A practical algorithm for balancing the max-min fairness and throughput objectives in traffic engineering

Emilie Danna, Subhasree Mandal, Arjun Singh
Google Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043
{edanna, subhasreem, arjun}@google.com

***Abstract*—One of the goals of traffic engineering is to achieve a flexible trade-off between fairness and throughput so that users are satisfied with their bandwidth allocation and the network operator is satisfied with the utilization of network resources. In this paper, we propose a novel way to balance the throughput and fairness objectives with linear programming. It allows the network operator to precisely control the trade-off by bounding the fairness degradation for each commodity compared to the max-min fair solution or the throughput degradation compared to the optimal throughput. We also present improvements to a previous algorithm that achieves max-min fairness by solving a series of linear programs. We significantly reduce the number of steps needed when the access rate of commodities is limited. We extend the algorithm to two important practical use cases: importance weights and piece-wise linear utility functions for commodities. Our experiments on synthetic and real networks show that our algorithms achieve a significant speedup and provide practical insights on the trade-off between fairness and throughput.**

I. Introduction

The goal of traffic engineering [1] is to decide how to route traffic in a network in order to balance several objectives such as maximizing throughput, balancing the link utilization across the network, controlling the bandwidth allocated to competing flows in a fair manner, minimizing latency and ensuring reliable operations when traffic patterns change or parts of the network fail. Traffic engineering is critical because it allows to route significantly more traffic than simple schemes such as routing all traffic on the shortest path.

In the following, the network is specified by its set of nodes, links and link capacities. We assume that the traffic to be routed is specified by a set of commodities, each with a source, a destination and a demand (upper bound on its access rate). This corresponds to elastic flows in a backbone network where the access rate is bounded by the network topology, the transport protocol such as TCP and possibly other congestion control mechanisms [2]. The traffic engineering algorithm decides how much bandwidth to allocate to each commodity (from zero up to the upper bound on its access rate) and how to route it in a splittable way (a set of paths and how much bandwidth to allocate on each path). The splits, that is the distribution of the bandwidth allocated to each commodity among its possible paths, are to be determined by the algorithm.

In this paper, we focus on one particular aspect of traffic engineering. We address the problem of fairness in bandwidth allocation and the trade-off between fairness and throughput. Fairness represents the user point of view: network resources should be divided among users without discrimination. Throughput represents the network operator point of view: the total bandwidth allocated should be maximized so that the network resources are used as much as possible. It is important to optimize these two criteria simultaneously, but they are conflicting.

Several variants of the problem of fair bandwidth allocation have been studied in the past. There exist several fairness measures [3], in particular: max-min fairness [4] that provides the strictest measure of fairness, proportional fairness [5] that provides a trade-off between fairness and efficiency, and α-fairness [6] that generalizes both measures. An approximation of the locally max-min fair allocation for splittable multi-path routing can be obtained with a fully polynomial ϵ-approximation algorithm [7]. The exact max-min fair allocation can be obtained with a polyhedral approach [8] that may necessitate an exponential number of constraints or by solving a series of linear programs [9], [10], [11], [12], [7]. However, this last linear programming algorithm requires to solve up to number of commodities linear programs when access rates are limited, which becomes impractical as the size of the graph increases. In practice, because of solving times in the order of minutes for medium-sized graphs, it is difficult to run it every time the demands or the topology change. If it is run with outdated or smoothed data, it provides a solution that may be sub-optimal or even infeasible.

When balancing the fairness and throughput objectives, it is important to provide a flexible trade-off so that the balance can be tipped one way or the other depending on the situation. The α-fairness measure provides a flexible trade-off: proportional fairness is obtained with $\alpha = 1$, max-min fairness is modeled with $\alpha \to \infty$, and all intermediate trade-offs can be obtained with α values inbetween. Similarly, [13] provides an approximation algorithm whose input is the competitive

ratio γ that controls the trade-off. However, the output of such algorithms is not intuitive and it is difficult to explain to users. For example, the α-fair solution optimizes a global objective function and it is difficult to explain the individual contribution of each commodity to the global objective. In addition, it is difficult to tune the algorithm to obtain the exact trade-off desired: how does the network operator choose the value of α?

Our contributions are the following:

- We provide a fast linear programming algorithm for max-min fairness. Compared to the previous linear programming algorithms [9], [10], [11], [12], we significantly decrease the number of steps needed when access rates are limited. In practice, the solving time is decreased from 2-3 minutes to 10-60 seconds for various networks with 50 nodes. This algorithm is fast enough to use online, reroute the traffic when the demands or the topology change and provide the optimal solution for current data.
- We generalize this algorithm to the important practical case where some commodities are more important than others, for example because they have a higher quality of service. We further generalize it to another important practical case that arises when commodities are aggregated or when users value bandwidth with diminishing returns: piece-wise linear functions for commodities.
- We propose a novel and intuitive way to balance the throughput and fairness objectives with linear programming. It allows the network operator to precisely control the trade-off by specifying the level of fairness desired compared to the max-min fair solution or the throughput degradation compared to the optimal throughput. It makes it easy to understand the contribution of each commodity to the trade-off.

The rest of the paper is organized as follows. In Section 2, we recall the standard linear programming algorithm for max-min fairness. In Section 3, we present our extensions to speed up the algorithm and handle importance weights and piecewise-linear utility functions. In Section 4, we describe an hybrid algorithm to control the trade-off between max-min fairness and throughput. In Section 5, we present experimental results. We conclude with open questions for future work.

II. Standard linear programming algorithm for max-min fairness

This algorithm comes from [9], [10], [11], [12]. The underlying property is that the max-min fair solution is the leximin maximal solution [11]. The idea is as follows: maximize the lowest allocation with a linear program, identify the commodities that cannot get more than this allocation in any solution and fix their allocation to this level, then iterate to maximize the second lowest allocation, etc. until all allocations are fixed. The difficulty is in identifying at each step the commodities whose allocation should be fixed.

