node v_α is a predecessor of node v_β if α is a initial segment of β, denoted $\alpha \prec \beta$. The root of the tree is v_ϵ. Let T_α denote the subtree of tree T rooted at node v_α.

All requests have unit bandwidth requirements and the source and destination nodes of the set of requests R are defined as follows:

$$R = \{(v_{\alpha 0}, v_\beta) \mid \alpha 1 \prec \beta \ \wedge \ |\beta| = n\}$$

The durations of the requests are determined dynamically by the adversary depending on the actions of the on-line call admission algorithm.

Adversary strategy

The *adversary* proceeds in phases. The goal of each phase is to set the durations of a small number of jobs so that the on-line algorithm makes little progress for some period of time, while it is possible to simultaneously find a subtree of the network and a subset of the unscheduled requests which constitute a smaller size problem of the same kind. After the adversary has performed l phases, the subtree is given by its root α_l and depth d_l (initially $\alpha_0 = \epsilon$, and $d_0 = d$). We define R_l to be the set of all requests whose durations were not specified during the first l phases; it always only contains requests within T_{α_l}, i.e. such that both the source and the destination are in T_{α_l}; in addition the source is never more than d_l levels below v_{α_l} (i.e. it is in the interesting subtree).

In each phase the adversary gives the algorithm one time step to schedule requests. After that he focuses on the set of nodes S_l which are at the level $d_l / \ln d$ of the tree T_{α_l}, with the intention to later use either the subtree above S_l or one of the subtrees rooted at some $\gamma \in S_l$. If for some such γ the algorithm schedules no request below γ, the adversary, in the NO REQUEST phase, removes all requests which do not lie completely within this subtree and continues with it. Note, the algorithm did not do any work in this subtree within the previous time step, which ensures that the algorithm has wasted time. If the algorithm does schedule at least one request below each such γ, the adversary, in the BLOCKING phase, sets the duration of these requests so that the on-line algorithm makes no progress during a time period of length $t = \frac{\ln d}{2 \ln\ln d}$, and then continues with the subtree between α_l and S_l. Note that the depth of the subtree decreases in t NO REQUEST steps about as much as in 1 BLOCKING step, which is proportional to the duration of the steps.

The invariant which the adversary preserves throughout the schedule says that (1) for any two nodes in the interesting part of the tree there still exists a request which has to use both of them, if

there was one at the beginning, and (2) all the remaining requests start in the interesting part of the tree. More formally,

1. For each pair of nodes $v_\gamma, v_\beta \in T_{\alpha_l}$ such that $\gamma 1 \prec \beta$ and $|\beta| = |\alpha_l| + d_l$, there exists $\beta' \succ \beta$, $|\beta'| = d$, such that $(v_{\gamma 0}, v_{\beta'}) \in R_l$.
2. For every $(v_{\gamma 0}, v_\beta) \in R_l$: $\gamma \in T_{\alpha_l}$ and $|\gamma| < |\alpha_l| + d_l$.

Adversary Strategy

$t := \frac{\ln d}{2 \ln \ln d}$;
$l := 0$; $\alpha_l := \epsilon$; $d_l := d$;
repeat until at least t_2 time steps have elapsed
  let the on-line algorithm schedule requests for one time step
  $S_l := \{\beta \in T_{\alpha_l} \mid |\beta| = |\alpha_l| + \lceil d_l / \ln d \rceil\}$;
  if there exist a $\beta \in S$ such that in this time step no request from R_l was scheduled within T_β
    NO REQUEST:
      Choose a β and $a \in \{0, 1\}$ such that no request from R_l scheduled in this time step intersects $T_{\beta a}$
      $\alpha_{l+1} := \beta a$; $d_{l+1} := d_l - \lceil d_l / \ln d \rceil - 1$;
      Remove all scheduled requests;
      Assign duration 0 to all requests that do not lie within $T_{\alpha_{l+1}}$;
  else
    BLOCKING:
      $\alpha_{l+1} := \alpha_l$; $d_{l+1} := \lceil d_l / \ln d \rceil$;
      for every $\beta \in S_l$,
        Choose $r_\beta \in R_l$ to be one request scheduled in this time step within T_β;
        Assign a duration to r_β so that it will finish t time steps from the beginning of this phase
      Remove all other scheduled requests (i.e., except $\{r_\beta \mid \beta \in S_l\}$);
      Assign duration 0 to all requests which do not use an edge of $\{(v_{\gamma 0}, v_\gamma) \mid (\exists \beta, \beta') : r_\beta = (v_{\gamma 0}, v_{\beta'})\}$ and to all requests that use no vertex of depth less than $|\alpha_{l+1}| + d_{l+1}$;
  $l := l + 1$;
end repeat
Assign duration 0 to all remaining requests;
Remove all currently running requests;

Correctness of the Adversary Strategy

In this section we analyze the adversary strategy from the previous section. Throughout the analysis we assume that d is sufficiently large, and in particular $t > 1$.

