Bandwidth Allocation in Networks: A Single Dual Update Subroutine for Multiple Objectives *

Sung-woo Cho †
University of Southern California

Ashish Goel ‡
Stanford University

May 31, 2005

Abstract

We study the bandwidth allocation problem

$$\text{Maximize } U(x), \text{ subject to}$$
$$Ax \leq c; x \geq 0$$

where U is a utility function, x is a bandwidth allocation vector, and $Ax \leq c$ represent the capacity constraints. We consider the class of *canonical* utility functions, consisting of functions U that are symmetric, non-decreasing, concave, and satisfy $U(0) = 0$. Natural primal-dual algorithms exist for maximizing such concave utility functions. We present a single dual update subroutine that results in a primal solution which is a logarithmic approximation, *simultaneously*, for all canonical utility functions. The dual update subroutine lends itself to an efficient distributed implementation.

We then employ the fractional packing framework to prove that at most $O(m \log m)$ iterations of the dual update subroutine are required; here m is the number of edges in the network.

1 Introduction

In this paper, we revisit the classic problem of distributed allocation of bandwidths to flows in a network (see [13, 3, 16, 1, 4, 2, 7] for some of the recent research

*A preliminary version of these results was presented at the workshop on combinatorial and algorithmic aspects of networking, 2004.

†Department of Computer Science, University of Southern California. Email: sungwooc@cs.usc.edu . Research supported by NSF Award CCR-0133968.

‡Departments of Management Science and Engineering and (by courtesy) Computer Science, Stanford University. Email: ashishg@stanford.edu. Research supported by NSF CAREER Award 0133968 and an Alfred P. Sloan Faculty Fellowship.

on the problem). We will assume that the route for each flow is given upfront. Formally, the problem is:

$$\text{Maximize } U(x), \text{ subject to}$$
$$Ax \leq c; x \geq 0.$$

Here, $x = \langle x_1, x_2, \ldots, x_n \rangle$ is an allocation of bandwidths to n flows, U is an arbitrary n-variate "utility function", and $Ax \leq c$ are linear capacity constraints (the matrix A encodes the routes for each flow). Let R denote the ratio of the largest capacity to the smallest capacity, and m denote the number of links in the network. We present an efficient dual update subroutine for this problem. Repeated invocations of this dual update subroutine result in a primal solution that simultaneously approximates the optimum solution for a large class of utility functions U. More specifically, our algorithm guarantees an $O(\log n + \log R)$ approximation simultaneously for all utility functions U that are symmetric, concave, non-decreasing, and satisfy $U(0) = 0$. Before presenting more details about our results and techniques, we review the related work and the motivation for our problem.

Motivation and related work: The Transport Control Protocol (TCP) is by far the most widely used solution to this problem is practice. In more abstract settings, Kelly, Maulloo, and Tan [13] proposed a distributed algorithm for this problem for the case where $U(x) = \sum_i U_i(x_i)$, and each U_i is a concave function. Their algorithm uses a primal-dual framework where the dual prices (which they call shadow prices) are maintained by "edge-agents" and primal flows are maintained by the "flow-agents". All communication is local, i.e., takes place between a flow-agent and an edge-agent on the path of the flow. The resulting solution is proportional with respect to the dual prices, and hence, their framework is widely referred to as the "proportional-fairness" framework. Subsequent work by Low, Peterson, and Wang [16] and others has shown that several variants of TCP essentially perform the above computation for different choices of utility functions. Since the behavior of different variants of TCP is quite different (different variants work better in different settings), the above work raises the following natural question: *Is it possible to obtain solutions which are simultaneously good for a wide range of utility functions?*.

Bartal, Byers, and Raz [3] presented a distributed algorithm for the above problem when $U(x) = \sum_i x_i$. Unlike the work of Kelly *et al.*, this work presents a running time analysis. They prove that a simple local computation along with local communication can lead to almost optimum solutions in polynomial time. Their work builds on the positive linear programming framework of Luby and Nisan [17]; for their problem, the positive linear programming framework is essentially iden-

tical to the fractional packing framework developed by Plotkin, Shmoys, and Tardos [19], and later simplified by Garg and Konemann [6]. Each edge-agent maintains dual costs, and each flow-agent uses these dual costs to update its own flow. Recently, Garg and Young [7] have shown how a simple MIMD (multiplicative increase multiplicative decrease) protocol can approximate the above objective. These results lead to the following natural question: *Can we obtain distributed algorithms with similar rigorous running time analyses for more involved utility functions?*.