Let A be the set of arcs in the directed graph. Let C be the set of all commodities. Each commodity d has a bandwidth demand D_d: it represents the upper bound on its access rate. Let U be the set of commodities whose allocation has not been fixed yet. Let Z_i be the set of commodities whose allocation has been fixed at step i. The algorithm outline is given in Fig. 1. P_i is the following linear program:

Fig. 1. Standard linear programming algorithm for max-min fairness.

Initialization: $U \leftarrow C$, $i \leftarrow 0$
while $U \neq \emptyset$ **do**
  Maximize the i-th smallest allocation: solve P_i, its optimal value is t_i
  Perform non-blocking test: $Z_i \leftarrow$ set of commodities in U that cannot be allocated more than t_i in any solution
  Fix the allocation of commodities in Z_i to t_i
  $U \leftarrow U \setminus Z_i$
  $i \leftarrow i + 1$
end while

$$\max t \text{ such that}$$
$$X_d \leq D_d \ \forall d \in C \quad (1)$$
$$t - X_d \leq 0 \ \forall d \in U \quad (2)$$
$$t_k - X_d \leq 0 \ \forall d \in Z_k \ \forall k = 0 \dots i-1 \quad (3)$$
$$X \in F \quad (4)$$

where X_d is the total bandwidth allocated to commodity d. (1) enforces that each commodity is allocated no more than its demand. (2) enforces that the commodities that have not been fixed yet get an allocation that is greater than the minimum allocation t, which is maximized in the objective. (3) enforces that the commodities that have been fixed in all previous steps are fixed at the levels t_k previously computed. (4) enforces that there is a feasible flow corresponding to the vector of allocations $X = (X_d)_{d \in C}$. There are two standard ways to model feasibility for a multi-commodity flow: the arc formulation [11] and the path formulation [10]. We will briefly discuss the merits of both formulations in Section V.

Let t_i be the optimal objective value of P_i. P_i may have multiple solutions. We need to identify the commodities for which $X_d = t_i$ in all solutions and fix them at the next step. [10], [11] and [14] provide several ways to identify such blocking commodities. [14] and [10] provide necessary and sufficient conditions. [11] examines whether the dual value of (2) is positive. This condition is not necessary but it is easy to enforce and it still guarantees convergence because at least one commodity is fixed at each step. This is the non-blocking test we use in this paper. The downside of this approach is that it leads to a larger number of steps. At step i, it can happen that several commodities are blocking but the non-blocking test of [11] identifies only a subset of them because the test is not necessary. These remaining blocking commodities will be identified in subsequent steps $i+1, \dots, i+p$ but the allocation level will remain the same throughout these steps: $t_i = t_{i+1} = \dots = t_{i+p}$. We call such steps *degenerate*. With a necessary and sufficient non-blocking test such as [10], no step is degenerate but the non-blocking test at each step may be more complicated or more expensive. We examine in Section V what is the fraction of degenerate steps in practice.

Fig. 2. Binary and linear search algorithm for max-min fairness.

Initialization: $U \leftarrow C$
while $U \neq \emptyset$ **do**
  Binary search phase (see Fig. 3): find the saturated commodity with the highest demand so that no additional edge is saturated and fix relevant commodities.
  One linear iteration as in the standard algorithm (see Fig. 1): find the next allocation level t_i dictated by the next saturated edge and fix relevant commodities.
end while

How many steps are needed? With a necessary and sufficient non-blocking test and in the absence of upper bounds on access rates, the number of steps is at most the number of arcs [10]. However, in the presence of upper bounds on access rates, there can be as many steps as the number of commodities. Consider for example a network with infinite link capacity and commodities with distinct demands. In such a case, commodities are fixed in the order of their increasing demands and one commodity is fixed at each step.

III. Improvements and generalization for the max-min fairness algorithm

A. Speedup with binary search

At each step of the standard algorithm recalled in the previous section, the blocking commodity (or commodities) cannot be allocated more throughput because of two possible reasons: they are saturated (their demand is equal to the allocation level t_k) or an edge is saturated. In the standard algorithm, the allocation level t is increased iteratively in a linear way to handle both cases. Our idea is to handle each case in a different way. We keep the standard way of identifying saturated edges. We change how saturated commodities are identified by using binary search to find the saturated commodity with the highest demand before the next edge is saturated. This *"binary and linear"* algorithm is outlined in Fig. 2 and Fig. 3. Commodities are ordered according to their increasing demand values. For speed and numerical stability, binary search is conducted on the demand indices (after sorting) not the demand values. Duplicate demand values are eliminated in order to avoid degenerate binary steps. At the end of each binary search phase, $t_{feas} = d_{id_{feas}}$ is equal to the highest saturated demand level such that no additional edge is saturated.

$\tilde{P}(t_{current})$ is the following linear system:

$$X_d \leq D_d \ \forall d \in C \tag{5}$$
$$\min(t_{current}, D_d) - X_d \leq 0 \ \forall d \in U \tag{6}$$
$$t_k - X_d \leq 0 \ \forall d \in Z_k \ \forall k = 0 \dots i-1 \tag{7}$$
$$t_d - X_d \leq 0 \ \forall d \text{ fixed in previous binary phases} \tag{8}$$
$$X \in F \tag{9}$$

(6) saturates all commodities with demands smaller than $t_{current}$ that have not been fixed yet. (7) and (8) enforce that commodities fixed in all previous linear steps and binary steps are fixed at the levels previously computed.

Fig. 3. Binary search phase for max-min fairness.

