Competitive Routing of Virtual Circuits with Unknown Duration*

Baruch Awerbuch[†] **Yossi Azar**[‡] **Serge Plotkin**[§] **Orli Waarts**[¶]

September 21, 1995

Abstract

In this paper we present a strategy to route *unknown duration* virtual circuits in a high-speed communication network. Previous work on virtual circuit routing concentrated on the case where the *call duration is known* in advance. We show that by allowing $O(\log n)$ reroutes per call, we can achieve $O(\log n)$ competitive ratio with respect to the maximum load (congestion) for the unknown duration case, were n is the number of nodes in the network. This is in contrast to the $\Omega(\sqrt[4]{n})$ lower bound on the competitive ratio for this case if no rerouting is allowed [3].

Our routing algorithm can be also applied in the context of machine load balancing of tasks with unknown duration. We present an algorithm that makes $O(\log n)$ reassignments per task and achieves $O(\log n)$ competitive ratio with respect to the load, where n is the number of parallel machines. For a special case of unit load tasks we design a constant competitive algorithm. The previously known algorithms that achieve up to polylogarithmic competitive ratio for load balancing of tasks with unknown duration dealt only with special cases of related machines case and unit-load tasks with restricted assignment [4, 11].

---

*A preliminary version of this paper was presented at the 5th ACM-SIAM Symposium on Discrete Algorithms, January 1994.

†Johns Hopkins University and Lab. for Computer Science, MIT. Supported by Air Force Contract TNDGAFOSR-86-0078, ARO contract DAAL03-86-K-0171, NSF contract 9114440-CCR, DARPA contract N00014-J-92-1799, and a special grant from IBM. E-Mail: baruch@theory.lcs.mit.edu.

‡Department of Computer Science, Tel Aviv University. E-Mail: azar@math.tau.ac.il. Research supported in part by Allon Fellowship and by the Israel Science Foundation administered by the Israel Academy of Sciences.

§Department of Computer Science, Stanford University. Research supported by U.S. Army Research Office Grant DAAL-03-91-G-0102, NSF Grant CCR-9304971, and by Mitsubishi Electric Laboratories. E-Mail: plotkin@cs.stanford.edu.

¶Department of Computer Science, Berkeley. A portion of this work was done while the author was at IBM Almaden Research Center. E-Mail: waarts@cs.berkeley.edu.

1 Introduction

1.1 Routing

Support for virtual circuits is one of the basic services provided by both existing and future high-speed communication networks. In order to use the network (say, transmit video signal from one point to another) the user requests a (virtual) circuit to be established between these points. Although the rate of information flowing through such a circuit might vary in time, the network has to guarantee that the circuit will support at least the bit rate that was agreed upon during the connection establishment negotiations. This guarantee is imperative for correct operation of many of the services, including constant bit-rate video and voice transmission. In other words, establishing a connection corresponds to reserving the requested bandwidth along some path connecting the end points specified by the user.

It is thus important to develop online strategies for virtual circuit routing that lead to provably efficient bandwidth utilization. As we will show, the problem appears to be especially hard when the duration of each virtual circuit is apriori unknown.

The routing problem considered in this paper is as follows. We are given a network where each edge has an associated capacity (*i.e.* bandwidth). Requests for establishment of connections arrive on-line; each request specifies the source and destination nodes, and the requested bandwidth; duration of the connection is not specified. Immediately upon arrival of a request, the algorithm establishes a connection by allocating the required bandwidth along some path between the source and the destination nodes. When a connection terminates, the bandwidth allocated to it is released. The algorithm can (possibly) *reroute* some of the existing connections.

A natural measure to evaluate the performance of the bandwidth allocation strategy is the "relative load" which is defined as the maximum congestion, *i.e.* maximum (over all links and over all moments in time) of the percentage of link-capacity utilization by the currently routed circuits. As usual, we use the notion of the *competitive ratio* [12], which in this case is the supremum, over all possible input sequences, of the ratio of the maximum relative load achieved by the online algorithm to the maximum relative load achieved by the optimal offline algorithm.

