Rapid Calculation of Max-Min Fair Rates for Multi-Commodity Flows in Fat-Tree Networks

Mollah, Md Atiqul
Yuan, Xin
Pakin, Scott D.
Lang, Michael Kenneth

To be published in: IEEE Transactions on Parallel and Distributed Systems

DOI to publisher's version: 10.1109/TPDS.2017.2746078

Permalink to record: http://permalink.lanl.gov/object/view?what=info:lanl-repo/lareport/LA-UR-17-27445

Rapid Calculation of Max-Min Fair Rates for Multi-commodity Flows in Fat-tree Networks

Md Atiqul Mollah and Xin Yuan
Department of Computer Science
Florida State University
Tallahassee, Florida 32306
Email: {mollah, xyuan}@cs.fsu.edu

Scott Pakin and Michael Lang
Computer, Computational, and Statistical Sciences Division
Los Alamos National Laboratory
Los Alamos, New Mexico 87545
Email: {pakin, mlang}@lanl.gov

Abstract

Max-min fairness is often used in the performance modeling of interconnection networks. Existing methods to compute max-min fair rates for multi-commodity flows have high complexity and are computationally infeasible for large networks. In this work, we show that by considering topological features, this problem can be solved efficiently for the fat-tree topology that is widely used in data centers and high performance compute clusters. Several efficient new algorithms are developed for this problem, including a parallel algorithm that can take advantage of multi-core and shared-memory architectures. Using these algorithms, we demonstrate that it is possible to find the max-min fair rate allocation for multi-commodity flows in fat-tree networks that support tens of thousands of nodes. We evaluate the run-time performance of the proposed algorithms and show improvement in orders of magnitude over the previously best known method. We further demonstrate a new application of max-min fair rate allocation that is only computationally feasible using our new algorithms.

Keywords

max-min fairness; fat-tree topology; high-performance computing; data center networks,

1. Introduction

The fat-tree topology is a popular candidate for both compute- and data-intensive networked systems. Its variants have been widely used in high-performance computing (HPC) clusters. Many supercomputers, including the Tianhe-2, which ranks No. 2 in the November 2016 list of the world's fastest supercomputers [1], are interconnected using this topology. The fat-tree topology is also widely deployed in data centers [2], [3]. As such, the ability to analyze and model the performance of large-scale fat trees is important and of practical significance. There are some qualitative differences, however, between fat-tree usage in Internet-facing data centers and in HPC data centers. In the former case, a substantial fraction of the total traffic goes between data-center nodes and the Internet. In the latter case, all communication is local to a single computer cluster. This paper focuses exclusively on that second, HPC, case, which presents a more controlled environment for modeling and analysis.

Max-min fairness (MMF) is commonly assumed in the modeling of network performance [4], [5]. Informally, for a given network and a pattern of traffic flows, an allocation of rates to the flows is said to be max-min fair if it is impossible to increase the allocated rate for any flow while decreasing only the rates of flows that have larger rates. MMF has gained wide acceptance in the networking community. It forms the basis for rate allocation and resource management in networks and is actively used as a benchmarking measure in a range of applications such as routing, congestion control, and performance evaluation. MMF is also viewed as an approximate presentation of the behavior of real networks [6].

Algorithms have been developed for calculating MMF rates in different settings. Gallager describes a progressive filling algorithm that computes MMF rates for networks with single-path routing [5]. A more general MMF multi-commodity flow problem (MMF-MCF), which assumes any multi-path routing with splittable flows (the packets in a flow may follow different paths), has also been studied [7]. The solution to an MMF-MCF problem is particularly interesting since it represents the theoretically optimal aggregate throughput performance that can be achieved for a given network and a given communication pattern while satisfying max-min fairness, as the results are obtained assuming optimal routing. Hence, solutions to MMF-MCF problems are often used to measure the quality of routing algorithms and the aggregate throughput performance of network designs.

While an algorithm exists for solving MMF-MCF for any arbitrary network topology [7], this algorithm has a high computational complexity because it requires iteratively solving linear programming (LP) problems in which the number of variables is proportional to the product of the number of flows and the number of links in the network. For a current or next generation fat-tree network, if one were to scale it to tens of thousands of processing nodes, the LP formulation can easily have more than 1 billion variables. LP problems of such sizes are computationally infeasible to solve with the current technology

In this work, we consider algorithms for solving MMF-MCF specifically for fat-tree topologies. We consider Parallel Ports Generalized Fat Trees (PGFT), which are extended from the family of Extended Generalized Fat Trees (XGFT) [8] and cover almost all fat-tree interconnects in HPC clusters [9]. By taking topological features into account, we develop three new algorithms for solving MMF-MCF on the fat trees. The first algorithm is based on linear programming with a much simpler formulation than that in the generic solution [7]. The second algorithm improves over the first algorithm by removing linear programming and following the progressive-filling approach that is used in the algorithm for single-path routing [5]. Our third algorithm applies further optimization techniques to the second algorithm to significantly reduce the computation complexity. The third algorithm, which is already the most efficient of the three newly proposed schemes, lends itself easily to parallelization using OpenMP [10]. Our empirical evaluation of the run-time performance of the three algorithms indicate that all three proposed algorithms achieve orders of magnitude speedups over the existing algorithm. Moreover, our algorithms are able to compute MMF rate allocation for current and next-generation fat-tree networks that support tens of thousands of processing nodes in a reasonable length of time. For example, the rate allocation for multiple permutation communication patterns on a fat tree with 11,664 processing nodes is computed on average in only a fraction of a second using a one-node server. We also demonstrate an application of the algorithms by calculating the LANL-FSU Throughput Indices (LFTI) [11], a recently proposed performance metric that requires fast calculation of the aggregate throughput under various communication patterns.

The rest of the paper is organized as follows. In Section 2 we summarize the background needed to understand our work, including the preliminaries of XGFT and PGFT, max-min fairness, and the generic LP-based algorithm for solving MMF-MCF. In Section 3 we show how fat-tree topological features can be explored to facilitate fast solutions of MMF-MCF on fat trees and then, we describe our proposed algorithms. We present our experimental results in Section 4 and present an application of our algorithms in Section 5. Finally, Section 6 draws some conclusions from our evaluations.

2. Background

2.1. XGFT and PGFT

Fat-tree topologies were first introduced by Leiserson as efficient ways to interconnect processors in parallel computers [12]. In a fat-tree network, the processing elements are located at the leaf nodes and the switching elements/routers make up the internal nodes. The family of fat-tree topologies are unified by Öhring [8] into Generalized Fat Tree (GFT) and Extended Generalized Fat Tree (XGFT) representations. Throughout this paper we use XGFT, which can represent most fat-tree variations, to describe fat trees. $XGFT(h;m_0,m_1, \ldots,m_{h-1};w_0,w_1, \ldots,w_{h-1})$ denotes a tree structure of height h, which consists of $h+1$ levels of nodes. The node levels are labeled from 0 to h in a bottom-up fashion with processing nodes at level 0. Each node in level i, $0 \leq i \leq h-1$, has w_i parents; and each node in level i, $1 \leq i \leq h$, has m_{i-1} children. $XGFT(h;m_0,m_1, \ldots,m_{h-1};w_0,w_1, \ldots,w_{h-1})$ has $(\prod_{i=k+1}^{h} m_{i-1}) \times (\prod_{i=1}^{k} w_{i-1})$ switches at level k,