Initialization: $id_{feas} \leftarrow$ last known feasible index
Initialization: find first value of id_{infeas} with exponential search
Choose $id_{current}$ between id_{feas} and id_{infeas}
while $id_{infeas} - id_{feas} > 0$ **do**
  $t_{current} \leftarrow d_{id_{current}}$
  Solve $\tilde{P}(t_{current})$: try to allocate up to $t_{current}$ to every commodity in U
  if $\tilde{P}(t_{current})$ is feasible **then**
    $id_{feas} \leftarrow id_{current}$
  else
    $id_{infeas} \leftarrow id_{current}$
  end if
  $id_{current} \leftarrow (id_{feas} + id_{infeas})/2$
end while
Fix relevant commodities: Each commodity $d \in U$ such that $D_d \leq t_{feas}$ is fixed to $t_d = \min(t_{feas}, D_d)$
Identify remaining free commodities: $U \leftarrow U \setminus \{d \in C : D_d \leq t_{feas}\}$

Theorem 1: The number of iterations of the "binary and linear" search algorithm is $O(|A_{sat}| + |A_{sat}| \log(|C|/|A_{sat}|))$ where $|A_{sat}|$ is the number of saturated edges.

Proof: One linear iteration is executed for each saturated edge, which explains the first term of the complexity. In each binary search phase, thanks to the exponential search initialization, the number of iterations is bounded by the logarithm of the number of commodities that are fixed in this binary search phase. Depending on whether the search finishes with a linear or a binary iteration, there are $|A_{sat}|$ or $|A_{sat}|+1$ binear search phases. There are at most $|C|$ commodities to be fixed in binary search phases. The logarithmic function is concave so the highest number of binary iterations is obtained when edge saturations are evenly spaced and each binary phase fixes $O(|C|/|A_{sat}|)$ commodities in $O(\log(|C|/|A_{sat}|))$ steps. This explains the second term of the complexity. ■

Corollary 1: The number of iterations of the "binary and linear" search algorithm is $O(|A| + |A| \log(|C|/|A|))$.

Proof: The logarithmic function is concave so the complexity function is monotonically increasing with $|A_{sat}|$. The highest number of iterations is obtained when the search is divided in the highest number of binary search phases, that is when $|A_{sat}| = |A|$. ■

When the congestion level is high, therefore many edges are saturated, repeating many small binary phases may be slow. To deal with such cases, we propose to execute one binary search phase, then switch to linear search and continue with linear search all the way to the end instead of switching back to binary search. This *"binary then linear"* search is outlined in Fig. 4. In theory, its number of iterations is higher than the "binary and linear" search algorithm. But it may be faster in practice in some cases, as we will show in Section V. When the network is well provisioned and all edges are saturated around the same level, then the "binary then linear" search is

Fig. 4. Binary then linear search algorithm for max-min fairness.

Initialization: $U \leftarrow C$

One binary search phase (see Fig. 3): find the saturated commodity with the highest demand so that no edge is saturated and fix relevant commodities.

while $U \neq \emptyset$ **do**

One linear iteration as in the standard algorithm (see Fig. 1): find the next allocation level t_i dictated by the next saturated edge or commodity and fix relevant commodities.

end while

expected to be as fast as the "binary and linear" search. On the contrary, if one edge is saturated much earlier than others, then the "binary then linear" search is expected to be slower.

Theorem 2: The number of iterations of the "binary then linear" search algorithm is $O(|C|)$.

Proof: In the worst case, the binary search phase at the beginning does not fix any commodities (no demand can be completely satisfied without saturating an edge) and one commodity is fixed at each iteration of linear search. ■

B. Fairness with a different weight for each commodity

In practice, some commodities are more important than others, for example because they have a higher quality of service. This is modelled by associating a different weight w_d to each commodity. We compute the max-min fair solution with this additional weight information by changing P_i:

$$\max t \text{ such that}$$
$$X_d \leq D_d \ \forall d \in C \tag{10}$$
$$X_d = w_d r_d \ \forall d \in C \tag{11}$$
$$t - r_d \leq 0 \ \forall d \in U \tag{12}$$
$$t_k - r_d \leq 0 \ \forall d \in Z_k \ \forall k = 0 \dots i-1 \tag{13}$$
$$X \in F \tag{14}$$
$$r_d \geq 0 \ \forall d \in C \tag{15}$$

The new variable r_d indicates the level for each commodity d. (11) links the level and the allocation for each commodity by taking into account the weight w_d. You can think of the level as the inverse of a linear utility function with slope w_d. The $\tilde{P}$ formulation is changed in the same way.

C. Fairness with a utility function for each commodity

Let us now extend the computation of the max-min fair solution when each commodity has a utility function. Utility functions arise in two situations:

- *Aggregation:* commodities with the same source and destination, but possibly different importance weights or other characteristics, can be aggregated in one commodity with a concave piece-wise linear utility function.
- *Diminishing returns for bandwidth:* users often value the first unit of bandwidth more than an additional unit of bandwidth when they already have a satisfying allocation. Users can identify thresholds of bandwidth they would like to obtain and their associated importance. This leads to piece-wise linear utility functions. Different forms of utility functions (for example logarithmic) can be approximated with piece-wise linear functions.

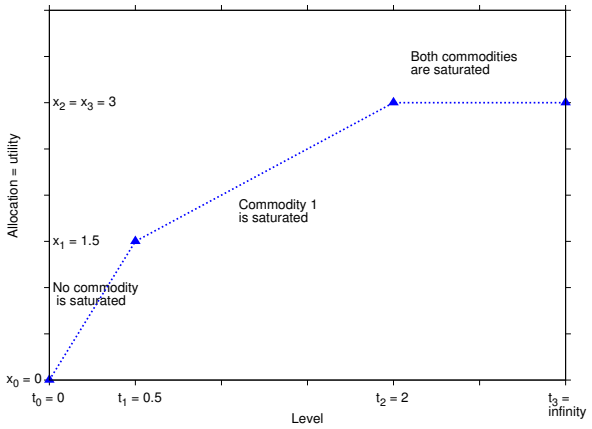

Fig. 5. Utility function obtained by aggregating commodity 1 with demand 2.0 and weight 1.0 and commodity 2 with demand 1.0 and weight 2.0.

Let the utility function u_d of commodity d be a piece-wise linear function specified by its sets of breakpoints $\{(t_d^l, x_d^l), l \in L_d\}$, as in Fig. 5. The series of linear programs is modified as follows to handle this non-linearity.