Observe that if the number of reroutings per connection is not limited, it is trivial to maintain optimum relative load, *i.e.* competitive ratio of 1. The other extreme is to totally disallow rerouting. For this case it is easy to adapt the lower bound of Azar, Broder, and Karlin [3] to show $\Omega(n^{1/4})$ lower bound for the competitive ratio, where n is the number of nodes in the network.[1] This should be compared with the case where the duration of each connection becomes known upon its arrival. For this case, it was shown in [1, 4] that one can achieve $O(\log nT)$-competitive ratio with no rerouting, where T is the ratio of the longest to the shortest duration of a connection. This indicates that the routing of virtual circuits with

[1]The result in [3] is an $\Omega(\sqrt{n})$ lower bound in the context of load-balancing. Adaptation of this lower bound to the routing model results in an $O(\sqrt[4]{n})$ lower bound.

unknown duration is significantly harder.

The main contribution of this paper is a routing strategy for *unknown duration* virtual circuits that is $O(\log n)$ competitive with respect to load and that reroutes each circuit at most $O(\log n)$ times. Our online algorithm is competitive even against an offline algorithm that is allowed to make any number of reroutings.

Informally, our routing strategy is based on maintaining a new *stability condition*. On one hand, this condition is strong enough to guarantee that the current load is within $O(\log n)$ factor of the optimum maximum load, and on the other hand it is weak enough so that the algorithm does not make more than logarithmic number of reroutings per call in order to maintain it. Moreover, this stability condition can be checked by considering only the current state of the routing algorithm and thus can be directly used to make rerouting decisions. In contrast to this, the condition used in [1] for proof of competitiveness, was based on both the state of the online and the state of the offline algorithms and hence can not be used to make rerouting decisions.

An alternative measure of network performance is the amortized throughput defined as the average over time of the number of bits transmitted by the accepted connections. In this setting, the network's bandwidth is assumed to be insufficient to satisfy all the requests so some of the requests may need to be rejected upon their arrival. An online algorithm in this setting is a combination of a decision mechanism that determines which requests to satisfy together with a strategy that specifies how to route these requests. The goal is to maximize the amortized throughput.

Competitive algorithms to maximize the throughput were provided by Garay and Gopal [8] (for the case of a single link); by Garay, Gopal, Kutten, Mansour and Yung [7] (for a line network); and by Awerbuch, Azar and Plotkin [2] (for general network topologies). None of these works provided competitive algorithms for connections with unknown durations. In view of the results presented in this paper it is natural to ask whether there exists a throughput-competitive algorithm for the case where the duration of calls is apriori unknown. However, as observed by Garay and Gopal [8], such an algorithm does not exist. Roughly speaking, the reasoning is as follows. First note that the throughput-maximization setting makes sense only if we allow the online algorithm to reject some of the calls. Now the fact that the call durations are apriori unknown means that an online algorithm may reject a call that would have otherwise continued to exist "forever", which shows that the competitive ratio of this online algorithm with respect to throughput is unbounded.

1.2 Load Balancing

In the second part of the paper we show how to apply our techniques to load balancing of tasks with unknown duration. Here, there are n parallel machines and a number of independent tasks. The tasks arrive one by one, where each task has an associate *load vector* and has to be assigned to exactly one of the machines, thereby increasing the *load* on this machine by an amount specified by the corresponding coordinate of the load vector. The duration of a task is not known upon its arrival. The objective is to minimize the maximum machine load.

