Combining Fairness with Throughput: Online Routing with Multiple Objectives

Ashish Goel *   Adam Meyerson †   Serge Plotkin ‡

October 12, 1999

Abstract

This paper presents online algorithms for routing and bandwidth allocation which simultaneously approximate fair and max-throughput solutions. In fact, the algorithms solve a more difficult problem: for any bandwidth b, the number of sessions that get bandwidth b in the online algorithm is not smaller than the number of sessions receiving γb offline, where γ is the competitive ratio. This problem is provably harder than the problem of maximizing throughput (e.g. [4]) or the problem of maximizing the bandwidth assigned to the most starved session (e.g. [3]).

For the case where the algorithm assigns bandwidths, we present an $O(\log^2 n \log^{1+\epsilon} U/\epsilon)$-competitive algorithm, for any ϵ, where U is the minimum (over all choices of routes) of the maximum number of sessions routed along any single link. We also show an $\Omega(\log^{1+\epsilon} U/\epsilon)$ lower bound in this model.

For a more practically interesting model where the algorithm assigns routes and *weights*, and where these weights are used to drive the Weighted Fair Queuing policy in the routers, we present an $O(\log^2 n \log U)$-competitive algorithm. We also show that the dependence on U is necessary by presenting an $\Omega(\sqrt{\log U})$ lower bound.

The upper and lower bounds presented in [4] for online maximization of throughput become invalid if we are allowed to assign weights. We prove an $\Omega(\log n)$ lower bound for this model and present an $O(\log n \log \log n)$-competitive online algorithm.

We present preliminary simulation results which show that our algorithm is effective in attaining high throughput without significantly sacrificing fairness.

*Department of Computer Science, University of Southern California, Los Angeles CA 90089-0781. Email: agoel@cs.usc.edu.

†Department of Computer Science, Stanford University CA 94305. Email: awm@cs.stanford.edu.

‡Supported by ARO Grants DAAG55-98-1-0170 and ONR Grant N00014-98-1-0589. Department of Computer Science, Stanford University CA 94305. Email: plotkin@cs.stanford.edu.

1 Introduction

Simultaneous fairness and throughput In a best-effort communication network two natural goals are to divide the available resources (bandwidth) as fairly as possible and, at the same time, to maximize the total bandwidth allocated to the users (throughput).

The accepted definition of fairness is *max-min fairness* [8]. Roughly speaking, bandwidth allocation is considered max-min fair if "poor" sessions cannot increase their bandwidth by stealing from "richer" sessions that share links with them. Since this definition assumes that the routes are given, the standard approach consists of two independent steps. The first step computes the routes, and the second step divides the bandwidth along these routes in a max-min fair way. The first step can be implemented using algorithms such as in [4, 3]. Several efficient distributed algorithms that implement the second step are known [1, 2, 5, 7].

Unfortunately, as long as the final step implements exact max-min fairness, it is impossible to approximate globally optimum total throughput. (Locally maximum throughput, i.e. maximum throughput without rerouting, can be approximated by distributed algorithms such as in [6]). On the other hand, by "averaging" fair and throughput-optimum solutions one can always exhibit bandwidth allocation that is within a factor of two of maximum throughput and where each session receives at least half of the bandwidth it would get in a fair solution. In this paper we address the problem of getting close to such allocation *online*.

Since max-min fairness assumes that routes are given, we need a more general fairness definition which will be equivalent to max-min fairness when the routes are already known. Kleinberg *et al.* [11] proposed the following definition of fairness in a variable-route scenario. Compute a *bandwidth vector* composed of the bandwidth allocated to connections, in increasing order. The ith coordinate is thus equal to the bandwidth of the connection receiving ith least bandwidth. Also define a *prefix vector* where the ith coordinate is equal to the sum of the bandwidths assigned to the i minimum sessions (i.e. the sum of the first i coordinates from the previously computed vector). Note that the first coordinate of the prefix vector represents the bandwidth given to the "most starved" connection while the last coordinate is equal to the total throughput. An allocation with *lexicographically maximum* bandwidth vector is considered *globally fair*. It's not difficult to show that the allocation with the larger bandwidth vector also has the larger prefix vector. If routes are preassigned, the globally fair assignment of bandwidths is identical to the max-min fair allocation [8].

Using the above definition of global fairness, we can restate our goal as follows: We need to design an online algorithm which assigns routes and bandwidths so as to guarantee that throughput is within polylogarithmic factors of optimal and each term of the bandwidth vector is within polylogarithmic factors of the globally fair bandwidth vector.

Unfortunately, the above stated criterion does not always lead to reasonable allocation of bandwidth. There are cases where one can start with a solution which achieves close to optimal throughput and assigns every session close to its globally fair bandwidth, and increase the bandwidth assigned to many "starved" connections by a polynomial factor, while reducing the total throughput by less than a constant factor. For example, consider a path that consists of n unit-capacity edges. Assume that there are n requests going along the line and, in addition, each link has $\sqrt{n}$ requests using that single link. Maximum throughput is n; max-min fairness assigns each connection $1/(n+\sqrt{n})$ bandwidth and achieves $\Theta(\sqrt{n})$ throughput. One can assign $1/2$ to n "short connections", one per link, and assign $1/2(n+\sqrt{n})$ to all the other connections. Observe that this achieves a constant fraction of the optimum throughput while each connection in this assignment gets at least a constant fraction of its "fair bandwidth". Now consider an alternative assignment, where each short connection gets $1/(2\sqrt{n})$ and each long one gets $1/(2n)$. Intuitively, the last assignment is much better.