Building on a series of papers about multi-objective fairness [14, 10], Kleinberg and Kumar [15] studied the problem of bandwidth allocation in a centralized setting with multiple fairness objectives. Goel and Meyerson [8] and Bhargava, Goel, and Meyerson [5] later expanded this work to a large class of linear programs and related it to simultaneous optimization [9]. In particular, they considered the class of utility functions U which are symmetric, concave, non-decreasing, and satisfy $U(0) = 0$. They presented an algorithm for computing a single allocation which is simultaneously an $O(\log n + \log R)$ approximation for all such utility functions. Their work builds on the notion of majorization due to Hardy, Littlewood, and Polya [11, 12, 18]; to obtain efficient solutions they use the fractional packing framework of Plotkin, Shmoys, and Tardos [19]. This leaves open the following question: *Can there be efficient distributed algorithms which achieve the same results?*

All three questions raised above are very similar, even though they arise in different contexts. They point towards a need for distributed bandwidth allocation algorithms which are good across multiple objectives and have provably good running times. We address precisely this problem. It is our hope that this line of research would ultimately lead to protocols which are comparable to TCP in simplicity (our algorithm is not) but perform well across multiple objectives.

The class of utility functions we consider is not arbitrarily chosen. This is a large class, and contains the important subclass $\sum_i f(x_i)$ where f is a uni-variate concave function (f must also be non-decreasing and $f(0)$ must be 0). Concavity is a natural restriction, since it corresponds to the "law of diminishing returns" from economics. Symmetry corresponds to saying that all users are equally important[1]. The requirements $U(0) = 0$ and U non-decreasing are natural in the setting of the bandwidth allocation problem. This class includes important special functions such as $\min$, $\sum_i \log(1 + x_i)$, and $\sum_i -x_i \log x_i$ (the entropy). It also contains a series of functions which together imply max-min fairness. Most interestingly, there is a concrete connection between this class and our intuitive notion

[1]Notice that the constraints are not required to be symmetric, and hence, the optimum solution need not be symmetric even though the objective function is symmetric.

of fairness. Suppose there exists some function which measures the fairness of an allocation. It seems natural that the allocation (x_1, x_2) should be deemed as fair as (x_2, x_1) and less fair than $(\frac{x_1+x_2}{2}, \frac{x_1+x_2}{2})$. This assumption implies that for any natural definition of fairness, maximizing fairness should be equivalent to maximizing some symmetric concave function; certainly, all the definitions of fairness that we found in literature are of this form.

We will use the term *canonical* utility functions to refer to this class[2].

Our result and a summary of our techniques: We present an algorithm that simultaneously approximates all canonical utility functions U up to a factor $O(\log n + \log R)$. Our algorithm performs $O(m \log m)$ iterations of a ***single*** dual subroutine, i.e., the subroutine does not depend on U. Here m is the number of edges in the network. Each invocation of the subroutine requires $O(\log n)$ steps. During each invocation, a flow-agent sends $O(1)$ messages of length $O(\log n + \log R)$. Also, during each invocation, each flow-agent exchanges $O(1)$ messages with each edge-agent on its route; this exchange between flow-agents and edge-agents is also used in the work of Kelly *et al.* [13] and Bartal *et al.* [3] and can be implemented efficiently by piggybacking information over data packets belonging to a flow.

To obtain the above results, we first describe (section 2) the centralized algorithm due to Goel and Meyerson [8]. This algorithm requires a solution to n fractional packing problems. Each packing problem requires a different dual optimization subroutine. A distributed implementation of this algorithm would require each edge-agent to maintain n different dual costs. Also, it is not clear how the dual subroutine can be implemented in a distributed fashion even for a single packing problem. To address these problems. we present (section 3) a single dual subroutine that approximates all the n different dual subroutines used in the centralized version. In this dual subroutine, the i-th flow-agent updates its cost by $1/C'_i$ where C'_i denotes the sum of the dual costs of all the flow-agents which are no more expensive than flow i. Because of its simplicity, this new dual subroutine is amenable to a distributed implementation. Since the same dual subroutine is used for all the fractional packing problems, the dual costs for the different packing problems grow identically, eliminating the need for maintaining multiple dual costs. Finally, we show how this dual subroutine can be used in conjunction with the fractional packing algorithm of Garg and Konemann [6] to obtain our result. Even though we use the algorithm of Garg and Konemann, we need to make several non-trivial changes

[2]The symmetry requirement implies that our class does not contain all utility functions to which the proportional-fairness framework applies. However, since our class does not require the utility function to be a sum of uni-variate functions, it contains important functions such as $\min$ which can not be addressed using the proportional-fairness framework. Hence the two classes of utility functions are incomparable.

to their analysis; details are presented in section 4. We prove that $O(m \log m)$ iterations suffice.