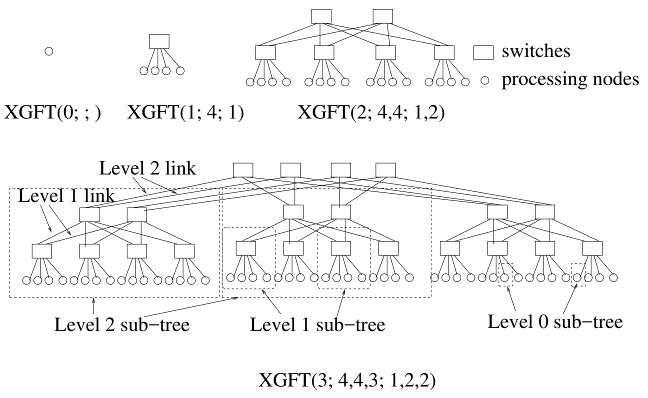


Fig. 1: XGFT examples

$1 \leq k \leq h$, and $\prod_{i=1}^{h} m_{i-1}$ processing nodes at level 0. $XGFT(h+1;m_0,m_1, \ldots,m_h;w_0,w_1, \ldots,w_h)$ can be recursively constructed by having $w_0 \times w_1 \times \cdots \times w_h$ top level switches and m_h copies of sub-fat-tree $XGFT(h;m_0,m_1, \ldots,m_{h-1};w_0,w_1, \ldots,w_{h-1})$. Each top level switch is connected to one outgoing link from each sub-fat-tree. $XGFT(h;m_0,m_1, \ldots,m_{h-1};w_0,w_1, \ldots,w_{h-1})$, in turn, has m_{h-1} copies of sub-fat-tree $XGFT(h-1;m_0,m_1, \ldots,m_{h-2};w_0,w_1, \ldots,w_{h-2})$. We will call sub-fat-trees with h levels of nodes *level h sub-fat-trees*. On a generalized fat-tree, no assumption is made on the bandwidth(capacity) of the links and as such, different links can have different bandwidths. However in practice, the same bandwidth is typically used for all links [13]. More details of constructing XGFT can be found in Öhring's paper [8].

Figure 1 illustrates the structure as well as the recursive construction of $XGFT(3;4,4,3;1,2,2)$, where $h=3$, $m_0=4$, $m_1=4$, $m_2=3$, $w_0=1$, $w_1=2$ and $w_2=2$. $XGFT(3;4;4;3;1;2;2)$ is constructed with $w_0 \times w_1 \times w_2 = 4$ top level switches and $m_2=3$ copies of level 2 sub-fat-tree $XGFT(2;4,4;1,2)$. Each level 2 sub-fat-tree $XGFT(2;4,4;1,2)$ has 4 copies of level 1 sub-fat-tree $XGFT(1;4;1)$. Each $XGFT(1;4;1)$ has four $XGFT(0;;)$ sub-fat-trees, each being a single node.

As can be seen from this example, the level k nodes are the root nodes in the level k sub-fat-trees. Links in a fat tree can also be classified based on their levels. In general, we will refer to the downlinks into a level k sub-fat-tree and the uplinks from a level k sub-fat-tree as *level k links* (uplinks/downlinks). Clearly, in $XGFT(h+1;m_0,m_1, \ldots,m_h;w_0,w_1, \ldots,w_h)$, each level k sub-fat-tree, $0 \leq k \leq h$, has $\prod_{i=0}^{k} w_i$ uplinks from the sub-fat-tree (and $\prod_{i=0}^{k} w_i$ downlinks into the sub-fat-tree). We will call the total link capacity of all level k uplinks from a level k sub-fat-tree the *total outgoing capacity* of the sub-fat-tree; and the total link capacity of all level k downlinks into a level k sub-fat-tree the *total incoming capacity* of the sub-fat-tree.

Due to the tree structure in the fat-tree topology, the (shortest) path from a source processing node s to a destination processing node d always consists of an upward path to one of the nearest common ancestors (NCAs) of both s

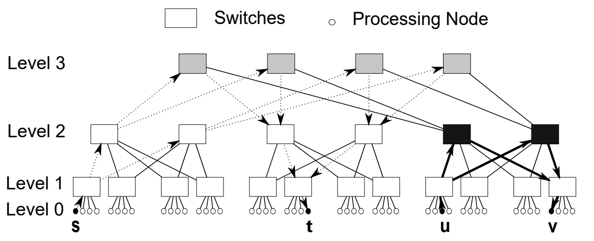


Fig. 2: Routing in a fat-tree ($XGFT(3;4,4,3;1,2,2)$)

and d, and a unique downward path from the NCA to d. Let the NCAs of both s and d be level k nodes of an $XGFT(h;m_0,m_1, \ldots,m_{h-1};w_0,w_1, \ldots,w_{h-1})$. Then, the total number of NCAs is $\prod_{i=0}^{k-1} w_i$. Each NCA connects the source and the destination nodes through a unique path. Therefore, the total number of paths between a source-destination (SD) pair having NCAs at level k is also $\prod_{i=0}^{k-1} w_i$. Figure 2 illustrates the possible data paths for two SD pairs: (s,t) and (u,v) on an $XGFT(3;4,4,3;1,2,2)$. Any traffic from s needs to reach a level 3 NCA before a downward path towards t may be found. Between s and t, there are a total of $1 \times 2 \times 2 = 4$ paths shown in dotted arrows and the same number of NCAs shown in solid gray rectangles. Similarly, the nearest common ancestors of u and v is located at level 2 and the number of paths available to route for (u,v) is thus, $1 \times 2 = 2$, shown in bold arrows. The two NCAs corresponding to those paths are shown in solid black rectangles. Routing on fat trees mostly focuses on deciding the upward paths to carry traffic for each SD pair.

In XGFT, it is assumed that all links have the same bandwidth. A more realistic variation, Parallel Ports Generalized Fat Trees (PGFT), represents the same topologies as XGFT but allows multiple parallel links to connect pairs of switches [9]. This is equivalent to providing different bandwidths between different layers of switches. PGFT describes a fat-tree as $PGFT(h;m_0,m_1, \ldots,m_{h-1};w_0,w_1, \ldots,w_{h-1};p_0,...,p_{h-1})$ where p_i, $i = 0,1,..,h-1$, specifies the number of parallel links between switches in layer i and layer $i+1$: $PGFT(h;m_0,m_1, \ldots,m_{h-1};w_0,w_1, \ldots,w_{h-1};p_0,...,p_{h-1})$ has the same topology as $XGFT(h;m_0,m_1, \ldots,m_{h-1};w_0,w_1, \ldots,w_{h-1})$ with each connection between switches in layers i and $i+1$ having p_i parallel links. Clearly, in $PGFT(h+1;m_0,m_1, \ldots,m_h;w_0,w_1, \ldots,w_h;p_0,p_1, \ldots p_h)$, each level k sub-fat-tree, $0 \leq k \leq h$, has $\prod_{i=0}^{k} w_i \times p_k$ uplinks from the sub-fat-tree (and $\prod_{i=0}^{k} w_i \times p_k$ downlinks into the sub-fat-tree). Note that in PGFT, all links have the same bandwidth, different bandwidth between switches is achieved through parallel links. XGFT is a special case of PGFT with the number of parallel links between all adjacent switch layers being 1.

2.2. Max-min Fairness (MMF)

Max-min fairness deals with fair rate allocation among a set of connections (flows) in a network with given link capacities. Let PE be the set of N processing nodes, and SE be the set of switching nodes in a system; $V = PE \cup SE$ is the set of nodes. Let E be links in the network with their corresponding capacities $C = \{C_i | i \in E\}$. We will call a set of connections (flows) a *communication pattern*, which is represented by a set of source-destination (SD) pairs (s,t) where both s and t are processing nodes ($s,t \in PE$). Let the N processing nodes be numbered from 0 to $N-1$. If S is a set then $|S|$ denotes the size of the set.