Our results The above example indicates that we need to modify our goal. Instead of trying to approximate the throughput and the globally fair bandwidth vector, we will simultaneously approximate *all of the possible bandwidth vectors*. More precisely, our goal is to design an algorithm that assigns routes and bandwidths such that its corresponding bandwidth vector is coordinate-wise at least a $1/\gamma$ fraction of *any possible bandwidth vector*. We call γ the competitive ratio of the algorithm. A γ-competitive bandwidth allocation guarantees, for any b, that the number of connections which were assigned bandwidth $\geq b$ is not lower than the number of connections which were assigned bandwidth $\geq \gamma b$ *in any legal solution*. It's fairly straightforward to show that our prefix vector is also coordinate-wise at least the same γ fraction of any possible prefix vector. Thus a γ-competitive bandwidth allocation also achieves throughput which is within a γ factor of optimum.

While we present such an algorithm in Section 5, we concentrate on a slightly different model, which we believe to be much more appealing from the practical perspective. We assume that the algorithm *assigns a weight* to each connection and that the routers implement a Weighted Fair Queuing (WFQ) [9, 13] scheme using these weights when forwarding the packets. In worst case terms this ensures that a connection will get at least the minimum, over all links that it uses, of $Bw_i / \sum_j w_j$, where B is link bandwidth, w_i is the weight for connection i, and the sum is over all the connections assigned to the link. We choose WFQ because it is widely accepted as a standard fair scheduling paradigm in the networking community and is supported (in one form or another) by most modern routers and switches.

It is important to note that our model is different from the model of Awerbuch *et al.*and Aspnes *et al.* [4, 3]; our model allows the assignment of variable weight or bandwidth to each request while the previous models assumed that requests had fixed, predetermined bandwidths. In particular, the logarithmic upper and lower bounds presented on throughput-competitiveness in [4] do not apply for our model. However, we present an $\Omega(\log n)$ lower bound on throughput-competitivity in our model (see Section 8.1). We also show an $O(\log n \log\log n)$ throughput-competitive upper bound (see Section 4).

Our main result is an online algorithm which achieves $O(\log^2 n \log U)$ coordinate-wise competitive ratio. For each coordinate, U is the reciprocal of the maximum value (over all solutions) of that coordinate. For the first coordinate in the bandwidth vector (for which U will be largest), U is equal to the minimum (over all solutions) of the maximum (over links) of the number of sessions routed along a link. We show that the dependence on U is necessary by presenting an $\Omega(\sqrt{\log U})$ lower bound on prefix-competitivity to global fairness.

In the case where bandwidth has to be assigned explicitly during the routing phase, we present an algorithm that achieves $O(\log^2 n \log^{1+\epsilon} U/\epsilon)$ coordinate-wise competitivity (for any chosen ϵ). We also show an $\Omega(\log^{1+\epsilon} U/\epsilon)$ lower bound in this model.

Related work Kleinberg *et al.* [11] addressed the problem of obtaining prefix optimality for scheduling and for single source unsplittable flow problem in an *offline setting*. For the single source unsplittable flow problem, they produce a solution which is a coordinate-wise two approximation of the globally fair allocation. In contrast, we study a more general problem in an *online setting*. We consider the multiple-source online case where the objective is coordinate-wise approximation of *any bandwidth allocation* (not just the fairest). The problem of fair bandwidth allocations for single source fractional flows was first studied by Megiddo [12].

The problem of online throughput maximization was considered in [4]. If one uses the algorithm in [3] where each connection is assumed to have unit bandwidth, it is possible to maximize the bandwidth allocated to the poorest session. It is interesting to note that, experimentally, the second approach achieves a very good approximation of global fairness. None of the above algorithms can be used to address the simultaneous approximation of fairness and throughput, considered in this paper.

Simulation Results Preliminary simulation results are presented in Section 7. Instead of implementing the proposed algorithm directly, we have implemented a modified version which attains the same competitive ratio up to a constant factor. As expected, the results are much better than worst case bounds and indicate that our algorithm is effective in achieving its goals: we get significantly larger total throughput without significantly deviating from optimum fairness.

2 Model

We are given a directed (or undirected) graph with n nodes and $m \leq n(n-1)$ edges. Requests arrive in an online fashion; each request specifies a source and sink node. When a request arrives, our algorithm must assign a *route* (path from source to sink) for the session, as well as a *weight* which will be used to determine the session bandwidth. All weights assigned by our algorithm will be between zero and one, although this requirement is not needed for our lower bounds. The final bandwidth given to each session will be determined according to *Weighted Fair Queueing* [9, 13]. Our analysis only requires that each session receive bandwidth at least equal to the minimum, over edges along its route, of the session weight divided by the total weight of all sessions using the edge.

We assume that sessions, once set up, remain in the network forever. We also assume that the various requests have equal priority, and that all edges have equal total bandwidth. None of these three assumptions are necessary for our algorithm to work; slight modifications will encompass the three scenarios described. We mention such modifications in Section 6.

Definition 1 *The* bandwidth vector *for an assignment of routes and bandwidths is composed of the assigned bandwidths in nondecreasing order.*

Definition 2 *The* globally fair solution *is the assignment of routes and bandwidths which produces the lexicographically largest bandwidth vector. There may be several globally fair solutions, each with an identical bandwidth vector.*

Observe that every globally fair solution is also max-min fair; moreover, if routes are preassigned, the max-min fair solution is unique and therefore globally fair.