While we present our algorithm in the setting in which flows needs to be computed from scratch, it can easily be adapted to the setting where we need to improve existing flows incrementally. Even in a centralized setting, our algorithm would be considerably more efficient than earlier solutions which involve solving n fractional packings. Further, the guarantee on the approximation ratio for our algorithm matches (up to constant factors) the best known guarantee for simultaneous optimization even in a centralized setting (see theorem 2) and is in the same ballpark as a lower bound of $O(\log n / \log \log n)$ due to Kleinberg and Kumar [15] for this problem[3].

Future directions: To completely answer the questions posed earlier in the introduction, we would need to have a completely distributed and efficient algorithm for simultaneous approximation of all canonical utility functions. We believe that the dual update procedure and the analysis outlined in this paper is a good step in that direction. However, we still need to address several issues before we can obtain simple and practical protocols that perform simultaneous optimization:

1. Our algorithm, while quite efficient, is not "local"; it requires a small amount of communication between different flow-agents. Consequently, it seems hard to design a simple TCP-like protocol which mimics our distributed algorithm. However, the simplicity of our dual solution makes us optimistic that a simple protocol which only exchanges local messages can be used to replace our distributed algorithm for computing the dual subroutine. Such a protocol would be a significant development, both theoretically as well as in terms of its practical impact.

2. Our analysis requires $m \log m$ iterations (ignoring constant factors) of the dual update procedure. Also, we can construct simple examples where any flow improvement algorithm using our dual update subroutine will take at least $\tilde{\Omega}(n)$ steps, where the $\tilde{\Omega}$ notation hides polylogarithmic factors. To be considered practical, the number of iterations must be sub-linear.

[3] Kleinberg and Kumar studied the case where $R = 1$.

2 Background: Simultaneous optimization in a centralized setting

In this section we describe the framework for simultaneous optimization of linear programs developed by Bhargava, Goel, and Meyerson and by Goel and Meyerson [5, 8]. This framework works in a centralized setting. We also point out the difficulties in making it distributed.

Suppose we are given a linear set of constraints $\{Ax \leq c, x \geq 0\}$ where $x = \langle x_1, x_2, \ldots, x_n \rangle$ is an n-dimensional vector. Recall that we call a utility function U *canonical* if U is symmetric and concave in its arguments, $U(0) = 0$, and U is non-decreasing in each argument. Define the k-th prefix, $P_k(x)$ to be the sum of the k smallest components of x (not $x_1 + x_2 + \ldots + x_k$ but $x_{\sigma(1)} + x_{\sigma(1)} + \ldots x_{\sigma(k)}$ where σ is the permutation that sorts x in increasing order.). Let P_k^* denote the maximum possible value of $P_k(x)$ subject to the given constraints

Definition 1 Approximate Majorization: *A feasible solution x to the above problem is said to be α-majorized if $P_k(x) \geq P_k(y)/\alpha$ for all $1 \leq k \leq n$ and all feasible solutions y.*

Informally, the k poorest users in an α-majorized solution get at least $1/\alpha$ times as much resource as the k poorest individuals in any other feasible allocation. Intuitively, this seems to have some ties to fairness. The following theorem [8] makes this intuition concrete:

Theorem 1 *A feasible solution x is α-majorized if and only if $U(x) \geq U(y)/\alpha$ for all feasible solutions y and all canonical utility functions U.*

In fact, the above theorem holds for an arbitrary set of constraints (integer, convex etc.) as long as they imply $x \geq 0$. Thus the notion of α-majorization captures simultaneous optimization. For this framework to be useful, we need to demonstrate that α-majorized solutions exist with small values of α; the following theorem does exactly that [8].

Theorem 2 *If the set of feasible solutions is convex and non-negative, then there exists an $O(\log \frac{P_n^*}{nP_1^*})$-majorized solution.*

For many problems of interest, the above theorem translates into $\alpha = O(\log n)$. For example, for the bandwidth allocation problem with unit capacities, $P_n^* \leq n$ whereas $P_1^* \geq 1/n$, implying the existence of an $O(\log n)$-majorized solution. For non-uniform capacities, the guarantee becomes $O(\log n + \log R)$ where R is the ratio of the maximum to the minimum capacity. However, even if there

exists an α-majorized solution, it is not clear a priori that finding such a solution is computationally tractable. The next theorem [8] resolves this issue assuming linear constraints. Here, α^* is the smallest possible value of α for which an α-majorized solution exists.

Theorem 3 *Given the constraints $\{Ax \leq c, x \geq 0\}$, both α^* and an α^*-majorized solution can be found in time polynomial in n and the number of constraints.*

Similar techniques were developed by Tamir [20] to compute majorized elements. Goel and Meyerson [8] also demonstrate how the above framework can be applied to several integer programming problems using suitable rounding schemes.