Let us consider the union of all breakpoint levels $\{t_l\}_{l \in L} = \cup_{d \in D} \{t_d^l\}_{l \in L_d}$. All utility functions can now be represented using the same breakpoint levels: u_d is represented by $\{(t^l, x_d^l)\}, l \in L\}$. The key idea is that, if we know which interval $[t^l, t^{l+1}]$ is in use, then the utility function is a linear function:

$$u_d(t) = x_d^l + \frac{x_d^{l+1} - x_d^l}{t^{l+1} - t^l}(t - t^l)$$

Each linear program P_i is changed depending on the relevant interval $[t^l, t^{l+1}]$:

$$\max t \text{ such that}$$
$$X_d \leq D_d \ \forall d \tag{16}$$
$$t - r_d \leq 0 \ \forall d \in U \tag{17}$$
$$X_d = x_{d,l} + \frac{x_d^{l+1} - x_d^l}{t^{l+1} - t^l}(r_d - t^l) \ \forall d \in U \tag{18}$$
$$X_d = u_d(t_k) \ \forall d \in Z_k \ \forall k = 0 \dots i-1 \tag{19}$$
$$X \in F \tag{20}$$
$$r_d \geq 0 \ \forall d \in U \tag{21}$$
$$t^l \leq t \leq t^{l+1} \tag{22}$$

(17) enforces that the commodities that have not been fixed yet get a level that is greater than the minimum level t, which is maximized in the objective. Following the key idea presented above, the utility function is expressed linearly in (18) and new (22) restricts the level t to be in the relevant interval. In (19), $u_d(t_k)$ is a constant equal to the utility function u_d computed at value t_k.

Each step now has an additional stopping criterion. Previously, each step of the linear search would terminate by

Fig. 6. Linear search algorithm for max-min fairness with piece-wise utility functions.

Initialization: $U \leftarrow C$, $i \leftarrow 0$
Initialization of the breakpoint index: $l = 0$
while $U \neq \emptyset$ **do**
  Maximize the i-th smallest allocation: solve P_i with interval $[t^l, t^{l+1}]$, its optimal value is t_i
  Perform non-blocking test: $Z_i \leftarrow$ set of commodities in U that cannot be allocated more than t_i in any solution
  if $Z_i \neq \emptyset$ **then**
    Fix the allocation of commodities in Z_i to t_i
    $U \leftarrow U \setminus Z_i$
  else
    Move to the next breakpoint: $l \leftarrow l + 1$
  end if
  $i \leftarrow i + 1$
end while

identifying at least one blocking commodity and fixing it at the current level. Now, a step of the linear search may terminate with no blocking commodities. In such a case, the stopping criterion is instead that the upper bound of the level interval has been reached: $t = t^{l+1}$. At the next step, the search moves on to the next interval $[t^{l+1}, t^{l+2}]$ and the expression of utility functions in (17) is changed accordingly. The algorithm is outlined more formally in Fig. 6.

The binary search is changed in a similar way. Previously the binary search was applied to the demands. Now the binary search is applied to both the demand values and the breakpoint values $(t^l)_{l \in L}$, ordered in increasing order. At each iteration of binary search, the formulation of $\tilde{P}$ is changed in the same way as P_i to use the expression of the utility function that depends on the relevant interval $[t^l, t^{l+1}]$.

Theorem 3: The number of iterations of the linear search algorithm in the presence of piece-wise linear utility functions is $O(|C| + |L|)$.

Proof: In the worst case, each iteration fixes one commodity or advances to the next breakpoint. ∎

Corollary 2: If each utility function has at most b breakpoints, the number of iterations of linear search is $O(b|C|)$.

Theorem 4: The number of iterations of the "binary and linear search" algorithm in the presence of piece-wise linear utility functions is $O(|A| + |A| \log((|C| + |L|)/|A|))$

Corollary 3: If each utility function has at most b breakpoints, then the number of iterations of the "binary and linear search" for max-min fairness in the presence of piece-wise linear utility functions is $O(|A| + |A| \log((b|C|)/|A|))$

Proof: Same reasoning as the proof in Section III-A. ∎

IV. Hybrid algorithm for a flexible trade-off between max-min fairness and throughput

The optimal throughput without fairness considerations T_{opt} is obtained with the standard linear program $(P_{throughput})$:

$$\max \sum_d X_d \text{ such that}$$
$$X_d \leq D_d \ \forall d \in C$$
$$X \in F$$

In order to obtain a trade-off between max-min fairness and throughput, we first compute the max-min fair allocation X^{MMF} as described in Section III. Then we solve a modification of $(P_{throughput})$ to optimize throughput under fairness constraints as follows. $(P'_{throughput})$ optimizes throughput under (23) that each commodity gets at least a given fraction $q_{fairness}$ of its fair allocation:

$$\max \sum_d X_d \text{ such that}$$
$$X_d \leq D_d \ \forall d \in C$$
$$X \in F$$
$$X_d \geq q_{fairness} X_d^{MMF} \ \forall d \in C \tag{23}$$

Similarly, the fairness degradation can be specified in a absolute rather than relative manner, by replacing (23) with the following constraint where $a_{fairness}$ is the largest absolute degradation allowed compared to the fair allocation:

$$X_d \geq X_d^{MMF} - a_{fairness} \ \forall d \in C \tag{24}$$

Alternatively, $(P''_{throughput})$ optimizes the fraction $q_{fairness}$ of the fair allocation that each commodity is guaranteed to be allocated, under (25) that the throughput is at least a fraction $q_{throughput}$ of the optimal throughput T_{opt}:

$$\max q_{fairness} \text{ such that}$$
$$X_d \leq D_d \ \forall d \in C$$
$$X \in F$$
$$X_d \geq q_{fairness} X_d^{MMF} \ \forall d \in C \tag{25}$$
$$\sum_d X_d \geq q_{throughput} T_{opt} \tag{26}$$
$$0 \leq q_{fairness} \leq 1 \tag{27}$$

A similar model can be formulated for an absolute rather than relative fairness degradation.