First, note that in the BLOCKING phase it is always possible to choose a as required. Any request that intersects T_β has to use an edge from β to its parent since no request is scheduled within T_β, by the choice of β. There can be only one such request, and it can intersect only one of $T_{\beta 0}$ and $T_{\beta 1}$.

Second, the invariant stated above is preserved by the adversary strategy. It is obviously true at the beginning of the schedule and preserved during each NO REQUEST phase. Let us suppose that

the invariant was true before BLOCKING phase l. Invariant (2) is preserved trivially. Fix $\gamma, \beta \in T_{\alpha_{l+1}}$ such that $\gamma 1 \prec \beta$ and $|\beta| = |\alpha_{l+1}| + d_{l+1}$. From the definition of α_{l+1} and d_{l+1} it follows that $\beta \in S_l$. Let $r_\beta = (v_{\gamma' 0}, v_{\beta'})$ (i.e., the request under β scheduled during phase l with duration more than 1). By the invariant (2) for l, there is $\beta'' \succ \gamma' 0$ such that $|\beta''| = |\alpha_l| + d_l$. Using the invariant (1) for l on γ and β'', there exists a request with source $v_{\gamma 0}$ which uses the vertex $v_{\beta''}$ and hence also v_β; thus the invariant (1) is true for $l+1$ as well.

Third, we need to show that $d_l > 0$ for at least t^2 time steps.

Lemma C.2 *If the time spent in the first l phases of the adversary strategy is $t_l \leq t^2$, then $d_l \geq d(\ln d)^{-t_l/t} \geq 2 \ln d$.*

Proof. First we prove the second inequality. We have $d(\ln d)^{-t_l/t} \geq d(\ln d)^{-\ln d/2 \ln\ln d} = d^{1/2} \geq 2 \ln d$ for sufficiently large d.

Now we prove the first inequality by induction on l. For $l = 0$ it holds trivially.

Suppose that the lth phase is BLOCKING. Then $d_{l+1} = \lceil d_l / \ln d \rceil \geq d_l / \ln d$. The induction step follows since the time spent in this phase is t.

Suppose that the lth phase is NO REQUEST. Then $d_{l+1} = d_l - \lceil d_l / \ln d \rceil - 1 \geq d_l - 2d_l / \ln d = d_l(1 - 2/\ln d)$ (using the induction assumption $d_l \geq 2 \ln d$). To finish the induction step, we need to prove that $1 - 2/\ln d \geq (\ln d)^{-1/t}$, since the time spent in this phase is 1. We have $(1 - 2/\ln d)^t \leq 1 - 2t/\ln d = 1 - 1/\ln\ln d > 1/\ln d$ for sufficiently large d. □

As a consequence of this lemma and the invariant, R_l is nonempty for all l, and the length of the schedule generated by the algorithm is at least t^2.

Off-line solution

For the proof of the lower bound we still need to prove that an off-line call admission algorithm for the tree network can schedule the requests in less time.

The next lemma shows that all the requests can be scheduled efficiently. It is easy to schedule the requests with duration 1. For the requests with longer duration we use the fact that their conflict graph is similar to the one used in the example showing that GREEDY is $\Omega(\log N)$ competitive for tree network.

Lemma C.3

1. *All the requests scheduled during the NO REQUEST phases can be scheduled in time at most* 1.
2. *All the requests with duration at most* 1 *scheduled during one BLOCKING phase can be scheduled in time at most* 1.
3. *All the requests with duration more than* 1 *scheduled during the BLOCKING phases can be scheduled in time at most* $2t$.

Proof.

1. The path used by any request scheduled in a NO REQUEST phase l is disjoint with the subtree $T_{\alpha_{l+1}}$, while all the requests scheduled later are within $T_{\alpha_{l+1}}$. This in particular means that any two requests scheduled in different NO REQUEST phases use disjoint paths and can be scheduled in parallel. Since all the requests scheduled during one NO REQUEST phase can be scheduled in time 1 using the schedule produced by the on-line algorithm, the total time needed to schedule them in parallel is 1.
2. Just use for these requests the same schedule as the on-line algorithm during that BLOCKING phase.