Let us focus on a proof of theorem 3 that highlights the difficulties involved in making the above framework carry over in a distributed setting. We will restrict ourself to the bandwidth allocation problem for simplicity, where A and b are both required to be non-negative. In order to compute α^*, we first need to compute P_k^*. Computing P_k^* is equivalent to solving the following linear program:

$$\begin{gathered}\text{Minimize } \lambda_k \text{ subject to:} \\ Ax \leq \lambda_k c \\ \textstyle\sum_{i \in S} x_i \geq 1 \text{ for all } S \subseteq \{1, \cdots, n\} \text{ with } |S| = k\end{gathered}$$

P_k^* would be $1/\lambda_k^*$ where λ_k^* is the solution to the above linear program. The linear program described above is a fractional packing problem, and can be solved efficiently using the framework of Plotkin, Shmoys, and Tardos [19] or the simplification thereof by Garg and Konemann [6]. In order to solve the fractional packing problem efficiently, we need a *dual optimization subroutine* to solve the following program:

$$\begin{gathered}\beta_k(l) = \text{Maximize } P_k(x) \text{ subject to:} \\ Cx = 1; x \geq 0\end{gathered}$$

where $l(e) \geq 0$ is the "dual cost" of edge e and $C_i \geq 0$ represents the dual cost for flow i. The dual cost C_i for flow i is computed by simply adding the dual costs $l(e)$ of each edge e used by the flow. It is important to note that the dual costs are artifacts of the solution methodology and do not correspond to any real entity in the original problem.

Lemma 4 *[8] For any k, there exists an optimum solution x to the dual problem such that*

1. *$C_i \leq C_j \Rightarrow x_i \geq x_j$,*
2. *The solution x is two-valued. In particular, there exists a value γ such that for all i, $x_i = 0$ or γ.*

We will use the term "candidate dual solutions" to refer to feasible solutions which satisfy properties 1 and 2 in the above lemma. There are only n candidate dual solutions, and they can be enumerated in time $O(n \log n)$. This leads to an efficient dual optimization subroutine, and hence, an efficient solution to the fractional packing problem (the fractional packing problem can be solved by making polynomially many invocations of the dual subroutine). In order to find P_k^* for all $k, 1 \leq k \leq n$, we need to solve n fractional packing problems. The quantity α^* and the corresponding α^*-majorized solution can then be computed using similar techniques.

The above discussion pertained to centralized solutions. As mentioned before, a distributed solution to the bandwidth allocation problem would be quite desirable. The following issues need to be addressed in order to make the above framework distributed:

1. We need to find a distributed solution to the dual problem. Enumerating all the candidate dual solutions is efficient in a centralized setting. But in a distributed setting, this would require communicating all the dual costs to all the users.

2. In the centralized framework, we need to solve n fractional packing problems, one for each P_k^*. The dual costs for each problem need not be the same as the algorithm for each problem progresses. Hence, the network would need to keep track of n different dual costs per edge.

3. The centralized solution first computes all P_k^* and then computes α^*. it would be desirable for a distributed algorithm to directly obtain a small α.

We address the above issues in the next section by giving a simple dual subroutine that simultaneously approximates the dual problems for each P_k^* and can be made to work efficiently in a distributed setting. Since the same dual solution is used for each of the n packing problems, the dual costs for each problem remain the same as the algorithm progresses. In section 3 we show how the framework of Garg and Konemann can be adapted to our setting where multiple primal problems are being approximated using a single dual subroutine.

3 Distributed simultaneous optimization of the dual problems

In this section we will assume that the users are re-numbered in increasing order of dual costs i.e. $C_1 \leq C_2 \leq \ldots \leq C_n$. We first use lemma 4 to describe the j-th

candidate dual solution, $x^{(j)}$:

$$x_1^{(j)} = x_2^{(j)} = \ldots = x_j^{(j)} = \frac{1}{\sum_{i=1}^{j} C_i}, \text{and } x_i^{(j)} = 0 \text{ for } i > j.$$

Let C'_j denote the quantity $\sum_{i=1}^{j} C_i$. Let $\overline{x}$ denote the upper envelope of all the candidate dual solutions, i.e., $\overline{x}_j = 1/C'_j$. This upper envelope solution has several desirable properties. First, it dominates all the candidate dual solutions and hence, $P_k(\overline{x}) \geq \beta_k(l)$ for all $1 \leq k \leq n$. Further, the cost of $\overline{x}$ is not too much:

Lemma 5 $C\overline{x} \leq 1 + \ln n + \ln \frac{C_n}{C_1}$.