The advantages of this hybrid approach are the following:

- By choosing a value for $q_{fairness}$ or $a_{fairness}$, the network operator precisely controls the impact on fairness. In contrast, choosing a value for α in the α-fairness framework provides an indirect control on the trade-off: it is known that increasing α gives fairness more importance relative to throughput [3] but the impact on fairness or throughput is not known quantitatively.
- The network operator can control the trade-off by its impact on fairness or throughput, depending on the criterion that is most important or best understood.

- The individual contribution of each commodity to the fairness/throughput trade-off is easy to understand. In α-fairness, a global objective function is optimized and it is not clear how much bandwidth a given commodity is giving up so that the overall solution has a larger α-fairness score. In contrast, the max-min fairness criterion is easy to understand at the commodity level: in a max-min fair solution, it is not possible to increase the rate of any commodity without decreasing the rate of a commodity that already has a slower rate. Our hybrid algorithm extends the simplicity of the max-min fairness criterion because it guarantees that the decrease in allocation compared to the max-min fair allocation is bounded by a given factor for all commodities. This simplicity is important to be able to explain decisions to users when they are unhappy with their allocation.

V. Experimental evaluation

A. Experimental setup

We have run simulations on several synthetic and real networks that have been previoulsy studied in the traffic engineering literature [15], [16], [17]. The synthetic topologies consist of 2-level hierarchical graphs, random graphs and Waxman graphs [15]. The Abilene topology is a real topology [18]: we use the true backbone link capacities of 10Gbps and the traffic matrix measured on November 15th 2005 used in [16], [17]. We have also studied the Google backbone network. For confidentiality reasons, we do not present complete results on this topology. Results omitted are marked with '*' in subsequent tables. The chararacteristics of the networks studied are summarized in Table I.

In all experiments, we use the path LP formulation where, for each source-destination pair, the set of possible paths is precomputed as the k shortest paths, where k is given in the Paths column of Table I. Indeed, the path LP model is solved much faster than the arc LP model and its throughput is within 1% to 4% of the optimal throughput of the arc LP model for the values of k given in Table I.

When the capacity of the network is sufficient for all demands to be routed, as it is the case in most of our datasets, fairness and throughput do not conflict. In order to create congestion for interesting experiments, we uniformly scale up the demands by multiplying all access rates by the same factor. However, the original demand matrices present different levels of network utilization in each dataset. In order to be able to compare results across datasets, for each network, we first scale up the demands by the highest posible factor such that all demands can be routed. This initial scaling factor is provided in Table I. In the following simulations, we further multiply the demands by a factor of up to 4 to create congestion.

Our simulations were performed using the CLP 1.13.3 [19] linear programming solver on a 2.40GHz Intel dual core processor.

TABLE I
Synthetic and realistic network topologies.

Name	Topology	Nodes	Edges	Demands	Initial scaling	Paths
hier50a	hierarch.	50	148	2,450	1.84	4
hier50b	hierarch.	50	212	2,450	2.38	4
rand50	random	50	228	2,450	1.65	4
rand50a	random	50	245	2,450	1.64	4
rand100	random	100	403	9,900	1.72	4
wax50	Waxman	50	169	2,450	1.61	4
wax50a	Waxman	50	230	2,450	1.64	4
Abilene	backbone	11	28	253	2.94	4
Google	backbone	*	*	*	*	*

B. Speedup with binary search

We compare the standard linear search algorithm described in Section II, our "binary and linear" search algorithm and the "binary then linear" variant described in Section III-A. In all experiments, we used primal simplex. Dual simplex is typically the preferred algorithm to solve linear programs but, in our case, primal simplex turned out to be faster. In particular, from one step of linear search to the next, the basis remains primal feasible so it is faster to use primal simplex.

Table II presents the solving time to obtain the max-min fair allocation when demands are uniformly scaled up by a factor of 2 (beyond the initial scaling factor of Table I). We assume that paths are computed offline so we do not report the path computation time and report only the time to compute the bandwidth allocation and routing.

Fig. 7 and 8 show how the solving times depend on the congestion level by varying the demand scaling. The speedup of "binary and linear" search compared to linear search is significant for all congestion levels studied. However, as explained theoretically in Section III-A, the "binary and linear" search solving time increases as the congestion in the network increases because the number of saturated arcs increases. Conversely, the linear solving time decreases as the congestion increases because more commodities are fixed at each step. As the graph for wax50a shows, the "binary then linear" search is sometimes faster than the "binary and linear" search for high congestion levels.

The improvement in the number of iterations is significant as well. Fig. 9 shows the number of iterations and distinguishes between the type of iterations: binary, linear degenerate and linear non-degenerate. We show the two networks with the lowest (Google) and highest (hier50b) fraction of degenerate steps in linear search. For the Google network, the fraction of degenerate steps in linear search is small: 11% (all numbers in this paragraph are given for 2x demand scaling). In this case, "binary and linear" search reduces the number of iterations by a factor of 6.4 compared to linear search. Compared to a hypothetical linear search that would use a sufficient and necessary non-blocking test, therefore would not execute any degenerate steps, "binary and linear" search would still reduce the number of iterations by a factor of 5.7. Similarly, the fraction of degenerate steps in linear search is small for the Abilene network: 22%. In contrast, for the hier50b network,

TABLE II
SOLVING TIMES IN SECONDS FOR MAX-MIN FAIRNESS COMPUTATION WITH SCALING FACTOR = 2.