Definition 3 *A solution is* γ_C coordinate-wise competitive against all solutions *if, for every i, the ith coordinate in the bandwidth vector is at least $1/\gamma_C$ times the maximum value, over all bandwidth vectors, of the ith coordinate.*

Definition 4 *A solution is* γ_P prefix competitive against global fairness *if the sum of the first i coordinates of the bandwidth vector is at least $1/\gamma_P$ times the sum of the first i coordinates in the globally fair optimal bandwidth vector, for every i.*

Our algorithm will guarantee $O((\log n)^2 \log U)$ coordinate-wise competitivity against every possible solution on the given graph and request sequence. We define U to be minimum over all solutions of the maximum number of requests routed along any link. Equivalently, U is also equal to the reciprocal of the minimum bandwidth assigned in the globally fair optimal solution.

Observe that any solution which is coordinate-wise competitive against all other solutions must also be prefix competitive against all other solutions. Since the total throughput of a solution is equal to the final prefix, it follows that our algorithm is also $O((\log n)^2 \log U)$ throughput competitive. Using the methods detailed in Section 4, we can prove an $O((\log n)^2 \log \log n)$ bound on our algorithm's throughput-competitivity.

3 The GMP Algorithm

The algorithm maintains a number of stages; virtual "copies" of the network. Each stage maintains loads on each edge, where λ_e^i is equal to the number of requests which stage i has routed along paths which include edge i.

Each stage can compute costs, $c_e^i = 2^{\lambda_e^i} - 1$, on the various edges. When a request comes in, the algorithm finds the minimum cost path from source to sink in each stage. The lowest-numbered stage which has a path of cost $n-1$ or less will route the request. If stage i is to route the request, then the new session will follow its minimum stage i cost path, and will be assigned weight $1/(i-1+\log n)$. The loads along this path in stage i will be increased by one; all other loads remain unchanged.

The algorithm used by each stage closely resembles the algorithm used in [4] running with capacity $\log n$.

3.1 Analysis of GMP

3.1.1 Single Stage Analysis

We wish to compare the number of requests routed at a single stage to the number of requests which could be routed using capacity u_e on each edge e. Suppose request sequence R reaches the current stage (these are the requests which no lower stage routed). We rewrite the cost function in terms of u_e as $c_e = n^{\lambda_e/Au_e} - 1$ for a constant A (note that $A = (\log n)/u_e$ yields our standard weight function). We route requests online, along their minimum cost route. If a request's minimum cost route has cost greater than $n-1$ then the request will be rejected by the current stage and passed on to the next; otherwise the request is accepted and we increment the loads (λ_e) along its path.

We compare the total number of requests we accepted (ρ_A) to the maximum possible number of requests which could be accepted offline, using capacity u_e on each edge (ρ^*). The following lemmas are modifications of lemmas appearing in [4]. They relate the value of the cost function to the number of sessions routed.

Lemma 3.1 *If $Au_e \geq \log n$, then $\sum_e Au_e c_e(L+1) \leq 2\rho_A \log n$*

Lemma 3.2 *$\rho_A \geq A\rho^*/(A + 2\log n)$ provided $Au_e \geq \log n$ for all edges e.*

Lemma 3.3 *The load on edge e is $\lambda_e \leq Au_e$.*

For our algorithm, every stage has $Au_e = \log n$. Treating every edge as having bandwidth b, we derive the following theorem which relates the number of requests accepted in a stage (ρ) with the maximum number of requests reaching that stage which could be routed using capacity b on each edge (ρ_b^*).

Theorem 3.1 *$\rho \geq \rho_b^*/(2b+1)$*

3.1.2 Proof of Competitivity

We now analyze the entire algorithm and prove our $O((\log n)^2 \log U)$ coordinate-wise competitivity against all other solutions. Define ρ_b^* to be the total number of requests which could be routed using capacity b. The number of requests which the optimal assigns bandwidth $1/b$ or more must clearly be smaller than ρ_b^* since all such high-bandwidth requests could be routed using capacity b. Define r_i to be the number of requests which the online algorithm routes in stages 1 through i.

We will first bound the number of requests (of those which could be routed with bandwidth b) remaining after the first i stages.

Lemma 3.4 *For any $b \geq 1$ and $i \geq 1$, $r_i \geq \rho_b^*(1 - (\frac{2b}{2b+1})^i)$.*

Proof: The proof will be by induction on i. For $i = 0$, the lemma is obvious since both sides are zero. After the first i stages, there are at least $\rho_b^* - r_i$ requests remaining which could be routed using capacity b. Theorem 3.1 guarantees that at least $1/(2b+1)$ of the remaining requests are routed in the next stage.

$$r_{i+1} \geq r_i + (\rho_b^* - r_i)/(2b+1) = r_i(\frac{2b}{2b+1}) + \frac{\rho_b^*}{2b+1}$$

Making use of the inductive hypothesis yields

$$r_{i+1} \geq \rho_b^*(\frac{2b}{2b+1} - (\frac{2b}{2b+1})^{i+1}) + \rho_b^*(\frac{1}{2b+1}) = \rho_b^*(1 - (\frac{2b}{2b+1})^{i+1})$$

■

We will now prove that the algorithm routes at least as many requests in the first $b + 2b\log m$ stages as any algorithm could routes using capacity b on each edge. It will follow that the first $b + 2b\log m$ stages capture at least as many requests as any solution routes with bandwidth $1/b$ or more.