Given a communication pattern D with $|D|$ flows, a rate allocation algorithm decides an allocation vector $\gamma = (\gamma_0, \ldots, \gamma_{|D|-1})$. γ_d is the rate allocated to flow $d \in D$. For a communication pattern D, an allocation γ vector is said to be *feasible* if $\gamma_d \geq 0$ for all $d \in D$, and all link capacity constraints are satisfied. A feasible allocation γ is *max-min fair* if and only if an increase of any rate within the domain of feasible allocations must be at the cost of a decrease of some already smaller rates [5]. The formal definition of max-min fairness is as follows.

Definition 1 (Max-min fair rate allocation): A feasible allocation γ is max-min fair if and only if for any other feasible allocation γ', if $\gamma'_s > \gamma_s$ for some flow s, then there must exist some flow s_1 such that $\gamma_{s_1} < \gamma_s$ and $\gamma'_{s_1} < \gamma_{s_1}$.

In all MMF rate allocation schemes, rates are determined for flows in the order from low to high MMF rates. During the MMF rate allocation, a link is said to be *saturated* when its capacity is fully allocated. A flow is said to be *saturated* when its rate cannot be further increased (without requiring a flow with a lower rate to decrease its rate), and is thus determined.

When a network uses single-path routing with only one path associated with each flow, the MMF rate allocation problem is referred to as the MMF Simple Routing Problem (MMF-SRP), and an iterative filling algorithm has been given in [5]. The algorithm works as follows. When the algorithm starts, all flows have a 0 rate. The algorithm then uniformly increases the rate for all flows until some link is saturated. The MMF rate of the flows that use any saturated link is determined; such flows are saturated (as its rate cannot be further increased) and removed from consideration; and the link capacities are modified. The algorithm repeats the process until the rates for all flows are determined. In each iteration, the algorithm decides the MMF rate for the most limiting flows among all flows under consideration.

The MMF multi-commodity flow problem (MMF-MCF) considers more general multi-path routing with splittable flows [7]. For a given traffic pattern, MMF-MCF considers all possible paths to route each flow and the solution is the rate allocation assuming the best routing scheme for the pattern. Unlike the MMF-SRP counterpart, the MMF-MCF rates are achieved without routing constraints, and thus, are the performance upper bound for any multi-path routing scheme: single path routing and multi-path routing with non-splittable flows are special cases of multi-path routing with splittable flows.

An algorithm for solving MMF-MCF for a general network has been developed [7]. The generic algorithm (*GEN*), which is described in Figure 3, follows a similar structure as that for the algorithm to solve MMF-SRP. The algorithm finds the MMF rate allocation iteratively. In each iteration, the maximum rate for all flows under consideration, as well as the set of flows that saturate, is computed by solving a multi-commodity flow linear programming formulation, which covers the best possible multi-path routing with splittable flows. The rate for the saturated flows are then fixed for the next iteration. This process continues until the rates for all flows are determined.

```
Data: A capacitated network $(V,E,C)$ and a set of flows $D$
Result: A max-min fair allocation vector $\lambda$ of size $|D|$
1 Set $k=0$ and $L_0=\phi$;
2 while $L_k \neq D$ do
      • Set $k$ = $k$+1;
      • Solve the LP problem $Generic_k$, and compute the
        maximum rate $\alpha$ for saturated flows;
      • Identify the set $D_k$ of saturated flows; set $\lambda_d=\alpha$ for
        all $d \in D_k$;
      • $L_k = L_{k-1} \cup D_k$;
3 end
4 The flow vector $\lambda$ obtained at the last step is the max-min fair
    allocation of rates for the flows in $D$
```

Fig. 3: Generic algorithm (*GEN*) to calculate MMF rates for multi-commodity flows

In Figure 3, D_k is the set of flows saturated at iteration k and L_k is the set of flows saturated in the first k iterations. The LP formulation, $Generic_k$ for identifying saturated flows and computing the rate at iteration k, is shown in Figure 4. f_{ij}^d is rate for flow d on link (i,j). s^d and t^d represent the source and the destination node of the flow d respectively. Constraints (1) are capacity constraints; Constraints (2) are flow conservation constraints; Constraints (3) are traffic constraints for the flows whose rates have been determined; Constraints (4) and (5) are traffic constraints for the flows whose rates have not been determined; Constraints (6) indicate that the flow values must be non-negative. This LP formulation does not assume any specific routing scheme.

We note that when the maximum number of iterations in (*GEN*) is limited to one (k=1), then the corresponding LP formulation $Generic_1$ represents the maximum attainable throughput by all flows of the traffic pattern. This is equivalent to solving the *maximum concurrent flow problem* [14] that finds the maximum rate uniformly across all flows. The first iteration of the MMF algorithm would obtain the solution for the corresponding maximum concurrent flow problem. Each iteration of the MMF algorithm solves a maximum concurrent flow problem for the unsaturated flows on the graph with remaining link bandwidths.

In principle, *GEN* can be used to compute MMF rates for any network and any traffic pattern. However, the compu-

Maximize α

Subject to:

$$\sum_{d\in D} f_{ij}^d \le C_{ij},\ \forall (i,j)\in E \qquad (1)$$

$$\sum_{j:(i,j)\in E} f_{ij}^d - \sum_{j:(i,j)\in E} f_{ji}^d = 0, \forall d\in D, \forall i \notin \{s^d,t^d\} \qquad (2)$$

$$\sum_{j:(i,j)\in E} f_{ij}^d - \sum_{j:(i,j)\in E} f_{ji}^d = \begin{cases} \lambda^d, i=s^d \\ -\lambda^d, i=t^d \end{cases} \forall d\in L_{k-1}, \forall i\in V \qquad (3)$$

$$\sum_{j:(j,i)\in E} f_{ji}^d - \sum_{j:(i,j)\in E} f_{ij}^d + \alpha \le 0, \forall d\in D\backslash L_{k-1}, i=s^d \qquad (4)$$

$$\sum_{j:(i,j)\in E} f_{ij}^d - \sum_{j:(j,i)\in E} f_{ji}^d + \alpha \le 0, \forall d\in D\backslash L_{k-1}, i=t^d \qquad (5)$$

$$f_{ij}^d \ge 0, \forall d\in D, \forall (i,j)\in E \qquad (6)$$

Fig. 4: The linear programming formulation $Generic_k$

tational complexity for this algorithm is high. In particular, the algorithm can require solving $O(|D|)$ $Generic_k$'s; each $Generic_k$ can have $|D||E|$ variables $f_{i,j}^d$, where $(i,j)\in E$ and $d\in D$. For practical large fat-tree networks, this is computationally prohibitive. For example, consider computing the MMF rate allocation for a 2-dimensional nearest neighbor (2DNN) communication pattern where each node communicates with four of its neighbors. In a 3-level 36-port full bisection bandwidth tree ($XGFT(3;18,18,36;1,18,18)$) that supports 11,664 processing nodes and has 58,320 links, a 2DNN pattern will have 46,656 flows and each LP formulation for the problem will have $46{,}656\times 58{,}320 = 2{,}720{,}977{,}920$ variables. Solving LP problems of this size is not feasible using current technology. Due to the lack of available tools that can be applied to find exact solutions for large networks, approximate algorithms have been developed [15], [16]. In contrast, our algorithms can obtain exact solutions for very large fat-tree networks.

3. MMF-MCF on fat trees

The generic MMF rate allocation algorithm, *GEN*, does not make any assumption about the topology and computes the MMF rates for the best multi-path routing with splittable flows. A multi-commodity flow problem must be solved in each iteration, which results in high computation complexity. In the following, we will show that by exploiting fat-tree features and routing schemes, MMF-MCF on fat trees can be solved much more efficiently.

3.1. Fat-tree routing and topological features

We consider a general $PGFT(h;m_0,\ \ldots,m_{h-1};w_0,\ \ldots,w_{h-1};p_0,\ \ldots,p_{h-1})$ network and a given communication pattern (a set of flows), D. For each flow $d\in D$ in the communication pattern, we denote the set of all paths used for routing as $P_d=\{P_d^1,P_d^2,\ \ldots,P_d^{|P_d|}\}$, and the fraction of the traffic routed through each path as $q_d=\{q_d^k \mid k=1,2,\ \ldots,|P_d|\}$.