	Linear	Binary and linear	Binary then linear
hier50a	136.3	9.7	36.7
hier50b	113.1	9.5	76.1
rand50	199.4	37.3	51.0
rand50a	207.8	59.4	64.5
rand100	6612.0	1993.6	1701.8
wax50	174.9	23.4	74.1
wax50a	195.9	22.6	58.2
Abilene	1.72	0.36	0.34
Google	See speedups in Fig. 8		

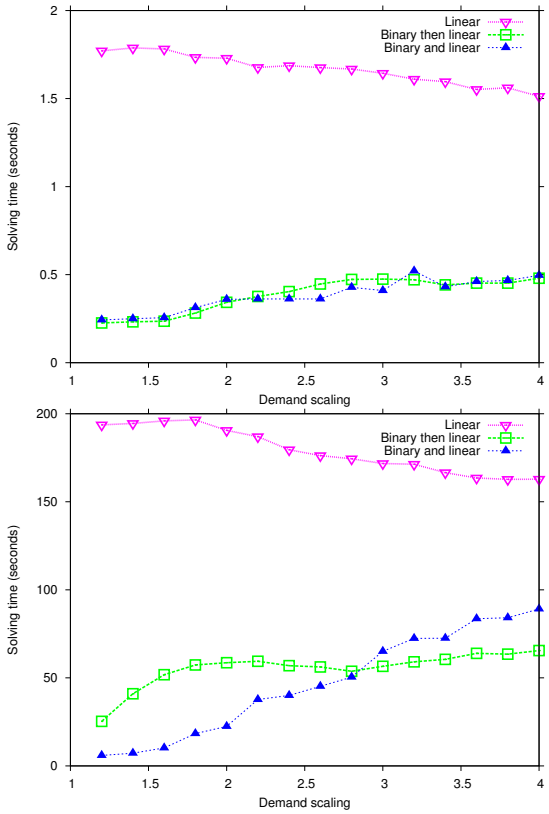

Fig. 7. Solving times in max-min fairness computation depending on the demand scaling: Abilene (top) and wax50a (bottom).

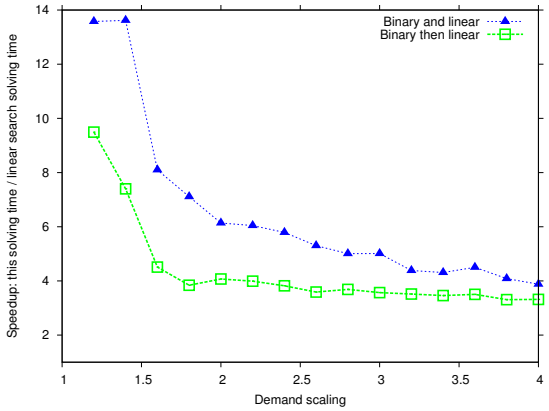

Fig. 8. Solving time speedups in max-min fairness computation compared to the linear search depending on scaling factor for the Google network.

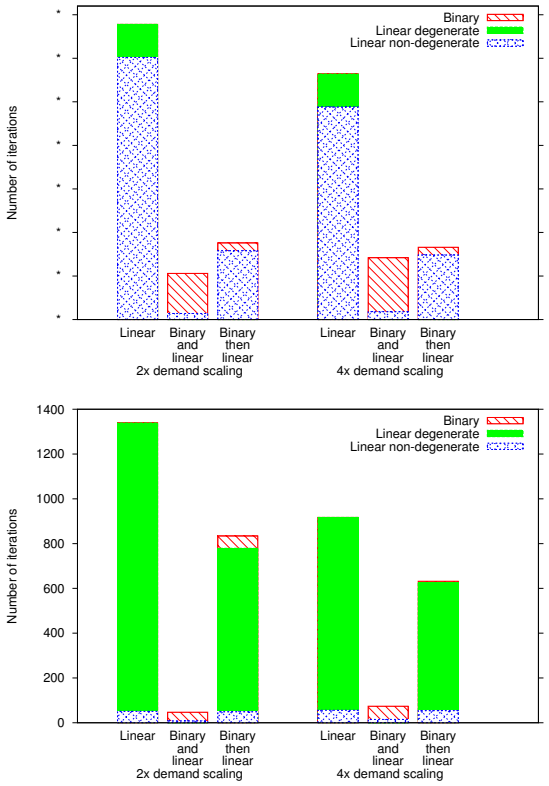

Fig. 9. Number of iterations in max-min fairness computation: Google (top) and hier50b (bottom).

the fraction of degenerate steps is very high: 96%. In this case, the "binary and linear" search reduces the number of iterations by a factor of 28.5 compared to linear search, but only 1.1 compared to linear search without degenerate steps. In such a case, beyond the number of iterations, the fastest implementation depends on the relative cost of a binary iteration, a linear iteration and the different non-blocking tests.

C. Trade-off between max-min fairness and throughput

Fig. 10 shows the flexible trade-off between fairness and throughput achieved by our hybrid algorithm described in section IV. We ran experiments for values of $q_{fairness}$ ranging from 0.0 to 1.0 in 0.1 increments. $q_{fairness} = 0$ is the solution that maximizes throughput without fairness considerations. $q_{fairness} = 1$ is the max-min fair solution. Intermediate values of $q_{fairness}$ represent different trade-offs between fairness and throughput. The horizontal axis represents the throughput degradation of the throughput $T = \sum_d X_d$ of a given solution compared to the optimal throughput obtained without fairness considerations: $\frac{T-T_{opt}}{T_{opt}}$. On the vertical axis, we show the fairness degradation of each solution compared to the max-min fair solution. For each commodity d, we measure the fairness degradation as $\frac{\max(0, X_d^{MMF} - X_d)}{X_d^{MMF}}$. There are several ways to aggregate the fairness degradation over all commodities in