Lemma 3.5 *For any $b \geq 1$, $r_{b+2b\log m} \geq \rho_b^*$.*

Proof: We know that $2^{1/2b} - 1 \leq 1/2b$. It follows that $(2b+1)/2b \geq 2^{1/2b}$ and reciprocating both sides yields $2b/(2b+1) \leq (1/2)^{1/2b}$. Plugging $i = 2b\log m$ into Lemma 3.4 and using this inequality yields

$$r_{2b\log m} \geq \rho_b^*(1 - (\frac{2b}{2b+1})^{2b\log m}) \geq \rho_b^*(1 - \frac{1}{m})$$

Since ρ_b^* requests can be routed using capacity b, it follows that $\rho_b^* \leq mb$, so after $2b\log m$ stages we have $r_{2b\log m} \geq \rho_b^* - b$. Since each stage routes at least one session before forwarding anything to the next stage, it follows that b stages later, we will have $r_{b+2b\log m} \geq \rho_b^*$ as desired. ■

When a request is routed in stage i, we assign a weight to the request of $1/(i - 1 + \log m)$. It follows that all the requests in the first $b + 2b\log m$ stages receive weight at least $1/(3b\log m)$. For any given b at least as many requests receive weight $1/(3b\log m)$ or more in the online as receive bandwidth $1/b$ or more in the optimal. We need to bound the total weight on an edge in order to relate weight to bandwidth in the online algorithm.

Lemma 3.6 *The total weight on any edge is at most $O((\log n)\log U)$ where U is the minimum over all solutions of the maximum number of requests routed along any edge.*

Proof: Since each stage uses capacity $\log n$, no stage places more than $\log n$ requests on an edge. There exists a solution which routes all the requests placing at most U on any edge; thus ρ_U^* is equal to the number of requests. After $U + 2U\log n$ stages, lemma 3.5 implies that the entire request sequence will be routed. So the total weight on edge e is bounded by:

$$w_e \leq \sum_{i=1}^{U+2U\log n} (\log n)\frac{1}{i - 1 + \log m} \leq (\log n)((\sum_{i=1}^{U+(2U+2)\log n} \frac{1}{i}) - (\sum_{i=1}^{\log m - 1} \frac{1}{i})) = O((\log n)\log U)$$

■

We can now prove our coordinate-wise competitive ratio.

Theorem 3.2 *The GMP algorithm attains $O((\log n)^2(\log U))$ coordinate-wise competitivity against all solutions.*

Proof: Define R to be the request sequence, with a total of $|R|$ requests.

Suppose the ith coordinate of a solution is $1/b$. It follows that this solution routed at least $|R| - i + 1$ requests using capacity b, and thus $\rho_b^* \geq |R| - i + 1$. Using lemma 3.5, we know $r_{b+2b\log m} \geq |R| - i + 1$. Thus our online algorithm routed at least $|R| - i + 1$ requests with weight $1/(b - 1 + 2b\log m) \geq 1/(3b\log m)$ or more. Thus at most $i - 1$ requests received weight less than $1/(3b\log m)$.

Using lemma 3.6, a session with weight w receives bandwidth at least $w/k(\log n \log U)$ (where k is a constant independent of U and n). It follows that at most $i-1$ requests received less than $1/(3bk(\log n)^2\log U)$ bandwidth. So coordinate i in our online solution is at least $1/(3bk(\log n)^2\log U)$, for a coordinate-wise competitive ratio of $\gamma_C = O((\log n)^2\log U)$. ■

4 Throughput-Only Competitivity

Previous algorithms for approximating throughput assumed that sessions request specific (known or unknown) bandwidth [4, 3]. We give an $O((\log n)\log\log n)$ approximation algorithm for throughput, in the model where weights are assigned to sessions as they arrive. The same algorithm achieves $O((\log n)\log\log n)$ competitivity if we must assign actual bandwidths up front (rather than weights). Our *GMP-THPT* algorithm will run only the first stage of the GMP algorithm. If the stage finds a path of cost $n-1$ or less, we assign weight 1. Otherwise we assign weight 0. Alternatively, we can assign bandwidth $1/\log n$ to the requests we give weight 1 and bandwidth 0 to the ones we give weight 0 (the analysis will remain the same). We will prove the existence of a near-optimal offline solution which assigns either zero or at least $1/\log n$ bandwidth to each request and places at most 1 total unit of bandwidth on any link. This offline solution's throughput is within constant factors of the best possible throughput using arbitrary bandwidths totaling at most one unit of bandwidth per link! In other words, using small nonzero bandwidths cannot help the throughput by much. We will then show that our algorithm approximates the near-optimal to within $O((\log n)(\log\log n))$, from which it follows that our algorithm approximates the true optimal throughput to within a constant of the same bound.

Theorem 4.1 *There exists an assignment of bandwidths and routes in which every request receives either zero or at least* $1/\log n$ *bandwidth, such that the total throughput is within a factor of* 32 *of the optimal throughput.*

Proof: Consider two possible assignments of routes and bandwidths. Assignment one gives bandwidth equal to the optimal bandwidth to every request which the optimal gave $1/\log n$ or more. All other requests receive bandwidth zero, and all requests use the same routes as in the optimal. Assignment one must be legal, since the optimal assignment is legal and all requests receive at most the same bandwidth along the same route as in the optimal. Assignment one attains throughput ρ_1.