The case where the duration of a task becomes known upon its arrival was extensively studied. In particular, the case where the duration is infinite was studied in [9, 10, 5, 6]; the case of finite but known durations was studied in [1, 2, 4]. In this paper we focus on the more general case where the durations of tasks are *apriori unknown*. For the related machine case, i.e the case that the load vectors are spanned by one vector, an algorithm that achieves a constant competitive ratio is shown in [4]. However, for the general case, Azar, Broder, and Karlin [3] showed that if reassignment of tasks is not allowed (*i.e.* once a task is assigned to a machine, it cannot be reassigned to another machine), then there is an $\Omega(\sqrt{n})$ lower bound on the competitive ratio. This lower bound holds even if we restrict the model to the case (introduced by [5]) where all coordinates of the load vector are either ∞ or 1. The $O(\sqrt{n})$-competitive algorithm presented in [4] matches this lower bound.

In order to overcome the above non-polylogarithmic lower bound, Phillips and Westbrook suggested to allow *task reassignments* [11]. In particular, for the case where all coordinates of the load vector are either ∞ or 1, they have presented an algorithm that achieves $O(\log n)$ competitive ratio with respect to load while making at most a constant amortized number of reassignments per task.

We show that our online routing techniques presented in the first part of this paper can be applied to the general load balancing case, *i.e.* load balancing of unknown duration tasks with no restrictions on the load vectors. This yields an algorithm that makes at most $O(\log n)$ reassignments per task and achieves $O(\log n)$ competitive ratio with respect to the load. It is not obvious how to extend the approach in [11] to achieve similar performance.

We also consider the special case where all coordinates of the load vector of task i are either ∞ or 1, and where the optimum load achieved by the offline algorithm is at least $\log n$. For this case we give an algorithm that achieves a *constant* competitive ratio while making only $O(\log n)$ amortized reassignments per task. As opposed to the algorithm in [11] and the other algorithms presented here that reassign tasks only as a result of task departures, this algorithm reassigns tasks as a result of a task arrival as well. In fact, it is easy to show that this is necessary in order to achieve sublogarithmic competitive ratio [5]. (Specifically, [5] shows an $\Omega(\log n)$ lower bound on the competitive ratio for the load balancing case where tasks never depart and no reroutings are allowed.)

2 Online Route Allocation

In this section we describe online strategy for routing (and rerouting) virtual circuits whose lifetime (duration) is apriori unknown. We are given a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, and a capacity function $u : E \rightarrow R^+$. The fact that each connection has an associated lifetime that is apriori unknown is modeled by assuming that the algorithm receives two types of requests online: "initiate connection" and "terminate connection". We say that connection i is *active* at time t if the "initiate connection i" request arrives before t and the "terminate connection i" request arrives after t. The "initiate connection" request is defined by a tuple

(s_i, t_i, p_i), where $s_i, t_i \in V$ are the source/sink pair, and $p_i \in R^+$ corresponds to the required bandwidth. As long as the connection is active, the required bandwidth has to be reserved along some path P_i from the source s_i to the sink t_i. The routing algorithm is allowed to change P_i at any time; such change is referred to as *rerouting.*

If we were to use edge e to route connection i, the load on this edge would have increased by p_i. Thus, we define the *relative load* on e due to connection i as $p_{i,e} = p_i/u(e)$. Note that $p_{i,e}$ is defined for every edge e, and not only for the edges that actually participate in the path P_i currently assigned to connection i by the algorithm. Let $\mathcal{P}_t$ and $\mathcal{P}_t^*$ be the set of paths associated with the connections that are active at time t by the online and the offline algorithms, respectively. The *relative load* on edge e at instance t of the online algorithm is defined by:

$$\ell_e(t) = \sum_{P_i \in \mathcal{P}_t : e \in P_i} p_{i,e}$$

Let $\lambda(t) = \max_{e \in E} \ell_e(t)$ and $\lambda = \max_t \lambda(t)$. Similarly, define $\ell_e^*(t)$, $\lambda^*(t)$ and λ^* to be the corresponding quantities for the routes $\mathcal{P}_t^*$ produced by the offline algorithm. The goal of the online algorithm is to produce and maintain a set of routes that minimizes λ/λ^*. Minimizing λ/λ^* corresponds to asking how much larger we should make the capacities of the edges in order for the online algorithm to be able to satisfy all the requests on the new graph that the offline algorithm could have satisfied on the graph with the original capacities.