Proof:

$$\begin{aligned} C \cdot \overline{x} &\leq \sum_{j=1}^{n} \frac{C_j}{C'_j} \\ &\leq 1 + \int_{C_1}^{C'_n} (1/x)dx = 1 + \ln \frac{C'_n}{C_1} \\ &\leq 1 + \ln \frac{nC_n}{C_1} = 1 + \ln n + \ln \frac{C_n}{C_1} \end{aligned}$$

■

Recall that the dual cost for each flow is just the sum of the dual cost for all the edges used by the flow. Hence, each flow-agent can obtain an estimate of its own cost by merely sending a control packet which sums up the cost of all the edges on the route for the flow. In order to compute C'_i, user i needs to know the sum of the costs of all the "cheaper" flows. This can be accomplished using a logical balanced tree T on all the flow-agents. Each node on the tree contains an array $A[1..n]$. Starting from the leaves, each node x_i records its cost in $A[i]$ and passes the array to its parent. The parent node x_j now merges the arrays which it received from its children and overwrites its cost on $A[j]$. Then it again passes the array to its parent. After $\log n$ recursive steps, the cost information of all the nodes is delivered to the root of the tree. Once the root collects the information, it distributes the array down the tree. The total number of steps needed is $2 \log n$ and each node sends at most three message. Unfortunately, the size of each message is quite large. We solve this problem by reducing the size of the array to $O(\log \frac{C_n}{C_1})$. The basic idea is to group the costs into logarithmic number of sets. This can be done by rounding C_i down to powers of 2. Let W_i be the rounded cost of agent i, i.e.

$$W_i = 2^{\lfloor \log C_i \rfloor}.$$

When agent i receives arrays from its children, it merges them, increments the entry of array $A[\log \frac{W_i}{W_1}]$ by one, and then passes the array to its parents. The rest of the

procedure is same as before. Rounding allows us to keep only $O(\log \frac{C_n}{C_1})$ number of groups, thus reducing the message size while introducing only a constant factor increase in the cost of the dual solution.

The following theorem summarizes the results of this section:

Theorem 6 *Given dual costs C, our distributed algorithm finds a dual solution x such that $P_k(x) \geq \beta_k(l)$ and $Cx = O(\log n + \log \max\{C_i\}/\min\{C_i\})$. The algorithm uses $O(\log n)$ message passing steps. Each flow-agent sends at most 3 messages of size $O(\log \max\{C_i\}/\min\{C_i\})$ each and does $O(\log \max\{C_i\}/\min\{C_i\})$ amount of computation.*

Armed with this distributed algorithm for simultaneous approximation of the dual problems, we now show how the fractional packing algorithm of Garg and Konemann can be adapted to obtain an $O(\log n)$-majorized solution for the bandwidth allocation problem.

4 Simultaneous optimization for the primal problems

We will now show how to use our dual subroutine from the previous section to achieve simultaneous optimization for the bandwidth allocation problems. While we will follow the structure and the notation of the fractional packing algorithm of Garg and Konemann [6], we can not use their result in a black-box fashion for several reasons. Since we are essentially solving n primal problems at the same time, we need to make sure that the same stopping condition can be used for each problem. Also, the approximation ratio for the dual problem depends on the ratio $\max\{C_i\}/\min\{C_i\}$. But these costs are dual costs, and we need to be careful they do not grow too large. The proof of lemma 8 uses the concavity of P_k. The proof of lemma 7 also depends crucially on properties of our dual solutions.

Recall that $c(e)$ is the capacity and $l(e)$ is the dual cost of edge e. Also recall that the dual cost $C_i(l)$ for flow i is given by $\sum_{e \in R_i} l(e)$ where R_i is the route of flow i. Let γ denote an upper bound on the cost of the dual solution obtained in the previous section.

For technical reasons, we will assume that the dual obtained in the previous section is multiplied by a scale factor s such that

$$\begin{array}{ll} \exists e \in E & \sum_{j:e \in r_j} s\overline{x}_j = c(e) \\ \forall e \in E & \sum_{j:e \in r_j} s\overline{x}_j \leq c(e). \end{array}$$

We will use the notation $x(l)$ for the final scaled dual solution, given edge costs l. The scale factor s can be obtained using a balanced tree; the details are similar to those in section 3 and are omitted.

Recall that $\beta_k(l)$ is the maximum value of $P_k(x)$ subject to the constraints $\{C(l)x = 1; x \geq 0\}$. After scaling, it is no longer necessarily true that $P_k(x(l)) \geq \beta_k(l)$ or that $C(l) \cdot x(l) \leq \gamma$. However, it is still true that, for all k,

$$C(l) \cdot x(l) \leq \frac{\gamma P_k(x(l))}{\beta_k(l)} \tag{1}$$

This is going to be sufficient for our purposes.