Assignment two gives bandwidth equal to the optimal bandwidth to every request which the optimal gave less than $1/\log n$. All other requests receive bandwidth zero, and all requests use the same routes as in the optimal. Assignment two is legal, since each request receives at most the same bandwidth, along the same route, as in the optimal. Assignment two attains throughput ρ_2.

The optimal throughput is given by $\rho^* = \rho_1 + \rho_2$. One of the two assignments must therefore attain at least half the optimal throughput. If assignment one attains half the optimal throughput we are done, since assignment one gives each request either zero bandwidth or at least $1/\log n$ as required. Otherwise $\rho_2 \geq (1/2)\rho^*$.

We perform a randomized rounding on the second solution. Let $\alpha > 1$ be a number whose value we will specify later. Each request in this randomized assignment will receive bandwidth $1/\log n$ with probability $(1/\alpha)b\log n$ where b is the bandwidth assigned to the request in assignment two. Otherwise, the request gets bandwidth zero. If the total bandwidth along any edge is more than 1 we assign zero bandwidth to all connections which use that edge. We thus obtain a feasible solution. The randomized throughput is ρ_2^R. Let X_b be the bandwidth assigned to a connection in the rounded solution, where $b < 1/\log n$ is the bandwidth assigned to the connection by the second component of the optimal solution.

Lemma 4.1 $E[X_b] \geq \frac{b}{2\alpha}$.

Proof: $X_b = 0$ if either the connection did not get rounded up to $1/\log n$, or if it got rounded up but some edge along the path of this connection exceeded its capacity. The probability of the first event is exactly $1-(1/\alpha)b\log n$. Since each connection traverses at most n edges, the probability of the second event is at most $n\cdot Z\cdot(1/\alpha)b\log n$, where Z is the probability of any edge exceeding $1-1/\log n$. Assume for simplicity that $\log n$ is integral. Then Z is the probability that some edge either exceeds its capacity, or is packed to capacity. The expected number of connections on an edge is at most $\mu = \log n/\alpha$. Also, the edge can accommodate $\log n$ connections. Therefore, for an edge to exceed its capacity, the number of connections which are "ON" must be more than α times the mean. We now apply Chernoff bounds to obtain

$$Z \leq (\frac{e^{\alpha-1}}{\alpha^{\alpha}})^{\log n/\alpha}.$$

It is easy to see that $Z \leq 1/(2n)$ for $\alpha = 8$ and for large enough n. Therefore, $P[X_b = 0] \leq 1 - \frac{1}{2\alpha} b \log n$ i.e. $E[X_b] \geq b/(2\alpha)$. ■

Summing over all connections, we get $E[\rho_2^R] \geq \rho_2/(2\alpha)$. It follows that there exists a solution which (without exceeding total bandwidth one on any edge) has throughput at least $\rho^*/4\alpha$ while assigning every request bandwidth either zero or at least $1/\log n$. ■

Theorem 4.2 *Our algorithm obtains within $O((\log n)\log U)$ of the throughput of any other "optimal" solution, where U is the reciprocal of the minimum nonzero bandwidth assigned to any request by the optimal.*

Proof: Suppose our algorithm routes r requests and obtains throughput ρ. We define b_i^* to be the total number of requests which the optimal routed with bandwidth between $1/i$ and $1/(i+1)$. We define r_i^* to be the maximum number of requests which could be routed without exceeding $i+1$ requests per link.

We will now relate the number of requests which the optimal routes at various bandwidths to the maximum number of requests which can be routed without exceeding capacity $i+1$.

All the requests which the optimal assigns bandwidth $1/i$ or more could be routed without exceeding i requests per link; the optimal has to do this so as to not exceed one unit of throughput on any link. It follows that $r_i^* \geq \sum_{j=1}^{i} b_j^*$. From this we can derive the following equation:

$$\sum_{i=1}^{U-1}(\frac{1}{i} - \frac{1}{i+1})r_i^* \geq \sum_{j=1}^{U-1}\sum_{i=j}^{U-1}(\frac{1}{i} - \frac{1}{i+1})b_j^* = \sum_{j=1}^{U-1}(\frac{b_j^*}{j} - \frac{b_j^*}{U})$$

The total throughput of the optimal is at most $\rho^* = \sum_{j=1}^{U}(b_j^*/j)$. It follows that

$$\frac{r_U^*}{U} + \sum_{i=1}^{U-1}(\frac{1}{i} - \frac{1}{i+1})r_i^* \geq \rho^*$$

Theorem 3.1 indicates that $r \geq r_i^*/(2i+1)$ and therefore $r_i^* \leq r(2i+2)$. Using this in the equation above yields the following:

$\rho^* \leq \frac{(2U+2)r}{U} + \sum_{i=1}^{U-1} \frac{2r}{i} = 2r + \sum_{i=1}^{U} \frac{2r}{i} = O(r \log U)$

Since the throughput of the online is $\rho \geq (r/\log n)$, it follows that $O(\log n \log U)\rho \geq \rho^*$. ■

We have shown that there is a near-optimum (within constant factors) solution with minimum nonzero bandwidth $U \geq 1/\log n$. We have also shown that our algorithm approximates the throughput of any other solution to $O(\log n \log U)$. Combining these theorems will give the result we want.