2.1 Routing Algorithm

We first assume that the algorithm has a knowledge of $\Lambda \geq \lambda^*$. This assumption can be easily dealt with by a simple doubling technique, which we discuss later. To simplify the formulas, we will use tilde to denote normalization by Λ, for example $\tilde{\ell}_e(t) = \ell_e(t)/\Lambda$.

Define $a = 1 + \gamma/2$ for some constant $\gamma < 1$. At every instance, each active connection i is associated with some path P_i. Since the algorithm is allowed to reroute connections, we define $T(P_i)$ as the last time when the connection was rerouted to use path P_i. For a route P that exists at time t and that uses edge e, let $Q_e^P(t)$ be the set of routes in the network at time t which use edge e and that have smaller values of $T(\cdot)$, *i.e.* paths that were using e before P. Define the *height* of a route P with respect to an edge e at time t as

$$h_e^P(t) = \sum_{P_j \in Q_e^P(t)} p_{j,e}$$

and the weight of a route P which satisfies request i as

$$W_i^P(t) = \sum_{e \in P} (a^{\tilde{h}_e^P(t) + \tilde{p}_{i,e}} - a^{\tilde{h}_e^P(t)})$$

Note that the weight of the first path connection i is assigned to immediately after its arrival is:

$$\sum_{e \in P} (a^{\tilde{\ell}_e(t)} - a^{\tilde{\ell}_e(t) - \tilde{p}_{i,e}}),$$

where t is the time immediately after the assignment.

From now on we will omit the parameter t where it can be deduced from the context. As we will show, it is sufficient to maintain the following *stability condition*:

Definition 2.1 Let P be some existing $s-t$ route satisfying active connection i, and let P' be any $s-t$ path in G. We say that the algorithm is in a *stable state* if for any P and P', we have:

$$W_i^P = \sum_{e \in P} (a^{\tilde{h}_e^P + \tilde{p}_{i,e}} - a^{\tilde{h}_e^P}) \le 2 \sum_{e \in P'} (a^{\tilde{\ell}_e + \tilde{p}_{i,e}} - a^{\tilde{\ell}_e}).$$

Intuitively, the main idea of the algorithm is to make sure that the above stability condition is satisfied. More precisely the algorithm is described as follows:

Route/Reroute Alg:

- Upon receiving the "initiate connection" request (s_i, t_i, p_i), the algorithm assigns it to an $s_i - t_i$ path P_i which minimizes W_i^P.
- If at any moment the stability condition is not satisfied by some connection i that is currently assigned to some path P, the algorithm reroutes i to use a path P_i' that minimizes $W_i^{P'}$.
- When a connection terminates, it is removed from the path it is assigned to, reducing the relative load on the edges of the path.

Observe that the algorithm will never reroute as a result of an arrival of a new connection.

The proof of performance of the algorithm is divided into two parts. First, we prove that if the stability condition is maintained then the current relative load can be bounded by a function of the optimum load. Next we use this fact to show that after any departure, the algorithm can move to the stable state by making only a small number of reroutings.

Lemma 2.2 If the algorithm is in a stable state, than $\sum_{e \in E} a^{\tilde{\ell}_e} \le m/(1-\gamma)$.