A routing algorithm, which we call *uniform all-shortest paths routing* (UAPR), is the base of our new formulations for solving MMF-MCF on fat trees. UAPR works as follows. For each SD pair (s,t), UAPR uses all of the shortest paths from s to t, each of the paths goes through an NCA to reach t. Let the NCAs be in at layer h. Using UAPR, the number of paths for (s,t) is equal to the number of NCAs of s and t multiplying $\Pi_{i=0}^{h-1} p_i \times \Pi_{i=0}^{h-1} p_i$, to account for the parallel up and down links in PGFT. Let the number of paths for (s,t) be X, the traffic from s to t is distributed evenly among all the paths. That is, $q_d^i = \frac{1}{X}$, for all $i = 1, 2, \ldots, X$. Note that the number of paths for (s,t) can be easily determined: if the NCAs of s and t are at level h, the number of NCAs is $\prod_{i=0}^{h-1} w_i$, and the number of paths is $\prod_{i=0}^{h-1} w_i \times \Pi_{i=0}^{h-1} p_i \times \Pi_{i=0}^{h-1} p_i$

UAPR is interesting because of the topological features of fat trees: a fat-tree topology consists of multiple levels of sub-fat-trees; and for a processing node inside a level i sub-fat-tree to communicate with a node outside the sub-fat-tree, the traffic must go through the level i uplinks from the sub-fat-tree regardless of the routing scheme. Let SUB be any level k, $0 \le k \le h-1$, sub-fat-tree of the XGFT. We define the *total outgoing rate* of SUB to be the total rate going out of SUB, and the *total incoming rate* of SUB to be the total rate going into SUB. More specifically, let γ be a rate allocation vector for the communication pattern D; let O be the set of all outgoing flows from SUB, that is, $O = \{(s,t) | (s,t) \in D$ and $s \in SUB$ and $t \notin SUB\}$. Let I be the set of all incoming flows to SUB, that is, $I = \{(s,t) | (s,t) \in D$ and $s \notin SUB$ and $t \in SUB\}$. Let R_o be the total outgoing rate of SUB, $R_o = \sum_{d \in O} \gamma_d$; and R_i be the total incoming rate into SUB, $R_i = \sum_{d \in I} \gamma_d$. Clearly, a precondition for γ to be feasible (for any routing scheme) is that the outgoing rate of any sub-fat-tree must be no more than the outgoing capacity of the sub-fat-tree and that the incoming rate of any sub-fat-tree must be no more than the incoming capacity of the sub-fat-tree. *Using UAPR, for every sub-fat-tree, all uplinks from a sub-fat-tree are evenly loaded to carry outgoing traffic from the sub-fat-tree; and all downlinks to a sub-fat-tree are evenly loaded to carry incoming traffic into the sub-fat-tree.* This is formally proved in Lemma 1.

Note that since using UAPR, all uplinks (downlinks) of a sub-fat-tree are evenly loaded, in computing MMF rate allocation for UAPR on PGFT, treating parallel links between a pair of switches as independent links that may form different paths is the same as treating the parallel links between a pair of switches as one aggregate link. In the discussion in this section, we assume different links are independent and thus, there are $\prod_{i=0}^{h-1} w_i \times \Pi_{i=0}^{h-1} p_i \times \Pi_{i=0}^{h-1} p_i$ paths between two nodes whose NCAs are in level h. In our implementation of the proposed algorithms, we treat the parallel links as one aggregate link (with the appropriate bandwidth).

Lemma 1: Given a $PGFT(h; m_0, \ldots, m_{h-1}; w_0, \ldots, w_{h-1}; p_0, \ldots, p_{h-1})$, a communication pattern D and a feasible rate allocation vector γ for D, let SUB be any level k sub-fat-tree of the PGFT with $0 \le k \le h-1$, R_o be the total outgoing rate of SUB and R_i be the total incoming rate of SUB. Using UAPR, the load on each level k uplink from SUB is $\frac{R_o}{w_0 \times \ldots \times w_k \times p_k}$ and the load on each level k downlink to SUB is $\frac{R_i}{w_0 \times \ldots \times w_k \times p_k}$.

Proof: This is straightforward from the definitions. Let us consider the load on uplinks. The logic to prove the load on downlinks is similar. As discussed in Section 2.1, for the general $PGFT(h; m_0, \ldots, m_{h-1}; w_0, \ldots, w_{h-1}; p_0, \ldots, p_{h-1})$, the level k sub-fat-tree SUB has $\prod_{i=0}^{k} w_i \times p_i$ outgoing uplinks (and $\prod_{i=0}^{k} w_i \times p_i$ incoming downlinks). Consider a flow $(s,t) \in D$ with $s \in SUB$ and $t \notin SUB$. The NCAs of s and t are all outside SUB. From the definition of UAPR, the traffic is evenly distributed among all of the outgoing uplinks from SUB. Hence, this flow contributes $\frac{\gamma_{(s,t)}}{w_0 \times \ldots \times w_k \times p_k}$ to each of the outgoing uplinks from SUB. Summing the rates of all flows going out of SUB, the load for all level k uplinks from SUB is $\frac{R_o}{w_0 \times \ldots \times w_k \times p_k}$. □

Lemma 2: Given a traffic pattern D and a PGFT where the capacity is the same for all links, if a rate allocation vector γ for D is feasible for any routing scheme r, then γ is also feasible for UAPR.

Proof: Let the capacity of each link be c in a generic $PGFT(h; m_0, \ldots, m_{h-1}; w_0, \ldots, w_{h-1}; p_0, \ldots, p_{h-1})$. Let us consider the load on an arbitrary uplink l. The proof for the load on the downlink follows a similar logic. Let l be an uplink from a level k sub-fat-tree, SUB, which has $w_0 \times \cdots \times w_k \times p_k$ outgoing links (including link l). Thus, the total outgoing capacity of SUB is $w_0 \times \cdots \times w_k \times p_k \times c$. Since γ is feasible for some routing r and since regardless of the routing, all outgoing flows from SUB must use some of the outgoing links of SUB, the total outgoing rate of SUB must be no more than the total outgoing capacity, that is, $R_o \le w_0 \times \cdots \times w_k \times p_k \times c$. From Lemma 1, using UAPR, the load for all uplink from SUB is $\frac{R_o}{w_0 \times \cdots \times w_k \times p_k} \le c$. Hence, using UAPR the capacity constraints for all uplinks are satisfied; and γ is feasible for UAPR. □

Lemma 2 indicates that the max-min fair rate allocation for any routing scheme can also be realized using UAPR. Because the max-min fair rate allocation is unique [7], the max-min fair rate allocation for any routing algorithm is also the max-min fair rate allocation for UAPR. In other words, solving the MMF-MCF problem for fat trees is equivalent to solving the MMF rate allocation assuming UAPR. This drastically simplifies the problem, allowing much more efficient solutions to be developed.