Theorem 4.3 *Our algorithm obtains within $O((\log n)\log\log n)$ of the best possible throughput.*

5 Assigning Bandwidth Directly

We consider a model where our algorithm must assign routes and bandwidths to each request (rather than assigning routes and weights). We modify our approximation algorithm to obtain competitive ratio $O((\log n)^2(\log U)^{1+\epsilon}/\epsilon)$ in this model, where ϵ is a new parameter for the algorithm. We also present a lower bound of $\Omega((\log U)^{1+\epsilon}/\epsilon)$, which suggests that the increase in competitive ratio was a necessary one.

The algorithm is identical to the one given in section 3, except that requests routed in stage i are assigned bandwidth $\epsilon/((i-1+\log m)(\log n)(\log(i+1))^{1+\epsilon})$ instead of being assigned a weight.

Only the assignment of bandwidths has changed, so lemma 3.5 still holds. It follows that, for any given b, at least as many requests receive bandwidth $\epsilon/((3b\log m)(\log n)(\log(b+1))^{1+\epsilon})$ as received bandwidth $1/b$ in the optimal. Provided we can show that our algorithm never overflows any edges, it will follow that we are $O((\log n)^2(\log U)^{1+\epsilon}/\epsilon)$ competive, as desired.

Theorem 5.1 *The bandwidth-assigning algorithm doesn't overflow any edges.*

Proof: Stage i places at most $\log n$ sessions on any edge. Thus the total bandwidth placed on an edge by stage i cannot exceed $\epsilon/((i-1+\log m)(\log(i+1))^{1+\epsilon})$. The total number of stages will not exceed $U+2U\log m$, so the total bandwidth on any edge is at most $\sum_{i=1}^{U+2U\log m}\epsilon/((i+\log m)(\log(i+1))^{1+\epsilon}) \leq \epsilon\sum_{i=1}^{U+2U\log m} 1/i(\log(i+1))^{1+\epsilon}$ As U goes to infinity, the summation is bounded by $1/\epsilon$, a fact which can be seen by using the integral as an upper bound. It follows that the total bandwidth is at most 1 and our algorithm does not overflow any edges. ■

We present a lower bound suggesting that the competitive ratio when bandwidths are assigned must depend superlinearly on the logarithm of U.

Theorem 5.2 *Any deterministic online algorithm which assigns bandwidths directly has competitive ratio $\omega(\log U)$.*

Proof: Consider a network composed of a single edge. Suppose our algorithm achieves a competitive ratio of $\gamma(U)$, expressed as a function of U. Requests arrive online, each across the single edge. After i requests have arrived, the globally fair solution would give every request bandwidth $1/i$. Thus the first prefix is $1/i$, so our first prefix must be $1/i\gamma(i)$. It follows that every request received bandwidth at least $1/i\gamma(i)$. In particular, request i receives bandwidth $1/i\gamma(i)$ or more. The total bandwidth of all requests cannot exceed one. It follows that

$$\sum_{i=1}^{i=U}\frac{1}{i\gamma(i)} \leq 1$$

We can lower bound this sum with an integral, so it follows that the integral of $1/i\gamma(i)$ as i goes from 1 to infinity is at most 1. If $\gamma(i)=O(\log i)$, the integral would be infinite. It follows that $\gamma=\omega(\log U)$. On the other hand, $\gamma(i)=(\log i)^{1+\epsilon}/\epsilon$ will converge to one as U goes to infinity, for any $\epsilon>0$. ■

6 Further Extensions

The algorithm in [4] also applies when requests have variable profit and when sessions have given start and finish times (instead of remaining forever). It should be straightforward to extend our algorithm (by varying the profit and/or time-expanding the network) to achieve competitive ratio of $O((\log\mu)^2(\log U))$ where μ is the product of the number of nodes in the graph, maximum to minimum request profit ratio, and maximum time duration of a request (assuming all time durations are integer). In this case, U is the maximum number of active sessions along any edge at any time unit for the globally fair solution. The proofs for this follow directly from [4].

7 Simulation Results

In order to make our algorithm practical, we need to modify it slightly. Experimentally, the $\log n$ factors in the competitive ratio seem to disappear. This was not unexpected, since [10] showed that exponential algorithms attain very tight approximations for throughput when the traffic is Poisson in nature. However, we essentially sacrifice a $\log U$ factor on the least-bandwidth requests. We avoid this problem by assigning

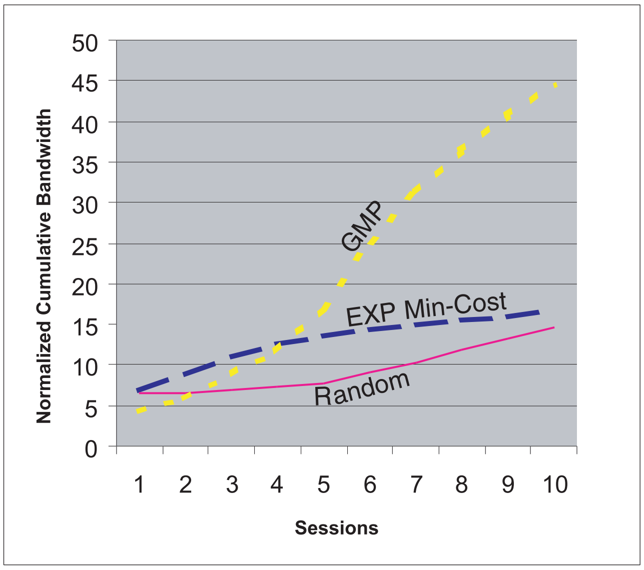

Figure 1: Comparing optimum fairness to combined fairness/throughput algorithm

half the bandwidth directly to requests and splitting the other half according to max-min fairness. Our algorithm will assign a route and bandwidth to each request; we divide all bandwidths by two to ensure at least half the bandwidth remains free. Free bandwidth is divided at the end in a max-min fair fashion.