3. Suppose that two requests $r = (v_{\gamma 0}, v_\beta)$ and $r' = (v_{\gamma' 0}, v_{\beta'})$ intersect and are scheduled during BLOCKING phases l and l' with duration more than 1. Obviously $l \neq l'$, since the requests of non-zero duration scheduled during the same phase are disjoint. By symmetry we can assume $l < l'$. It follows from the algorithm that $|\gamma| > |\gamma'|$ and the edge $(v_{\gamma 0}, v_\gamma)$ is used by r', since otherwise r would be assigned duration 0 in BLOCKING phase l. (r cannot intersect $r_{\gamma''} \neq r'$ for $\gamma'' \in S_l$, since then it would be disjoint from r'.) It follows that each edge is used by at most two such requests, and all these requests can be partitioned into two sets of requests such that each set uses disjoint edges. Since the duration of all requests in this class is at most t, we can schedule them in time $2t$. □

Since each BLOCKING phase takes time t, there are at most t BLOCKING phases. Thus the last lemma guarantees that the makespan of the optimal schedule is at most $3t + 1$. Since the algorithm generated a schedule of length at least t^2, the competitive ratio is at least $t^2/(3t + 1) = \Omega(t) = \Omega(\log d / \log \log d)$. □

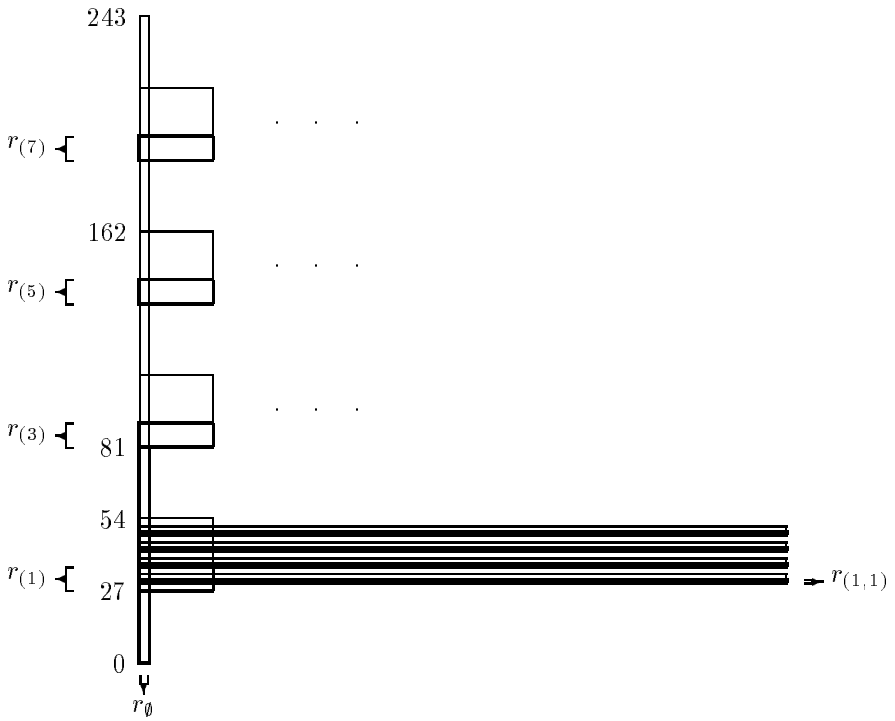

Figure 1: An example of the large set of requests for $T = 243$ and $c = 2$. The horizontal dimension denotes the nodes in the linear array, and the vertical dimension is the time. The thick rectangles denote the requests as if they were scheduled immediately after their arrival, the thin rectangles delimit the area within which each request has to be scheduled.