3.2. LP based solution

We develop an LP based formulation that computes MMF rate allocations for UAPR. The algorithm follows almost identical steps as the generic algorithm in Figure 3, but only solves a simpler problem that assumes UAPR in each iteration. We will prove later in Theorem 1 that this algorithm solves MMF-MCF on fat trees. The algorithm is shown in Figure 5. In our LP formulation, we no longer need to consider constraints for each combination of flows and links in the way we had to for *GEN* due to lack of routing information. By limiting the routing to UAPR, the complex LP formulation in Figure 4 is replaced by the simple one variable LP formulation in lines 5–7 in Figure 5. Our formulation finds all shortest paths for each flow using UAPR and uses the *edge-path form* to generate traffic constraints. We describe the steps of our LP formulation in the following text. The number of constraints is $|E|$ because the capacity constraint on each link needs to be specified.

```
Data:
    $N$: set of nodes in the PGFT
    $E$: set of links in the PGFT
    $C$: set of link rates (normalized to 1)
    $D$: set of flows
    $P_d$ = set of shortest paths used to realize flow $d$
Result: A max-min fair allocation vector $\lambda$ of size $|D|$
1  Set $k = 0$, $L_0 = \phi$, and $BW_{(i,j)} = C_{(i,j)}, \forall (i,j) \in E$;
2  while $L_k \neq D$ do
3      Set $k = k + 1$ ;
4      Solve the LP problem LP_PGFT_UAPR$_k$;
5          Maximize $\alpha$
6          Subject to:
7          $\sum_{d \in D \setminus L_{k-1},\ p \in P_d,\ (i,j) \in p} \frac{1}{|P_d|} \times \alpha \leq BW_{ij}, \forall (i,j) \in E$
8      Identify the set $D_k$ of saturated flows.
9      Set $\lambda_d = \alpha$ for all $d \in D_k$ and $L_k = L_{k-1} \cup D_k$
10     Adjust $BW_{(i,j)}$ by deducting the rates used by
         saturated flows.
11 end
12 The flow vector $\lambda$ obtained at the last step is the
     max-min fair allocation rate for $D$
```

Fig. 5: Algorithm *MMF_PGFT_LP*

The algorithm first initializes the iteration number (k), the set of flows whose rates have been determined ($L_0 = \phi$), and the available bandwidth on each link (to be equal to link capacity). It then iteratively computes the rates (lines 2–11). In each iteration, it solves the LP problem described in lines 5–7. Basically, this LP finds the maximum possible rate for all flows under consideration assuming UAPR. With UAPR, for each flow d, the set of paths for the flow is P_d and the number of paths is $|P_d|$. Hence, if the rate for the flow is α, it contributes $\frac{1}{|P_d|} \times \alpha$ rate on each link along each path. The constraints at Line 7 ensure that the total rate on each link is no more than its remaining link capacity. Once the maximum rate α is determined, the link loads on all links in the network can be decided and the saturated links are identified using the same technique as used by Nace et al. [7]. Any flow that uses a saturated link, is considered saturated and put in D_k. The saturated flows are assigned their rates and then removed from consideration, and the remaining bandwidth on each link is adjusted accordingly. Although this algorithm also requires solving a LP problem in every iteration, the LP formulation only contains one variable and $|E|$ constraints, much simpler than the multi-commodity formulation in Figure 3.

Theorem 1: The rate allocation computed by *MMF_PGFT_LP* is the same as that computed by *GEN* in Figure 3 for any pattern on a fat tree.

Proof: Let the communication pattern be D. We prove the theorem by induction.

Base case: In the first iteration, both algorithms compute the maximum rate for all flows in D. Assume that the generic algorithm in Figure 3 yields a maximum rate of α_1^1. The rate allocation where all flows are allocated rate α_1^1 can be realized by some multi-path routing scheme. From Lemma 2, α_1^1 can be realized by UAPR. Let us assume that *MMF_PGFT_LP* yields a maximum rate of α_2^1. Since α_1^1 can be realized by UAPR, we have $\alpha_2^1 \geq \alpha_1^1$. However, since α_1^1 is the maximum rate that can be achieved assuming any routing scheme including UAPR, $\alpha_1^1 \geq \alpha_2^1$. Hence, $\alpha_1^1 = \alpha_2^1$. As shown in [7], the set of saturated flows is unique. The unique set will be found by both algorithms.

Induction case: Assume that in the first $k-1$ iterations, both algorithms have the same L_{k-1} and the corresponding rates for the same flows in L_{k-1} are the same. In the k-th iteration, both algorithms try to decide the maximum rate that can be allocated to all flows in $D \setminus L_{k-1}$. Assume that the generic algorithm in Figure 3 yields a maximum rate of α_k^1 in the k-th iteration and that *MMF_PGFT_LP* yields a maximum rate of α_k^2: α_k^1 and α_k^2 are the maximum rate that can be allocated to all flows in $D \setminus L_{k-1}$ for the two algorithms, respectively. From Lemma 2, the rate allocation for flows in L_{k-1} and α_k^1 for all flows in $D \setminus L_{k-1}$ can also be realized with UAPR. Hence, $\alpha_k^2 \geq \alpha_k^1$. On the other hand, α_k^1 is the maximum that can be allocated for all flows in $D \setminus L_{k-1}$ with any routing including UAPR, assuming that the same rates are allocated to corresponding flows in L_{k-1}: $\alpha_k^1 \geq \alpha_k^2$. Hence $\alpha_k^1 = \alpha_k^2$. Both algorithms will find the same maximum data rate α_k^1 and the same set of saturated flows. Hence, in each iteration, both *GEN* and *MMF_PGFT_LP*, will find the same set of saturated flows with the same rate. □

Let the number of flows in a communication pattern be F, the worst case time to solve one LP formulation in the algorithm be $T(LP)$, the worst case time complexity for *MMF_PGFT_LP* is $O(F \times T(LP))$ since each iteration will find the rate for at least one flow. In practice, many flows in a pattern will have the same MMF rate and the actual time will be much better than the worst case situation. We note that analyzing the time complexity with LP solvers is a bit complicated. Although there exist worst-case polynomial-time algorithms to solve LP problems ($O(n^3)$, where n is the number of variables and constraints) [17], real-world LP solvers such as IBM's ILOG CPLEX[18] use practical algorithms such as the popular simplex method that solve LP problems efficiently in practice, but have exponential worst-case time complexity. As such, we use $T(LP)$ to represent the time to solve one LP problem.

3.3. Progressive filling algorithm

Using UAPR, given a rate for a flow, the rate contribution of the flow to every link in the fat tree can be decided based on the flow rate. Hence, one can find the maximum possible rate for all flows under consideration by raising the rate for all flows uniformly and examining all links to decide the mostly capacity-limited link. This way, for a given fat tree with available bandwidth on each link, the maximum possible rate for all flows in a pattern can be decided without solving an LP formulation. Instead, a progressive-filling-based algorithm that directly calculates the MMF rate allocation for UAPR can be devised. This is the base of the progressive-filling algorithm, *MMF_PGFT_PF*, which is shown in Figure 6.

```
**Data:**
    $N$: set of nodes in the PGFT
    $E$: set of links in the PGFT
    $C$: set of link rates (normalized to 1)
    $D$: set of flows
    $P_d$ = set of shortest paths used to realize flow $d$
**Result:** A max-min fair allocation vector $\lambda$ of size $|D|$
1  Set $k=0$, $L_0=\phi$ and $BW_{(i,j)}=C_{(i,j)}, (i,j)\in E$;
2  Set $\lambda_d=0$; $\forall d\in D$;
3  while $L_k\neq D$ do
4      Set $k$ = $k$+1; $L_k=L_{k-1}$;
5      Compute the rate limiting factor for each link
         $(i,j)\in E$;
6          $RL_{(i,j)} = \frac{BW_{(i,j)}}{\frac{\Sigma_{d\in D\backslash L_{k-1},}[d \;\; uses \;\; (i,j)]}{w_0\times w_1\times..\times w_l\times p_l}}$, where $(i,j)$ is at
           level $l$;
7      $RL=\min_{(i,j)}\{RL_{(i,j)}\}$ is the smallest rate limiting
         factor;
8      The links with the smallest rate limiting factor are
         saturated links;
9      For all $d\in D\backslash L_{k-1}$ that uses any saturated link at
         level $l$
10         $\lambda_d = RL\times(w_0\times w_1\times..\times w_l\times p_l)$;
11         $L_k=L_k\cup\{d\}$;
12         Adjust $BW_{(i,j)}$ for all links $(i,j)$ that $d$ uses;
13     end
14 end
15 $\lambda$ is the max-min fair allocation rate for $D$
```