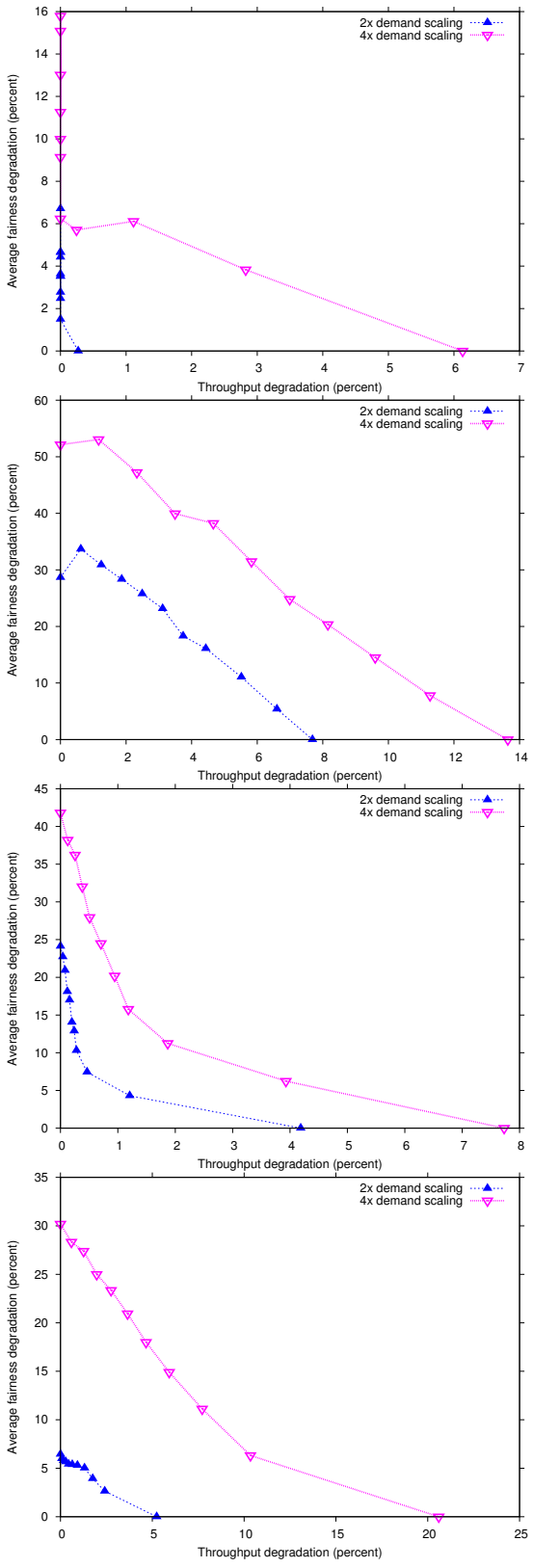

Fig. 10. Flexible trade-off between fairness and throughput for several networks, from top to bottom: Abilene, Google, hier50b and wax50a.

one number. The maximum allocation degradation over all commodities is an aggregate indicator consistent with the intent of max-min fairness. By design, it is equal to $q_{fairness}$ for our hybrid algorithm. Each point of the graph corresponds to a value between 0 and 1 in 0.1 increments. To provide additional information, we show on the vertical axis the average fairness degradation over all commodities.

Fig. 10 shows that each network has a different behavior. For example, the max-min fair requirement leads to a very small degradation in throughput for the Abilene network (0.27% for 2x demand scaling) and a large degradation for the Google network (7.7%). Simple indicators such as the link utilization and the average number of hops per path do not explain these differences. It is nonetheless interesting to see that max-min fairness and throughput do not necessarily compete much even when the congestion level is high.

D. Distribution of the fairness degradation across commodities

Let us study the allocation distribution across commodities in more details. Fig. 11 to 13 provide additional information on the wax50a network. We have seen a similar behavior on the other networks.

Fig. 11 shows the ratio hybrid allocation / fair allocation in function of the fair allocation. As expected from the definition of max-min fairness, the commodities with a hybrid allocation smaller than the max-min fair allocation are the commodities with a small allocation. As expected from the definition of the hybrid solution, the ratio hybrid / fair allocation is greater than $q_{fairness} = 0.9$ for all commodities. What is more surprising is that some commodities have a much greater allocation (up to 8.1x greater) in the hybrid allocation than in the fair allocation. The vertically aligned points (for example for fair allocation = 25.2) represent commodities that have the same fair allocation: they were fixed at the same step during max-min fairness search because they compete for the same saturated edge. In the hybrid solution, their allocation varies a lot: the hybrid solution definitely introduces some unfairness. This figure raises the question of what type of unfairness is acceptable: is it sufficient to guarantee a worst degradation compared to the max-min fair solution or is it also important to control the best improvement to avoid that the allocation of some commodities increases much more than others? Note that our hybrid algorithm can easily accomodate such an additional restriction by changing (23), (24) or (25).

Fig. 12 shows that a trade-off favoring throughput over fairness ($q_{fairness} = 0.7$ vs. 0.9) leads to fewer commodities receiving less than their fair allocation (but the degradation of each such commodity is higher) and more commodities receiving more than their fair allocation. Fig. 13 shows that a higher congestion level (demand scaling = 4x vs. 2x) leads to an increase in unfairness: the number of commodities receiving less than their fair allocation is greater, the number of commodities receiving more than their fair allocation is greater and their allocation is increased by a higher factor.

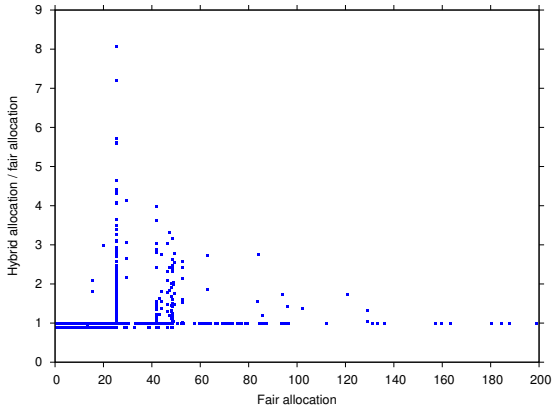

Fig. 11. Hybrid / fair allocation ratio for the wax50a network, $q_{fairness} = 0.9$ and 4x demand scaling. Each dot represents one commodity.