We also modified our algorithm to reduce the number of stages. Instead of using $O(U \log n)$ stages each with capacity $\log n$, we can use $O(\log U \log n)$ stages and increase the capacity by a factor of two for each stage. Weights are proportional to the total capacity of the stage. This modification seems not to effect our performance from a practical standpoint (theoretically we can show that we lose at most a constant factor). However, the reduction in the number of stages speeds up the running time significantly.

Our sample graph is a pair of parallel lines, each sixty-four nodes long. Two kinds of requests arrive online, "long" requests which specify one of the two lines (75% the top line, 25% on the bottom line), and "short" requests which which connect two randomly chosen nodes and may be satisfied along either line. We ran five thousand requests, with 20% of them being long and 80% short.

Figure 7 shows a comparison of our modified algorithm to an algorithm which routes over a random shortest path and to a modified AAFPW algorithm [3]. Experimental results show that the latter algorithm achieves close to global fairness. The graph shows scaled prefix vector, i.e. we divide the i-th coordinate by i. Note that the leftmost point corresponds to the minimum bandwidth assigned to a connection, while the rightmost point corresponds to average bandwidth per connection, i.e. total throughput divided by the number of connections. Observe that our total throughput is significantly more than the other algorithms while our minimum assigned bandwidth doesn not suffer too much.

8 Lower Bounds

We present lower bounds for online approximation of the globally fair prefix. These two bounds suggest logarithmic dependence on both n, the number of nodes in the graph, and U, the maximum number of requests routed by the globally fair solution along any edge. Both bounds will use directed graphs.

8.1 Dependence on the number of nodes

The lower bound given in [4] does not apply directly to our weighted model. We present a new $\Omega(\log n)$ lower bound for deterministic online throughput approximation.

Theorem 8.1 *There is an $\Omega(\log n)$ lower bound on the competitive ratio of deterministic online throughput to globally fair throughput.*

Proof: The graph consists of a line of length n directed from node 1 towards node n with additional source and sink nodes. The first request goes either from node 1 to node $n/2$ or from node $(n/2)+1$ to node n. In other words, it uses either the left or right half of the line. We introduce a source node connected to node 1 and $(n/2)+1$ and a sink node which can be reached from nodes $n/2$ and n for this. The online uses either the left half of the line or the right half (or both). Supposing the online uses the left half, the next source will connect to nodes 1 and $(n/4)+1$ and the sink will be reachable from $n/4$ and $n/2$. Again the online algorithm uses the right or left half of the half of the line it's already used. Continuing this process, we end up with $\log n$ requests all of which the online overlapped on a single edge. The online throughput cannot exceed 1 with this edge as a bottleneck, regardless of the assignment of weights. By choosing the opposite half of the line at each step, the optimal can route all requests without any overlap and give them all throughput 1. We need to introduce enough source and sink nodes to allow this process regardless of the online choices. This means one source at stage one, two at stage two, four at stage three, and so on. The total number of sources will be at most $2n$ for a total of $5n = O(n)$ nodes. This completes the proof that the online throughput is $\Omega(\log n)$ less than the optimal.

■

Theorem 8.2 *There is an $\Omega(\log n)$ lower bound on the competitive ratio of deterministic online throughput to optimal throughput.*

8.2 Fairness Depends on the Maximum Requests per Link

We prove that any online algorithm which obtains prefix competitivity against the globally fair optimal better than $\Omega(n)$ will have competitive ratio dependent on the maximum number of requests which the optimal places along any edge (U). The proof of the following theorem is deferred to Appendix A.

Theorem 8.3 *The prefix-competitivity of a deterministic online algorithm against the globally fair solution is lower-bounded by $\Omega(min(n, \sqrt{\log U}))$.*

8.3 Unit Weight Lower Bounds

We consider the model where the algorithm can only assign routes; bandwidths are determined by max-min fairness. We prove that we can approximate neither optimal throughput nor global fairness to better than $O(n)$ in this model; this is the reason for our introduction of weights. We omit the proofs of Theorems 8.4 and 8.5 from this version of our paper.

Theorem 8.4 *As long as max-min fairness determines bandwidths, deterministic online throughput competitivity is bounded by $\Omega(n)$.*

Theorem 8.5 *There is an $\Omega(n)$ lower bound on prefix-competitivity for deterministic online algorithms against global fairness as long as max-min fairness determines the bandwidths.*

References

[1] Y. Afek, Y. Mansour, and Z. Ostfeld. Convergence complexity of optimistic rate based flow control algorithms. *Proceedings of the 28th ACM Symposium on Theory of Computing*, pages 89–98, 1996.

[2] Y. Afek, Y. Mansour, and Z. Ostfeld. Phantom: A simple and effective flow control scheme. *ACM SIGCOMM*, pages 169–182, 1996.

[3] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. *25th ACM Symposium on Theory of Computing*, pages 623–31, 1993.