Proof: Consider a connection i that is currently assigned by the online algorithm to some path P. Let P^* be the path that is currently assigned to this connection by the offline algorithm. Note that since $a = 1 + \gamma/2$ we have that $\forall x \in [0,1]: \quad 2(a^x - 1) \le \gamma x$. Moreover, the fact that the offline algorithm routes the ith request through P^* implies that for each $e \in P^*$ we have $0 \le \tilde{p}_{i,e} \le 1$, and hence the inequality above applies for $x = \tilde{p}_{i,e}$. This and the stability condition imply

$$
\begin{aligned}
W_i^P &= \sum_{e\in P}(a^{\tilde{h}_e^P+\tilde{p}_{i,e}} - a^{\tilde{h}_e^P}) \le 2\sum_{e\in P^*}(a^{\tilde{\ell}_e+\tilde{p}_{i,e}} - a^{\tilde{\ell}_e}) \\
&= 2\sum_{e\in P^*} a^{\tilde{\ell}_e}(a^{\tilde{p}_{i,e}} - 1) \le \gamma \sum_{e\in P^*} a^{\tilde{\ell}_e}\tilde{p}_{i,e}
\end{aligned}
$$

Summing over all currently active connections, we get:

$$
\sum_{P\in\mathcal{P}}\sum_{e\in P}(a^{\tilde{h}_e^P(t)+\tilde{p}_{i,e}} - a^{\tilde{h}_e^P(t)}) \le \gamma \sum_{P^*\in\mathcal{P}^*}\sum_{e\in P^*} a^{\tilde{\ell}_e}\tilde{p}_{i,e}
$$

Exchanging the order of summation yields

$$
\sum_{e\in E}\sum_{P\in\mathcal{P}|e\in P}(a^{\tilde{h}_e^P(t)+\tilde{p}_{i,e}} - a^{\tilde{h}_e^P(t)}) \le \gamma \sum_{e\in E} a^{\tilde{\ell}_e} \sum_{P^*\in\mathcal{P}^*|e\in P^*} \tilde{p}_{i,e}
$$

Clearly, the left hand-side is a telescopic sum for each edge e. Observe that the fact that the normalized load of the offline algorithm never exceeds 1 implies that $\sum_{P^*\in\mathcal{P}^*|e\in P^*} \tilde{p_{i,e}} \le 1$. Thus we conclude that

$$
\sum_{e\in E}(a^{\tilde{\ell}_e} - 1) \le \gamma \sum_{e\in E} a^{\tilde{\ell}_e}
$$

Using the fact that $\gamma < 1$, we get

$$
\sum_{e\in E} a^{\tilde{\ell}_e} \le m/(1-\gamma).
$$

∎

Lemma 2.3 A connection that arrived when the algorithm was in a stable state will not be rerouted more than $O(\log n)$ times.

Proof: Consider a request for connection i with source s_i and sink t_i that arrived when the algorithm was in a stable state. Let Q_i be a path that minimizes $Z_i^P = \sum_{e\in P}(a^{\tilde{p}_{i,e}} - 1)$. Note that $Z_i^{Q_i}$ is a lower bound on the weight of a path associated with this connection at any time.

By assumption, connection i arrived when the algorithm was in a stable state. Hence, if we would have assigned this connection to Q_i upon its arrival, then by Lemma 2.2, its weight would have been

$$
\sum_{e\in Q_i}(a^{\tilde{\ell}_e+\tilde{p}_{i,e}} - a^{\tilde{\ell}_e}) \le m/(1-\gamma)\sum_{e\in Q_i}(a^{\tilde{p}_{i,e}} - 1) = m/(1-\gamma)Z_i^{Q_i}.
$$

Since connection i was assigned to a path with minimum weight, the original weight of the connection is no greater than the weight it would have had if it had been assigned to Q_i, and hence it is at most $m/(1-\gamma)Z_i^{Q_i}$. Arrival, termination, and reroutings of other connections can not increase the weight associated with this connection, and each rerouting of i decreases the weight of the route assigned to i by at least a factor of 2. Thus, since the final weight of the connection is at least $Z_i^{Q_i}$ we conclude that the number of reroutings per connection is bounded by $\log(m/(1-\gamma)) = O(\log n)$. ∎

Theorem 2.4 The route/reroute algorithm maintains load of at most $O(\Lambda \log n)$ while rerouting each connection at most $O(\log n)$ times.