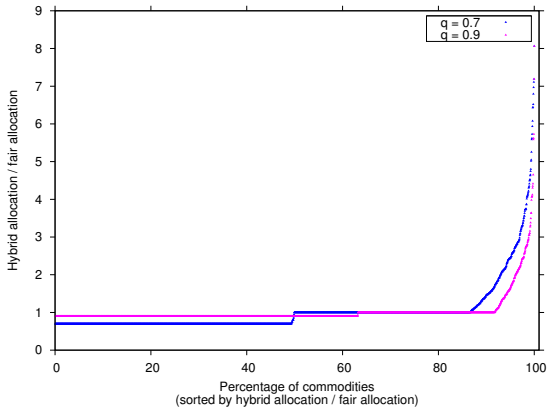

Fig. 12. Distribution of the hybrid / fair allocation ratio for the wax50a network, 4x demand scaling and different $q_{fairness}$ values. Each dot represents one commodity.

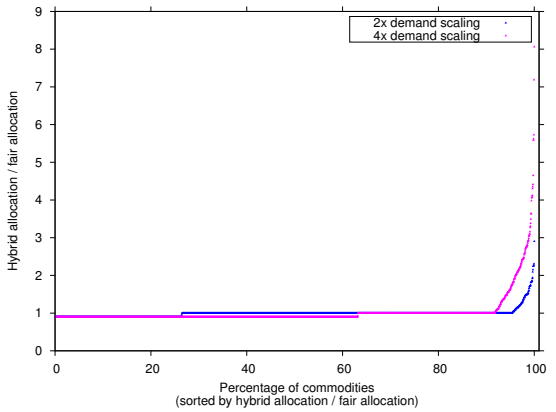

Fig. 13. Distribution of the hybrid / fair allocation ratio for the wax50a network, $q_{fairness} = 0.9$ and different demand scaling values. Each dot represents one commodity.

VI. Conclusion and Future Work

In this paper, we have provided a fast and general algorithm for max-min fairness and a hybrid algorithm to obtain a trade-off between fairness and throughput that is easy to understand.

We have shown that it is important to consider the impact of the trade-off between fairness and throughput on the allocation of each commodity rather than at an aggregate level. It remains to be investigated what kind of unfairness is acceptable: is it sufficient to guarantee a worst degradation compared to the max-min fair solution or is it also important to control the best improvement to avoid that the allocation of some commodities increases much more than others?

A second open question is to what extent fairness and throughput compete. What are the characteristics of the topology or the demand distribution for which the max-min fairness requirement does not significantly decrease throughput?

Acknowledgements

The authors thank Haim Kaplan for the exponential search initialization idea to improve the complexity proof, Jennifer Rexford and Martin Suchara for sharing datasets, and Amin Vahdat for his advice on how to write the paper.

References

[1] N. Wang, K. Ho, G. Pavlou, and M. Howarth, "An overview of routing optimization for internet traffic engineering," *IEEE Communications Surveys Tutorials*, vol. 10, no. 1, pp. 36–56, 2008.

[2] J. He, M. Chiang, and J. Rexford, "Can congestion control and traffic engineering be at odds?" in *IEEE GLOBECOM*, 2006.

[3] T. Lan, D. Kao, M. Chiang, and A. Sabharwal, "An axiomatic theory of fairness in network resource allocation," in *IEEE INFOCOM*, 2010, pp. 1–9.

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

[5] F. Kelly, A. Maulloo, and D. Tan, "Rate control for communication networks: Shadow prices, proportional fairness and stability," *The Journal of the Operational Research Society*, vol. 49, no. 3, pp. 237–252, 1998.

[6] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Transactions on Networking*, vol. 8, pp. 556–567, 2000.

[7] M. Allalouf and Y. Shavitt, "Centralized and distributed algorithms for routing and weighted max-min fair bandwidth allocation," *IEEE/ACM Transactions on Networking*, vol. 16, no. 5, pp. 1015–1024, 2008.

[8] G. Rétvári, J. J. Bíró, and T. Cinkler, "Fairness in capacitated networks: a polyhedral approach," in *IEEE INFOCOM*, 2007.

[9] D. Nace, "A linear programming based approach for computing optimal fair splittable routing," in *Computers and Communications, ISCC 2002*, 2002, pp. 468–474.

[10] M. Pioro, P. Nilsson, E. Kubilinskas, and G. Fodor, "On efficient max-min fair routing algorithms," in *Computers and Communication, ISCC 2003*, vol. 1, 2003, pp. 365–372.

[11] O. K. Dritan Nace, Linh Nhat Doan and A. Bashllari, "Max-min fairness in multi-commodity flows," *Computers and Operations Research*, vol. 35, 2008.

[12] D. Nace and M. Pioro, "Max-min fairness and its applications to routing and load-balancing in communication networks: a tutorial," *IEEE Communications Surveys Tutorials*, vol. 10, no. 4, pp. 5–17, 2008.

[13] A. Goel, A. Meyerson, and S. Plotkin, "Combining fairness with throughput: Online routing with multiple objectives," *Journal of Computer and System Sciences*, vol. 63, no. 1, pp. 62–79, 2001.

[14] M. T. Robert M. Freund, Robin Roundy, "Identifying the set of always-active constraints in a system of linear inequalities by a single linear program," MIT, Tech. Rep. 1674-85, 1985.

[15] B. Fortz and M. Thorup, "Optimizing ospf/is-is weights in a changing world," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 4, pp. 756–767, 2002.

[16] D. Xu, M. Chiang, and J. Rexford, "Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering," *IEEE/ACM Transactions on Networking*, 2011.

[17] M. Suchara, D. Xu, R. Doverspike, D. Johnson, and J. Rexford, "Network architecture for joint failure recovery and traffic engineering," in *ACM SIGMETRICS*, 2011, pp. 97–108.

[18] "Abilene backbone network," http://abilene.internet2.edu.

[19] "Clp: Coin-or linear programming," https://projects.coin-or.org/Clp.