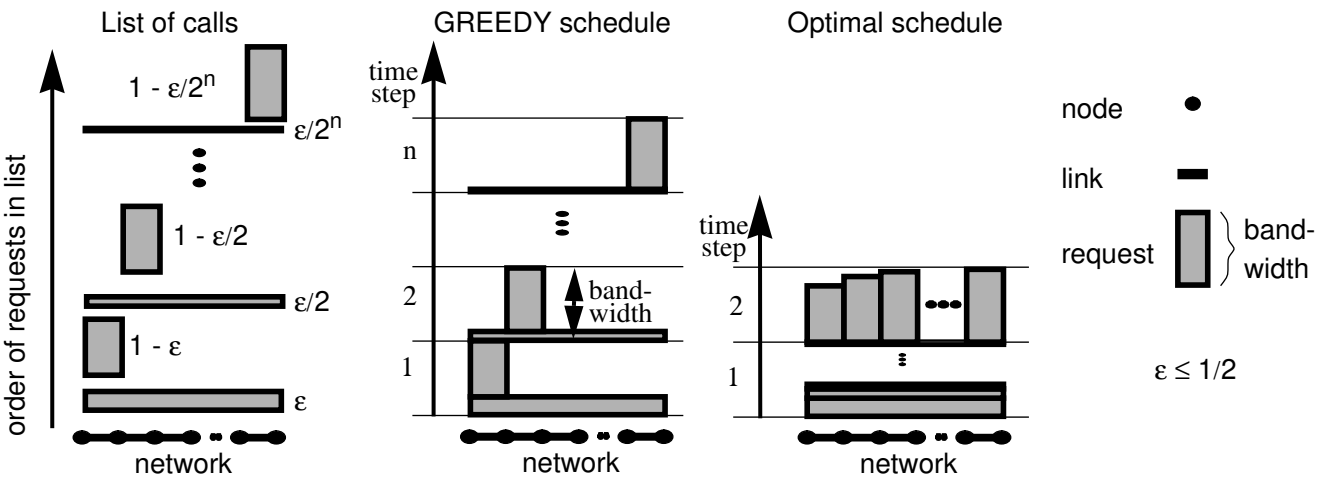

Figure 2: An example for a GREEDY schedule vs. an optimal schedule for an $(n+1)$-node linear array network. All requests arrive at time 0 and have unit duration, but their bandwidth requirements vary. In this figure the height of each request is proportional to its bandwidth requirement. Some of the requests span just a single edge of the network, while others span the entire network. In this figure width of a request is proportional to its span. The first picture shows the requests in the order they appear in the list of the GREEDY algorithm. The second picture shows the resulting GREEDY schedule while the third one gives the optimal schedule. In this example $\epsilon \leq 1/2$.

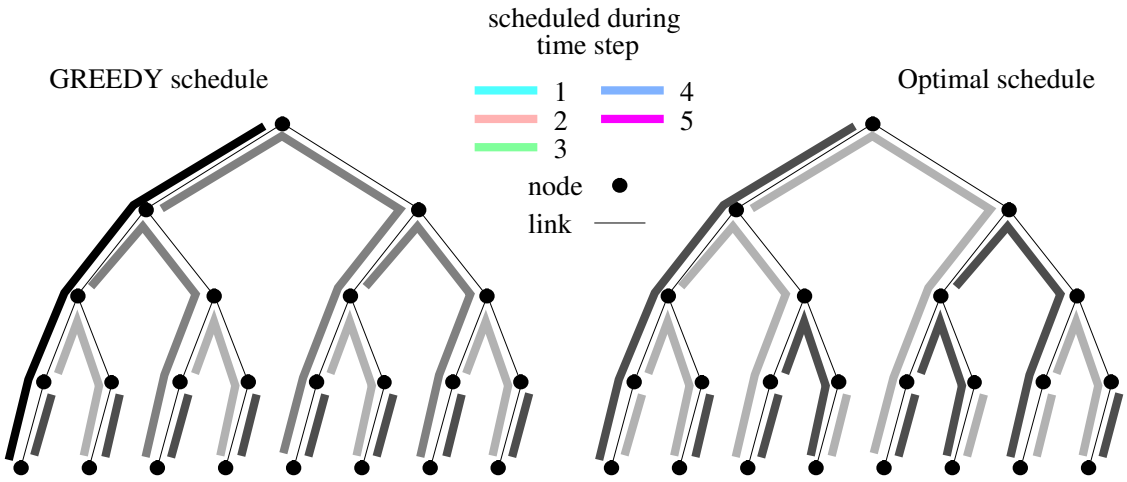

Figure 3: An example for a GREEDY schedule vs. an optimal schedule for a tree network. All capacities, bandwidth requirements, and durations are 1, and all requests arrive at time 0. The first picture shows the schedule produced by the GREEDY algorithm if the requests appear in the input list in the following order: if a request was scheduled at time t than it appears before all requests which were scheduled at a time greater than t. The second picture shows an optimal schedule for this set of requests.