Proof: Clearly this algorithm starts in a stable state. Note that arrival of new connections can not affect the stability condition. Departure of a connection might cause several reroutings. Since, inductively, the connections that are rerouted arrived when the algorithm was in a stable state, Lemma 2.3 implies that the rerouting will terminate and a stable state will be reached again. Lemma 2.2 then implies that at this point we have

$$\sum_{e \in E} a^{\tilde{\ell}_e} \le m/(1-\gamma).$$

This, in turn implies that

$$\lambda = \max_t \max_{e \in E} \ell_e(t) \le \Lambda \log_a(m/(1-\gamma)) = O(\Lambda \log n).$$

Since every connection arrives when the algorithm is in a stable state, Lemma 2.3 implies that the total number of reroutings per connection is bounded by $O(\log n)$. ∎

We can eliminate the need to know the optimal load in advance by the standard doubling technique. Initiate Λ to be the minimum $p_{1,e}$ and if at some time the current load exceeds the competitive ratio times Λ, then double Λ and ignore all of the connections that currently exist in the system. This increases the competitive ratio by at most a factor of 4 and thus it remains at most $O(\log n)$.

3 Online Machine Load Balancing

In this section we consider the online load balancing problem defined as follows. Tasks arrive and depart online. At any instance, each active task, *i.e.* task which has arrived but has not terminated yet, has to be assigned to one of the n machines. The finishing times of the tasks are apriori unknown. The algorithm is allowed to reassign tasks from one machine to another.

Each task j is defined by a vector $p_j = (p_{1j}, p_{2j}, \ldots, p_{nj})$; for the period that task j is assigned to machine i it increases the load on i by p_{ij}. The goal is to minimize the ratio between the maximal load of the online algorithm to that of the offline one while executing only a small number of reassignments.

As usual, one can categorize load balancing problems for tasks with unknown duration into classes according to the properties of the vectors p_j. The most general case is the *unrelated machines* case, where we place no restrictions on these vectors. In the next section we discuss this case and show that the routing strategy presented above can be modified to provide a competitive load balancing algorithm for this case.

Another interesting case is when all p_{ij} are either 1 or ∞ and where the optimum load achieved by the offline algorithm is at least $\log n$. In other words, each task is associated with a subset of machines that can execute it; assigning the task to one of these machines causes a unit increase in the load. In Section 3.2 we present an algorithm that achieves a *constant* competitive ratio with respect to load for this case, while performing $O(\log n)$ amortized number of reassignments per task.

3.1 The Unrelated Machines Case

The routing problem discussed in the previous section can be considered as a generalization of the unrelated machines scheduling problem. Roughly speaking, in the load balancing problem, each task can be assigned to one of the machines in a given set, increasing the load on this machine. In the routing context, assigning a connection to a route increases the load on a *subset* of edges, where the subsets correspond to possible $s-t$ routes.

It is easy to see that the routing algorithm presented in the previous section works even for the case where the graph is directed and where the values of relative load $p_{i,e}$ are arbitrary and are not necessarily related to $p_i/u(e)$. This allows us to reduce an instance of the load balancing problem on unrelated machines to online "generalized routing" in the following way. Construct a directed graph with 2 vertices, s and t, and n parallel edges between them. Edge i corresponds to machine i. Given a sequence of tasks arrivals and departures, we generate a corresponding sequence of requests to initiate connections and to terminate connections. An arrival of task j with a corresponding load vector p_j is translated into an "initiate connection j" request between s and t with precisely the same load vector. Observe that assignment of task j to machine i corresponds to using the unique $s-t$ path through the ith edge to satisfy the corresponding request "initiate connection j". This immediately implies:

Theorem 3.1 For the unrelated machines problem where the duration of tasks is apriori unknown, there is an assignment algorithm which makes $O(\log n)$ reassignments per task and achieves $O(\log n)$ competitive ratio with respect to the load.