Fig. 6: Algorithm *MMF_PGFT_PF*

In each iteration, the algorithm computes the rate limiting factor for each link (lines 5–6). Assume that link (i,j) is at level l, and that there are Y flows using the link. $\frac{1}{w_0\times\cdots\times w_l\times p_l}$ of the rate for each flow is routed through the link since there are $w_0\times\cdots\times w_l\times p_l$ such links to carry the traffic in the flow. Since the rate for all flows increases uniformly, all Y flows that use this link can at most communicate at the rate of $BW_{(i,j)}/\frac{Y}{w_0\times\cdots\times w_l\times p_l}$: this is the rate limiting factor for the link. Once the rate limiting factors for all links are computed, the smallest rate limiting factor will determine the maximum rate that all flows can have; and all links with the smallest rate limiting factor are saturated after this iteration; and all flows that use the saturated links are saturated in this iteration. In lines 10–14, the rates for these flows are determined and the remaining bandwidth on all links are adjusted.

In the worst case, each iteration will compute the MMF rate for at least one flow. Let $|E|$ be the number of links in the network, F be the number of flows, and I be the number of iterations to solve the problem. Within each iteration, in the worst case, computing the rate limiting factor can take $O(|E|\times F)$; lines 7 and 8 take $O(F)$ time; and lines (10) to (14) can take $O(|E|\times F)$ time. Hence, the time complexity of the algorithm is $O(|E|\times F\times I)$. In the worst case, it is $O(|E|\times F^2)$. In cases when the number of different MMF rates is small and the number of iterations to solve the problem is a small constant, the time complexity is $O(|E|\times F)$.

3.4. Optimizing the progressive filling algorithm

In the progressive filling algorithm *MMF_PGFT_PF* in Section 3.3, the most computation-intensive components are in the calculation of the rate limiting factors (lines 5–7) and the adjusting the link bandwidth when the saturated flows are removed (lines 10–14). Both components require iterating through all active flows. For each active flow, another iterative operation is required to scan through all the paths of that flow and to update variables(bandwidth, rate limiting factor, etc) associated with each link along those paths. Since each flow in the worst case can have a large number of paths with UAPR as discussed in Section 3.1, the worst case time complexity of each of these two components is $O(|E|\times F)$ in each iteration.

The computation complexity of *MMF_PGFT_PF* can be further reduced by utilizing the topological feature of PGFT that all links have the same capacity (different capacity between switches is achieved by using parallel links). Recall from Lemma 1 that using UAPR, for a flow with a given rate that utilizes level k uplinks (or downlinks) of a level k sub-fat-tree, the flow will contribute the same rate to each of the level k uplinks (or downlinks) of the level k sub-fat-tree. This follows that the remaining bandwidth ($BW_{(i,j)}$ in Figure 6) for all uplinks (and downlinks) of a sub-fat-tree in the PGFT will be exactly the same for all iterations in *MMF_PGFT_PF*. Hence, we can calculate the MMF-MCF rates by using the bandwidth and rate limiting factor of *one* representative uplink (and downlink) for each sub-fat-tree instead of all uplinks (or downlinks) of the sub-fat-tree as in Figure 6. This significantly reduces the time complexity of the MMF-MCF rate calculation algorithm.

Figure 7 is an example illustrating the reduction in the complexity. In the figure, we use the left uppermost uplink and downlink (which are connected with the left uppermost switch of the sub-fat-tree) of each sub-fat-tree to record to the information for the uplink and downlink of the sub-fat-tree. This is also how it is implemented in our implementation of the optimized algorithm to be described later. As shown in the figure, only the links represented by the red thick lines are to be explicitly considered in the computation. The value for each of the links represented by the black thin lines is implied. For an $h+1$ level fat-tree, the number of sub-fat-trees that a flow must pass by is at most $2h$ (h sub-fat-trees up and h sub-fat-trees down). Hence, to compute the rate limiting factors for all sub-fat-trees, we can loop through each active flow. In each iteration, the rate limiting factor of each of the $2h$ sub-fat-trees is updated: the time complexity is $O(2h\times F)$, where F is the number of active flows. For a practical fat-tree of small height (e.g. $h=3$), the time complexity is $O(F)$ which

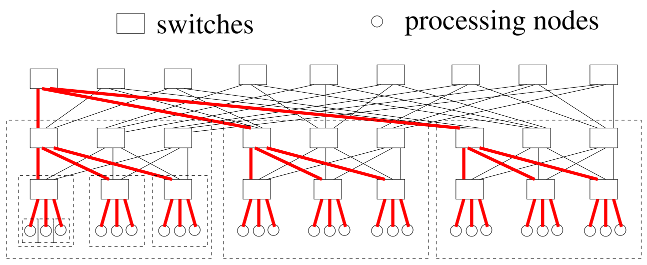


Fig. 7: The reduction of computation by using the concepts of bandwidth and rate-limiting factor of the uplink and downlink of sub-fat-tree

is significantly better than the $O(|E| \times F)$ time complexity for the same functionality in the unoptimized *MMF_PGFT_PF*.

The optimized algorithm *MMF_PGFT_OPT* is shown in Figure 8. This algorithm has the same structure as *MMF_PGFT_PF*. The only difference is that the bandwidth and the rate limiting factor are now associated with two links (one representative uplink and one representative downlink) of each sub-fat-tree instead of every link in the network. Lines 6–9 compute the smallest rate limiting factors for the uplink and downlink of each sub-fat-tree. Lines 12–16 update the bandwidth of the representative uplink and downlink of each sub-fat-tree for saturated flows.

In the worst case, each iteration will compute the MMF rate for one flow. Let $|E|$ be the number of links in the network, F be the number of flows, and I be the number of iterations to solve the problem. Assuming the height of the fat-tree (h) is a small constant, within each iteration, in the worst case, computing the rate limiting factors for all sub-fat-trees (lines 6–8) can take $O(F)$. Computing the minimum of the rate limiting factor (line 9) takes $O(|E|)$ time. The loop in lines 12–16 takes $O(F)$ time. Hence, the time complexity of the algorithm is $O((|E|+F) \times I)$, which is similar to the complexity for computing MMF with a single-path routing scheme. In the worst case, it is $O((|E|+F) \times F)$. In cases when the number of different MMF rates is small and the number of iterations to solve the problem is a small constant, the time complexity is $O(|E|+F)$.