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

[5] B. Awerbuch and Y. Shavitt. Converging to approximated max-min flow fairness in logarithmic time. *Proceedings of the 17th IEEE Infocom conference*, pages 1350–57, 1998.

[6] Y. Bartal, J. Byers, and D. Raz. Global optimization using local information with applications to flow control. *38th Annual Symposium on Foundations of Computer Science*, pages 303–312, 1997.

[7] Y. Bartal, M. Farach-Colton, M. Andrews, and L. Zhang. Fast fair and frugal bandwidth allocation in atm networks. *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 92–101, 1999.

[8] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, 1992.

[9] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Internetworking: Research and Experience*, 1(1):3–26, 1990.

[10] A. Kamath, O. Palmon, and S. Plotkin. Routing and admission control in general topology networks with poisson arrivals. *7th ACM-SIAM Symposium on Discrete Algorithms*, pages 269–278, 1996.

[11] J. Kleinberg, Y. Rabani, and E. Tardos. Fairness in routing and load balancing. *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, 1999.

[12] N. Megiddo. Optimal flows in networks with sources and sinks. *Math Programming*, 1974.

[13] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks - the single node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.

A Proof of Theorem 8.3

In this section we prove Theorem 8.3 from Section 8 in the main text:

Theorem 8.3: The prefix-competitivity of a deterministic online algorithm against the globally fair solution is lower-bounded by $\Omega(min(n, \sqrt{\log U}))$.

Proof: We will first construct the graph. The graph contains $n+1$ nodes labelled a_0 through a_n along a line, with edges from a_i to a_{i+1} for each i. In addition, we have another line of $n+1$ nodes labelled b_0 through b_n with edges from b_i to b_{i+1} for each i. The two lines intersect at $a_0 = b_0$. We have n additional source nodes s_0 through s_{n-1} with edges from s_i to a_i and from s_i to b_i. We have n additional sink nodes t_1 through t_n with edges from a_i to t_i and b_i to t_i. The total number of nodes is $4n+1$ and the total number of edges is $6n$.

The request sequence begins with n "short" requests. Short request i will be from s_{i-1} to t_i. The graph only permits two possible routings for short request i, either $(s_{i-1}, a_{i-1}, a_i, t_i)$ intersecting line a, or $(s_{i-1}, b_{i-1}, b_i, t_i)$ intersecting line b. We assume without loss of generality that the online algorithm routes at least half of these requests intersecting line a. Define κ to be the maximum weight assigned to any short request which the online routed intersecting line a.

The request sequence continues with U "long" requests. Each long request is from a_0 to a_n. The graph permits only a single possible routing for each long request. Suppose the online algorithm assigns weight w_j to the jth long request.

Lemma A.1 $w_j \geq \kappa/(j\gamma_P)$

Proof: After the first j long requests, the optimal will give the j long requests each bandwidth $1/j$ and the n short requests bandwidth 1 along the b-intersecting route. It follows that the first prefix for the optimal will be $p_1^* = (1/j)$. The first prefix for the online must therefore be $p_1 \geq (1/j\gamma_P)$. It follows that every online request receives bandwidth at least $(1/j\gamma_P)$.

There exists a short request which intersects line a and has weight κ. The edge with this request will be a bottleneck edge. Long request j receives bandwidth at most w_j divided by the total weight of the bottleneck edge, no more than w_j/κ. It follows that $w_j/\kappa \geq (1/j\gamma_P)$ and therefore $w_j \geq \kappa/(j\gamma_P)$ as desired. ∎

A total of $U+n$ requests have been made. Consider the prefix consisting of the $U+(n/2)$ requests receiving least bandwidth. The optimal assigns the long requests bandwidth $1/U$ and the short requests bandwidth 1 along the b-intersecting route. The $U+(n/2)$ prefix includes all the long requests and half the short requests, and the optimal obtains $p^*_{U+(n/2)} = (n/2)+1 > (n/2)$.

The online algorithm gives bandwidth 1 to the short requests which intersect line b, but there are at most $(n/2)$ of those and none of them will be included in the prefix. Prefix $U+(n/2)$ will be at most the total bandwidth of long requests plus the bandwidth of short requests which were routed intersecting line a. The long requests cannot exceed total bandwidth 1. Each short request gets weight at most κ. But the total weight on every edge of line a is at least $\sum w_j \geq (\kappa \log U)/\gamma_P$. So each short request gets bandwidth at most $(\gamma_P/\log U)$. Since there are at most n short requests intersecting line a, it follows that the online prefix is $p_{U+(n/2)} \leq 1 + (n\gamma_P/\log U)$.

In order to guarantee our prefix-competitive ratio of γ_P, we need to have $\gamma_P p_{U+(n/2)} \geq p^*_{U+(n/2)}$.

If $(n\gamma_P)/(\log U) \leq 1$, then it follows that the online prefix is at most 2 compared to the optimal prefix of $n/2$, and this implies that $\gamma_P \geq (n/4)$. Otherwise, the online profit is at most $(2n\gamma_P)/(\log U)$ from which it follows that $2n\gamma_P^2/\log U \geq (n/2)$. Solving this inequality for γ_P, it follows that $\gamma_P \geq (1/2)(\log U)^{1/2}$. Combining the two cases yields

$$\gamma_P \geq min(n/4, (1/2)\sqrt{\log U}) = \Omega(min(n, \sqrt{\log U}))$$

∎