Let $D(l)$ denote the quantity $\sum_e l(e)c(e)$. We now state the algorithm for simultaneous primal optimization:

APPROX-FLOWS(ϵ, γ)

```
1  for each edge e
2      do l_0(e) = δ/c(e)    // δ will be defined later.
3  i = 0
4  x = 0
5  while D(l_i) < 1
6      do Flow phase: Assume we are given a length function l_i.
7          Let x(l_i) be the scaled γ-majorized solution for the dual problems.
8          x = x + x(l_i)
9          Edge phase:
10         for each edge e
11             do f_i(e) = Σ_{j:e∈r_j} x_j(l_i)
12                l_{i+1}(e) = l_i(e)(1 + ε f_i(e)/c(e))
13         i = i + 1
14 return x
```

We have already discussed how a scaled γ-majorized solution can be obtained in a distributed fashion. The stopping condition $D(l_i) < 1)$ can also be checked easily using the same technique. Hence the above algorithm is amenable to an efficient distributed implementation.

We will now discuss the feasibility, the number of iterations, and the approximation ratio (i.e. the α) of the above algorithm. We will set $\epsilon = 1/2$ and $\delta = m^{-\frac{1}{\epsilon}}$. The proof for feasibility and time complexity mirrors that of Garg and Konemann; all the details which are particular to our problem manifest themselves in the analysis of the approximation ratio.

Feasibility of the Primal: Note that the solution returned by the above algorithm may be infeasible since there is always at least one saturated edge in any iteration. To make our flows feasible, we use the same trick as used by Garg and Konemann. Consider the increase of each edge's length. Since every new flow on edge e does not exceed its capacity, for every $c(e)$ units of flow routed through edge e, its

length is increased by at least $1+\epsilon$. Therefore for every edge e, $c(e)$ is overflowed by the factor of $\log_{1+\epsilon} \frac{l_{T-1}(e)}{l_0(e)}$ at most where T is the time when the algorithm terminates. Since $D(l_{T-1}) < 1$, $l_{T-1}(e) < 1/c(e)$. Therefore, flows divided by $\log_{1+\epsilon} \frac{1/c(e)}{\delta/c(e)} = \log_{1+\epsilon} \frac{1}{\delta}$ give us a feasible solution. In practice, we would multiply the flows obtained during each iteration by this feasibility factor, rather than doing it once at the end.

Running time: Define the running time for computing the majorized dual solution as T_{sub}. Since our algorithm runs iteratively, and each iteration takes $O(T_{sub})$ time, the running time would be $O(T \cdot T_{sub})$ where T is the number of iterations to finish our algorithm.

In one iteration, we increase the length of a saturated edge by a factor of $(1+\epsilon)$. Therefore, the number of iterations in which we increase the length of an edge e is at most $\left\lceil \log_{1+\epsilon} \frac{l_T(e)}{l_0(e)} \right\rceil$. Since the algorithm stops when $\sum_e l(e)c(e) > 1$, we have $l_T(e) \leq 1/c(e)$, which gives

$$\begin{aligned} \log_{1+\epsilon} \frac{l_T(e)}{l_0(e)} &\leq \log_{1+\epsilon} \frac{1}{\delta} \\ &= \frac{1}{\epsilon} \log_{1+\epsilon} m. \end{aligned}$$

The above equation represents the number of iterations related to a particular edge. Multiplying by the total number of edges, m, yields

$$T \leq \left\lceil \frac{m}{\epsilon} \log_{1+\epsilon} m \right\rceil.$$

Since we have set ϵ to be $1/2$,

$$T = O(m \log m).$$

Approximation guarantee: For brevity, let $D(i)$, $C(i)$, $\beta_k(i)$ and $x(i)$ denote $D(l_i)$, $C(l_i)$, $\beta_k(l_i)$ and $x(l_i)$ respectively.

Lemma 7 *Let P_k^* denote the optimum solution of the primal problem. When the algorithm terminates at time T,*

$$\frac{\sum_{j=1}^{T} P_k(x(j-1))}{P_k^*} \geq \frac{\ln \frac{1}{m\delta}}{\epsilon\gamma}.$$

Proof: For $i \geq 1$

$$\begin{aligned} D(i) &= \sum_e l_i(e)c(e) \\ &= \sum_e l_{i-1}(e)\left(1+\epsilon\sum_{j:e\in r_j} x_j(i-1)/c(e)\right)c(e) \\ &= \sum_e l_{i-1}(e)c(e) + \epsilon\sum_e\left(l_{i-1}(e)\sum_{j:e\in r_j} x_j(i-1)\right) \\ &= D(i-1) + \epsilon\sum_j\left(\sum_{e\in r_j} l_{i-1}(e)x_j(i-1)\right) \\ &= D(i-1) + \epsilon\sum_j C_j(i-1)x_j(i-1) \\ &\leq D(i-1) + \frac{\epsilon\gamma P_k(x(i-1))}{\beta_k(i-1)} \qquad \text{(from equation 1)} \end{aligned}$$