MMF_PGFT_OPT and *MMF_PGFT_PF* share the same fat-tree data structure. In the implementation of *MMF_PGFT_PF*, we use the labeling and numbering scheme used in [19] to identify switches, links, and compute nodes in a fat tree. In *MMF_PGFT_OPT*, the same labeling scheme is used to identify the sub-fat-trees: logically the upper-leftmost switch as seen in Figure 7 of a sub-fat-tree is used to represent the sub-fat-tree, and the leftmost uplink and downlink of the switch is used to represent the uplinks and downlinks of the sub-fat-tree. Given the label of the endpoints of a traffic flow, one can enumerate the labels of each intermediate sub-fat-tree (or switch) along the flow path in constant time.

```
**Data:**
    $N$: set of nodes in the PGFT
    $E$: set of links in the PGFT
    $C$: set of link rates (normalized to 1)
    $D$: set of flows
**Result:** A max-min fair allocation vector $\lambda$ of size $|D|$
1  Set $k = 0$ and $L_0 = \phi$;
2  Set $BW_{SUB,\{up,down\}} = C_{SUB,\{up,down\})}$ for each sub-fat-tree SUB;
3  Set $\lambda_{st} = 0$; $\forall (s,t) \in D$;
4  **while** $L_k \neq D$ **do**
5      Set $k = k+1$; $L_k = L_{k-1}$;
6      Compute the rate limiting factors for each level-$i$ sub-fat-tree $SUB$ corresponding to its uplinks and downlinks;
7          $RL_{SUB,up} = \frac{BW_{SUB,up}}{\frac{\sum_{(s,t)\in D\setminus Lk-1}[s\in SUB, t\notin SUB]}{w_0 \times w_1 \times .. w_i \times p_i}}$
8          $RL_{SUB,down} = \frac{BW_{SUB,down}}{\frac{\sum_{(s,t)\in D\setminus Lk-1}[s\notin SUB, t\in SUB]}{w_0 \times w_1 \times .. w_i \times p_i}}$
9      $RL = \min_{SUB}\{RL_{SUB,up}, RL_{SUB,down}\}$ is the smallest rate limiting factor;
10     Identify the sub-fat-trees with the smallest rate limiting factor for saturation;
11     Compute the maximum rate $R$ for all unsaturated flows based on $RL$;
12     **forall** $(s,t) \in D \setminus L_{k-1}$ *using a saturated sub-fat-tree* **do**
13         $\lambda_{st} = R$;
14         $L_k = L_k \cup \{(s,t)\}$;
15         Adjust $BW_{SUB,up}$ and $BW_{SUB,down}$ for all sub-fat-tree SUB that flow $(s,t)$ uses
16     **end**
17 **end**
18 $\lambda$ is the max-min fair allocation rate for $D$
```

Fig. 8: Algorithm *MMF_PGFT_OPT*

3.5. Parallelization of *MMF_PGFT_OPT*

MMF_PGFT_OPT has the lowest time complexity among the newly proposed algorithms. We further improve its efficiency for shared memory architectures such as machines with multi-core CPUs by parallelizing this algorithm with OpenMP [10]. As discussed in the previous subsection, the main complexity is in the calculation of the smallest rate limiting factor (lines 6–9 in Figure 8) and the updating of sub-fat-tree uplink and downlink bandwidth (lines 12–16). Our parallelization effort focuses on these two components.

As can be seen from Figure 8, the rate limiting factor for the uplink of a sub-fat-tree depends on its bandwidth as well as the number of active flows that use it. The same observation applies for downlinks too. To compute the the smallest rate limiting factor, *MMF_PGFT_OPT* has two loops: the first loop iterates through all active flows and counts the number of flows using each uplink and downlink of sub-fat-trees; and the second loop computes the rate limiting factor for each link and finds the minimum. One parallelization strategy is to directly parallelize both loops. Since different flows can affect the counters for different sub-fat-trees, to execute the first loop in parallel, the counters for the uplink and downlink of each sub-fat-tree must be updated using atomic operations. Another strategy to parallelize this loop is to pre-calculate for each uplink and downlink of sub-fat-trees, all flows that use the uplink/downlink. This can be done before the outer-loop. With this information, counting the number of flows using each uplink and downlink of sub-fat-trees can be done by iterating through all sub-fat-trees and counting the active flows for each sub-fat-tree. We empirically decided that the second approach is more effective in most practical cases, and adopt this approach to report the results in our performance evaluation. The second loop computes the minimum of all rate limiting factors. This loop can be parallelized by declaring the minimum rate limiting factor as a reduction variable with the MIN operator.

The second component is in the loop in lines 12–16 in Figure 8. The loop iterates through all active flows and checks for saturated flows, which are flows using a saturated uplink or downlink of a sub-fat-tree. Within each iteration, it assigns the flow rate to the saturated flow (line 13), marks the flow as saturated (line 14), and update the bandwidths of uplinks/downlinks of sub-fat-trees that are affected by the saturated flow (line 15). In the implementation, each flow is initially assigned a flow rate of -1, which also marks that the flow is not saturated. Hence, assigning a positive flow rate to a flow in line 13 also implicitly marks the flow as saturated: line 14 is not necessary in the implementation. This loop can also be parallelized in a straightforward manner: line 13 causes no dependence; and line 14 is removed in the implementation. However, different flows can update the bandwidth of the same uplink or downlink of a sub-fat-tree. Hence, the adjustment of $BW_{SUB,\{up,down\}}$ in line 15 must be performed using atomic operations. Another parallelization strategy is to split this loop into two loops. The first loop iterates through all sub-fat-trees, identifies the saturated flows, and assigns the flow rates for the saturated flows. The second loop iterates through all sub-fat-trees and adjust $BW_{SUB,\{up,down\}}$ accordingly. With the pre-calculation of flows for each uplink and downlink of sub-fat-trees, no atomic operation is necessary. The implementation that we use to report performance results uses the second parallelization strategy. We note that the bandwidth update operations in the last loop can be merged with the load calculation loop of the subsequent loop. Therefore in our implementation, the effective bandwidth of each sub-fat-tree is evaluated in a lazy fashion in the same parallel region that implements lines 6–9 to reduce the fork/join overhead of our parallel routine.

4. Performance evaluation

We compare the run-time performance of *MMF_PGFT_LP*, *MMF_PGFT_PF* and *MMF_PGFT_OPT*, our three newly developed algorithms, with the performance of an implementation of the generic multi-path max-min fair rate allocation algorithm given by Nace et al. [7]. In addition, we also implement and use the classical, progressive-filling based MMF-SRP algorithm given by Bertsekas et al. as a performance reference [5]. The single-path routing algorithm for the MMF-SRP algorithm is the widely used destination-mod-k routing [19]. Note that computing MMF rate allocation with MMF-SRP is commonly considered to be efficient [5]. We denote the generic algorithm for the MMF-MCF problem as *GEN*, the MMF-SRP solution algorithm with single-path destination-mod-k routing as *DMK*, our linear programming based algorithm *MMF_PGFT_LP* as *LP*, our progressive filling based algorithm *MMF_PGFT_PF* as *PF* and lastly, our optimized algorithm with sub-fat-tree based calculation *MMF_PGFT_OPT* as *OPT*. We evaluate the performance of these algorithms by recording the total amount of time required to calculate rate allocation by each algorithm on different practical fat-tree topologies and with different communication patterns. Note that all of the algorithms except *DMK* yield the same results (within numerical errors).

As discussed in Section 3.1, using our algorithms to compute MMF rates on PGFT, parallel links between a pair of switches in PGFT can be treated as an aggregate link since all our algorithms are based on UAPR. The computation complexity of our algorithms is not affected by the number of parallel links in PGFT. Hence, we only use XGFT in the evaluation. The topology instances considered in this set of experiments are 2-level and 3-level full bisection bandwidth fat trees with 24- and 36-port switches, consisting of a *2-level 24-port* full bisection tree ($XGFT(2;12,24;1,12)$) that supports 288 processing nodes, a *2-level 36-port* full bisection tree ($XGFT(2;18,36;1,18)$) that supports 648 processing nodes, a *3-level 24-port* full bisection tree ($XGFT(3;12,12,24;1,12,12)$) that supports 3,456 processing nodes, and a *3-level 36-port* full bisection tree $XGFT(3;18,18,36;1,18,18)$ that supports 11,664 processing nodes. In all topology instances considered, the link capacity is assumed to be the same throughout the network.

The following traffic communication patterns were used in our study: (1) random permutation patterns (*Perm.*) where each node sends traffic to one destination and receives traffic from exactly one source, and (2) random 2-dimensional nearest-neighbor communication patterns ($2DNN$) where each node communicates with 4 nearest-neighbors on a random sized grid, and (3) random patterns with each source node sending traffic to 20 random destination nodes ($RANDN(20)$). We note that these patterns also represent increasing problem sizes given the same network configuration.