In [5], an $\Omega(\log n)$ lower bound was proved on the competitive ratio for the load balancing case where tasks never depart. Observe that our algorithm reassigns tasks only as a result of task departures, and hence can not achieve better than $O(\log n)$ competitive ratio with respect to load.

3.2 The Case of Unit-load Tasks

In this section we consider the special case where all p_{ij} are either 1 or ∞, and where the optimum load achieved by the offline algorithm is at least $\log n$ (*i.e.* the size of a job is at most $1/\log n$ times the optimal maximal load). In other words, each task has unit weight and can be assigned to one of the machines that belong to a subset associated with this task. We describe an algorithm which maintains a *constant* competitive ratio while performing $O(\log n)$ amortized number of reassignments per task.

As before we assume that the algorithm has a knowledge of $\Lambda \geq \lambda^*$, where λ^* is the optimum load achieved by the offline algorithm. This assumption can easily be dealt with by a simple doubling technique. The algorithm will maintain the following *stability condition*:

Definition 3.2 Let j be some task which is currently assigned to machine i. Consider a machine i' which is an eligible assignment for task j (*i.e.* machine i with $p_{i'j} = 1$). We say that the algorithm is in a *stable state* if for any i and i', we have:

$$\ell_i - \ell_{i'} \leq 2\Lambda/\log n$$

Similarly to the routing algorithm of Section 2.1, the main idea of the algorithm is to make sure that the above stability condition is satisfied. More precisely the algorithm is described as follows:

Load Balancing/Rebalancing Alg:

- Each new task j is assigned to an eligible machine that currently has minimum load.
- If at any moment the stability condition is not satisfied by some task j that is currently assigned to machine i, the algorithm reassigns j to a least loaded machine among the machines that are eligible with respect to j.
- When a task finishes, it is removed from the machine it is assigned to, reducing the load on this machine.

Observe that, as opposed to the algorithms described previously, this algorithm reassigns tasks both as a result of task arrival and departure. As mentioned in Section 3.1, this is necessary to achieve constant competitive ratio, since the lower bound of [5] implies that an algorithm that does not reassign tasks as a result of task arrivals can not achieve better than $\Omega(\log n)$ competitive ratio.

Again, the proof of performance of the algorithm is divided into two parts. First, we prove that if a stable condition is attained, then the current load is within a constant factor of optimum load. Then we show that after any arrival or departure, a stable condition can be reached by making only a small number of reassignments.

Lemma 3.3 If the algorithm is in a stable state, than $\lambda \leq 4\Lambda$.

Proof: Assume that at some instance there is a machine i such that $\ell_i > 4\Lambda$. We claim that at that time, for any $0 \leq k \leq \lfloor \log n \rfloor$ there is a set S_k of at least 2^k machines such that

$$\forall i \in S_k; \quad \ell_i > \Lambda(4 - 2k/\log n)$$

Thus for $k = \lfloor \log n \rfloor$ it follows that there are more than $n/2$ machines with load larger than 2Λ. This yields a contradiction since at every instance in time, the cumulative load on all the machines has to be below $n\lambda^* \leq n\Lambda$.

We prove the claim by induction on k. By assumption, the induction holds for $k = 0$. Assume the claim is correct for k. Consider the tasks currently assigned to machines in the set S_k. The fact that $k \leq \log n$ implies $\Lambda(4 - 2k/\log n) \geq 2\Lambda$. Thus,

$$\sum_{i \in S_k} \ell_i \geq 2k\Lambda.$$

Since the optimal load on each machine is bounded by Λ, the offline algorithm must assign the tasks which are currently assigned to machines in S_k to a set of machines S_{k+1} of size at least $2k$. Consider any machine i in $S_{k+1} - S_k$. By construction of S_{k+1}, machine i is eligible with respect to some task j that is currently assigned to one of the machines $i' \in S_k$. The stability condition implies that the load on i is not much lower than the load on i'. More precisely, $\ell_i \geq \ell_{i'} - 2\Lambda/\log n$. Since $i' \in S_k$ we can use the induction hypothesis to get:

$$\forall i \in S_{k+1}; \quad \ell_i > \Lambda(4 - 2(k+1)/\log n).$$

This completes the inductive proof and therefore the proof of the Lemma. ∎

Lemma 3.4 If every task arrives when the algorithm is in a stable state then the total number of reassignments is at most $O(\log n)$ times the number of tasks.