Applying the duality theorem to the linear program for computing P_k^* yields

$$P_k^* \leq D(l)\beta_k(l).$$

The above step constitutes an important part of our proof. Notice that the duality theorem gives a different relation for each P_k. Informally. the guarantee obtained from equation 1 will exactly balances out this difference, resulting in the same guarantee for all dual problems. Now,

$$\begin{aligned} D(i) &\leq D(i-1) + \epsilon\gamma P_k(x(i-1))\frac{D(i-1)}{P_k^*} \\ &= D(i-1)\left(1+\epsilon\gamma\frac{P_k(x(i-1))}{P_k^*}\right). \end{aligned}$$

Hence,

$$\begin{aligned} D(i) &\leq \prod_{j=1}^{i}\left(1+\epsilon\gamma\frac{P_k(x(j-1))}{P_k^*}\right)D(0) \\ &\leq \prod_{j=1}^{i}\left(e^{\epsilon\gamma\frac{P_k(x(j-1))}{P_k^*}}\right)m\delta \\ &= e^{\epsilon\gamma\sum_{j=1}^{i}\frac{P_k(x(j-1))}{P_k^*}}m\delta. \end{aligned}$$

Suppose our procedure stops at iteration T for which $D(T) \geq 1$. Then,

$$1 \leq D(T) \leq e^{\epsilon\gamma\sum_{j=1}^{T}\frac{P_k(x(j-1))}{P_k^*}}m\delta.$$

This implies

$$\frac{\sum_{j=1}^{T} P_k(x(j-1))}{P_k^*} \geq \frac{\ln\frac{1}{m\delta}}{\epsilon\gamma}.$$

■

We need to specify δ so that we get the approximation ratio to be appropriately bounded. The following theorem shows that our algorithm has a good bound on approximation ratio for an appropriate δ.

Lemma 8 *For $\delta = m^{-\frac{1}{\epsilon}}$, our algorithm generates at most $\frac{\gamma}{(1-\epsilon)^2}$-approximate solution.*

Proof: Define $S = \log_{1+\epsilon} 1/\delta$ which is the feasibility factor that every $x(i)$ should be divided by. In addition, define ρ to be $\frac{P_k^*}{P_k\left(\sum_{j=1}^{T} x(j-1)\right)/S}$ which is the approximation ratio. Since P_k is a concave function and $P_k(0) = 0$, we have

$$P_k\left(\sum_{j=1}^{T} x(j-1)\right) \geq \sum_{j=1}^{T} P_k(x(j-1)).$$

Therefore,

$$\begin{aligned}
\rho &\leq \frac{P_k^*}{\sum_{j=1}^{T} P_k(x(j-1))} \cdot S \\
&\leq \frac{\epsilon\gamma}{\ln \frac{1}{m\delta}} \cdot S \qquad \text{(from lemma 7)} \\
&= \frac{\epsilon\gamma \log_{1+\epsilon} \frac{1}{\delta}}{\ln \frac{1}{m\delta}} \\
&= \gamma \frac{\epsilon}{\ln(1+\epsilon)} \frac{\ln \delta}{\ln(m\delta)}.
\end{aligned}$$

Since $\ln(1+\epsilon) \geq \epsilon(1-\frac{\epsilon}{2})$ and $\frac{\ln \delta}{\ln(m\delta)} = \frac{1}{1-\epsilon}$ for $\delta = m^{-\frac{1}{\epsilon}}$,

$$\begin{aligned}
\rho &\leq \gamma \frac{1}{(1-\frac{\epsilon}{2})(1-\epsilon)} \\
&< \gamma \frac{1}{(1-\epsilon)^2}.
\end{aligned}$$

■

We set ϵ to be $1/2$. Then our desired approximation ratio would be 4γ. Recall that our bound on γ from theorem 6 is $O(\log n + \log \max\{C_i\}/\min\{C_i\})$. The quantities C_i are dual costs – we now relate them to primal capacities.

Lemma 9 *Define R to be $\frac{\max_{e \in E} c(e)}{\min_{e \in E} c(e)}$. Then,* $\log \max\{C_i\}/\min\{C_i\} = O(\log m + \log R)$.

Proof: Since the length of each edge is monotonically increasing with time, and C_i is just the sum of lengths,

$$\min_e l_0(e) \leq C_i \leq \sum_e l_{T-1}(e)$$

where T is the time when our algorithm terminates. Hence,

$$\max\{C_i\}/\min\{C_i\} \leq \frac{\sum_e l_{T-1}(e)}{\min_e l_0(e)}.$$

Since $l_0(e) = \delta/c(e)$, $\min_e l_0(e)$ is $\frac{\delta}{\max_e c(e)}$. On the other hand, note that our algorithm terminates when $D(l_T) > 1$, and hence $D(l_{T-1}) \leq 1$, which implies that $\sum_e l_{T-1}(e)c(e) \leq 1$. Thus, $\sum_e l_{T-1}(e) \leq \frac{1}{\min_e c(e)}$. Therefore,

$$\begin{aligned} \max\{C_i\}/\min\{C_i\} &\leq \frac{\frac{1}{\min_e c(e)}}{\frac{\delta}{\max_e c(e)}} \\ &= \frac{\max_e c(e)}{\min_e c(e)} \cdot m^{1/\epsilon}. \end{aligned}$$

Since we set $\epsilon = 1/2$,

$$\log \max\{C_i\}/\min\{C_i\} = O(\log m + \log R).$$

■

Our main result follows from the above sequence of lemmas, and is summarized by the following theorem:

Theorem 10 *The fractional packing framework of Garg and Konemann, combined with our distributed dual solution, results in an $O(\log n + \log R)$-majorized solution for the bandwidth allocation problem. The number of iterations required is $O(m \log m)$. Each iteration requires $O(\log n)$ message passing steps. Also, during each iteration, each flow-agent sends $O(1)$ messages of length $O(\log n + \log R)$ and performs at most $O(\log n + \log R)$ units of computation.*

It is important to note that the guarantee offered by the above theorem is the same (up to constant factors) as the existential guarantee offered by theorem 2 for this problem.

References

[1] Y. Afek, Y. Mansour, and Z. Ostfeld. Convergence complexity of optimistic rate based flow control algorithms. *Journal of Algorithms*, 30(1):106–143, 1999.

[2] 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.

[3] 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.

[4] 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.

[5] R. Bhargava, A. Goel, and A. Meyerson. Using approximate majorization to characterize protocol fairness. *Proceedings of ACM Sigmetrics*, pages 330–331, June 2001. (Poster paper).

[6] N. Garg and J. Konemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *39th Annual Symposium on Foundations of Computer Science*, pages 300–309, 1998.

[7] N. Garg and N. Young. On-line, end-to-end congestion control. *IEEE Foundations of Computer Science*, pages 303–312, 2002.

[8] A. Goel and A. Meyerson. Simultaneous optimization via approximate majorization for concave profits or convex costs. *Technical report CMU-CS-02-203, Computer Science Department, Carnegie Mellon University*, December 2002.

[9] A. Goel, A. Meyerson, and S. Plotkin. Approximate majorization and fair online load balancing. *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms*, pages 384–390, 2001.

[10] A. Goel, A. Meyerson, and S. Plotkin. Combining fairness with throughput: Online routing with multiple objectives. *Journal of Computer and Systems Sciences*, 63(1):62–79, 2001. A preliminary version appeared in ACM Symposium on Theory of Computing, 2000.

[11] G.H. Hardy, J.E. Littlewood, and G. Polya. Some simple inequalities satisfied by convex functions. *Messenger Math.*, 58:145–152, 1929.

[12] G.H. Hardy, J.E. Littlewood, and G. Polya. *Inequalities*. 1st ed., 2nd ed. Cambridge University Press, London and New York., 1934, 1952.

[13] F.P. Kelly, A.K. Maulloo, and D.K.H. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49:237–252, 1998.

[14] J. Kleinberg, Y. Rabani, and E. Tardos. Fairness in routing and load balancing. *J. Comput. Syst. Sci.*, 63(1):2–20, 2001.

[15] A. Kumar and J. Kleinberg. Fairness measures for resource allocation. *Proceedings of 41st IEEE Symposium on Foundations of Computer Science*, 2000.

[16] S. Low, L. Peterson, and L. Wang. Understanding TCP Vegas: a duality model. *Proceedings of ACM Sigmetrics*, 2001.

[17] M. Luby and N. Nisan. A parallel approximation algorithm for positive linear programming. *Proceedings of 25th Annual Symposium on the Theory of Computing*, pages 448–57, 1993.

[18] A.W. Marshall and I. Olkin. *Inequalities: theory of majorization and its applications*. Academic Press (Volume 143 of Mathematics in Science and Engineering), 1979.

[19] S. Plotkin, D. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Math of Oper. Research*, pages 257–301, 1994.

[20] A. Tamir. Least majorized elements and generalized polymatroids. *Mathematics of Operations Research*, 20(3):583–589, 1995.