Proof: Define the following non-negative potential function

$$\Phi = \sum_i (\ell_i/\Lambda)^2$$

We will bound the changes in the value of this function that are caused by arrivals, departures and reassignments of tasks. Clearly task departures can only cause Φ to decrease. Next we consider tasks arrivals. If a task arrived when the algorithm was in a stable state, and was assigned to some machine i whose load before the assignment was ℓ_i, then by Lemma 3.3 we get that the increase in Φ is bounded by:

$$\Delta\Phi = ((\ell_i + 1)/\Lambda)^2 - (\ell_i/\Lambda)^2 = (2\ell_i/\Lambda + 1/\Lambda)/\Lambda \leq 9/\Lambda.$$

For tasks reassignments, if a task is reassigned from machine i to machine i' then Φ is reduced by at least:

$$
\begin{aligned}
-\Delta\Phi &= (\ell_i^2 - (\ell_i - 1)^2 + \ell_{i'}^2 - (\ell_{i'} + 1)^2)/\Lambda^2 \\
&= 2(\ell_i - \ell_{i'} - 1)/\Lambda^2 \geq 2(2/\log n - 1/\Lambda)/\Lambda
\end{aligned}
$$

The last inequality follows from the fact that the job was reassigned from i to i' because it did not satisfy the stability condition. Now we use the fact that $\Lambda \geq \log n$ to conclude that $-\Delta\Phi \geq 2/(\Lambda \log n)$ The claim of the lemma follows from the fact that at any instance in time we have $\Phi \geq 0$. ▮

Theorem 3.5 The assignment algorithm maintains load of at most 4Λ with amortized $O(\log n)$ reassignments per task.

Proof: Clearly the algorithm starts in a stable state. Since, inductively, new tasks arrive when the algorithm is in a stable state, Lemma 3.4 implies that the reassignment process terminates, a stable state is reached, and the amortized number of reassignments per task is limited by $O(\log n)$. Lemma 3.3 implies that the algorithm maintains load of at most 4Λ. ▮

As before we eliminate the need to know the optimal load in advance by the doubling technique. This increases the competitive ratio by at most a factor of 4 to be 16.

References

[1] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 623–631, May 1993.

[2] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive on-line routing. In *Proceedings of 34th IEEE Annual Symposium on Foundations of Computer Science*, pages 32–40, November1993.

[3] Y. Azar, A. Broder, and A. Karlin. On-line load balancing. In *Proceedings of the 33rd IEEE Annual Symposium on Foundations of Computer Science*, pages 218–225, 1992.

[4] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. Pruhs, and O. Waarts. On-line load balancing of temporary tasks. In *Proceedings of the Workshop on Algorithms and Data Structures*, pages 119–130, August 1993.

[5] Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignment. In *Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms*, pages 203–210, 1992.

[6] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 51–58, May 1992.

[7] J. Garay, I. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithms. In *Proceedings of the of 2nd Annual Israel Conference on Theory of Computing and Systems*, 1993.

[8] J.A. Garay and I.S. Gopal. Call preemption in communication networks. In *Proceedings of IEEE INFOCOM '92*, volume 44, pages 1043–1050, Florence, Italy, May 1992.

[9] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[10] R. Karp, U. Vazirani, and V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings f the 22nd Annual ACM Symposium on Theory of Computing*, pages 352–358, 1990.

[11] S. Phillips and J. Westbrook. Online load balancing and network flow. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 402–411, 1993.

[12] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Comm. ACM*, 28(2):202–208, 1985.