A $RANDN(20)$ problem takes significantly longer time to solve than a $2DNN$ problem, which in turn takes much longer time than a *Perm.* problem. To more or less offset the timing differences for different type of problems, the timing results reported for *Perm.* is the total time to solve 20 random permutation problems; the timing results reported for $2DNN$ is the total time to solve 5 random 2DNN problems; the timing results reported for *RANDN*(20) is the timing for solving one *RANDN*(20) problem.

All experiments in this section are run on a server equipped with an intel Core i7-5820K CPU (having 6 physical cores running at 3.3 GHz clock speed and 16MB L3 cache) and 64GB of memory. The linear programming problems used in *GEN* and *LP* are solved by the IBM ILOG CPLEX Optimization Studio software version 12.5.1.0 [18]. The algorithms in discussion, along with the corresponding routing mechanisms, are implemented in the C programming language and compiled with the GNU Compiler Collection (GCC) version 4.8.4. The programs are run in the Ubuntu 14.04 LTS operating system environment. For each algorithm, we report the average time taken to solve 2 random instances of each traffic pattern category. As we scale up the network size, the execution time of some of our routines increases beyond several days. Therefore, we set a time threshold of 30 hours for all of the experiments. Only measurements that complete within that threshold are reported.

4.1. Timing Results for sequential algorithms

Table 1 lists the experimental results of the algorithms *GEN*, *LP*, *PF*, *OPT* and *DMK*. The generic MMF-MCF solver *GEN* takes longer than our 30 hour time limit for all traffic inputs used in Table 1. As a special case, we observe the performance of *GEN* on our smallest problem instance: the 2-level 24-port fat tree (XGFT(2; 12, 24; 1, 12)) with 288 processing nodes and a single permutation pattern (288 flows). Even on such small network and traffic pattern sizes, *GEN* takes 1 hour and 49 minutes on average to solve each problem instance. This indicates that even though *GEN* can be used to compute MMF-MCF rate for any topology, it is computationally too expensive for almost any practical cases. Note that *GEN*, *LP*, *PF*, and *OPT* all produce the same results (within numerical errors), while *DMK* yields different rate allocation since with DMK, only single path routing scheme is used to route each flow.

As can be seen from Table 1, by simplifying the LP formulation with fat-tree specific features, our *MMF_PGFT_LP* can solve all but the largest problem (*RANDN*(20) on XGFT(3; 18,18,36;1,18,18)) within the time limit. By directly computing the rate limiting factor without using the LP formulation, our *MMF_PGFT_PF* significantly reduces the execution time of *MMF_PGFT_LP*. Furthermore, *MMF_PGFT_OPT* further reduces the execution time of *MMF_PGFT_PF* very significantly. As discussed in Section 3, *MMF_PGFT_OPT* improves the time complexity of *MMF_PGFT_PF* from $O(|E| \times F \times I)$ to $O((|E|+F) \times I)$, where $|E|$ is the number of edges in the graph, F is the number of flows in the problem, and I is the number of iterations to allocate all rates. The reduction of the complexity is confirmed in our experience.

As discussed in Section 3, *MMF_PGFT_OPT* has time complexity similar to that of *DMK*. However, in our experiments, *MMF_PGFT_OPT* consistently takes less time than *DMK*. Such performance difference is attributed to the load-balancing property of single-path and multi-path routing schemes. The multi-path UAPR routing used by *MMF_PGFT_OPT* tend to utilize the network uniformly, causing many links to have equal loads and thus, to be saturated in the same step of the MMF-MCF rate calculation algorithm. On the other hand, the single-path routing used by *DMK* causes relatively imbalanced load distribution over the network, resulting more distinct link load values and thus, requiring more iterations to compute the MMF-MCF rates. Therefore, even though *MMF_PGFT_OPT* has similar time complexity as *DMK* for every iteration, the total time to compute rate allocation is usually less than *DMK*, sometimes significantly. Since *DMK* is in general considered to be efficient, we conclude that MMF-MCF rate allocation for fat-trees can be computed efficiently with *MMF_PGFT_OPT*.

4.2. Performance of parallelized *MMF_PGFT_OPT*

Table 2 shows the timing results of *MMF_PGFT_OPT*. Two 3-level fat-trees, 24-port ($XGFT(3;12,12,24;1,12,12)$) and 36-port($XGFT(3;18,18,36;1,18,18)$), are considered. We run the OpenMP version with different numbers of threads for the three traffic patterns, *permutation*, 2*DNN*, and *RANDN*(20). As can be seen from the table, when the number of threads is set to 1, the timing results are the same as that in the sequential implementation. When the number of threads increases, there are two situations. For the *permutation* and 2*DNN* problems, the problem size is not large enough to explore parallel execution: using more threads do not have significant impacts on the execution time. This is because after the optimization in *MMF_PGFT_OPT*, the parallel region is small when the number of flows is not sufficiently large. In such cases, the benefits of parallel execution is overshadowed by the overhead of parallel execution. When there is a sufficiently large number of flows to be considered in a problem, parallelization can have significant benefits. For example, with the *RANDN*(20) pattern, the parallelization achieves a speed-up of 3.4 with 4 threads for $XGFT(3;12,12,24;1,12,12)$ and a speed-up of 3.5 with 4 threads for $XGFT(3;18,18,36;1,18,18)$. This demonstrates the effectiveness of the parallelized *MMF_PGFT_OPT*.

5. An application

We demonstrate an application of the proposed algorithms by showing how the new algorithms facilitate the evaluation of the potential performance degradation by limiting the routing to the popular destination-mod-*k* single path routing [19] on current generation large-scale fat trees. In particular, we model the throughput performance of fat trees using the newly proposed LANL-FSU Throughput Indices (LFTI) [11] and compare the performance of a fat tree with the destination-mod-*k* routing (in terms of LFTI) to that without any routing constraint. The ability to solve an MMF-MCF problem rapidly for a given communication pattern is essential for computing LFTI for a fat tree without routing constraints.

LFTI characterizes the performance of interconnects by capturing the throughput behaviors corresponding to the common

TABLE 1: Timing results in seconds (s)

Topology	Pattern	Average Execution Time				
		GEN	*LP*	*PF*	*OPT*	*DMK*
$XGFT(2;12,24;1,12)$ (288 proc. nodes)	*Perm.*	> 30h	0.593s	0.150s	0.003s	0.096s
	2DNN	> 30h	1.228s	0.145s	0.002s	0.002s
	RANDN(20)	> 30h	127.618s	0.592s	0.033s	0.093s
$XGFT(2;18,36;1,18)$ (648 proc nodes)	*Perm.*	> 30h	1.016s	0.348s	0.007s	0.513s
	2DNN	> 30h	3.606s	0.324s	0.006s	0.005s
	RANDN(20)	> 30h	323.986s	3.672s	0.007s	0.532s
$XGFT(3;12,12,24;1,12,12)$ (3,456 proc. nodes)	*Perm.*	> 30h	27.222s	4.172s	0.031s	0.326s
	2DNN	> 30h	7.783s	2.278s	0.024s	0.027s
	RANDN(20)	> 30h	30,080.077s	3,717.002s	10.189s	29.63s
$XGFT(3;18,18,36;1,1818)$ (11,664 proc. nodes)	*Perm.*	> 30h	230.870s	26.151s	0.150s	1.083s
	2DNN	> 30h	38.447s	13.532s	0.089s	0.134s
	RANDN(20)	> 30h	> 30h	96,730.001s	310.650s	782.923s

TABLE 2: Performance of the parallelized *MMF_PGFT_OPT* with 3-level fat-trees (s: seconds)

Topology	pattern	Number of Threads		
		1	2	4
24-port (3,456)	*Perm.*	0.031s	0.032s	0.038s
	2*DNN*	0.024s	0.025s	0.024s
	RANDN(20)	10.189s	5.31s	2.985s
36-port (11,664)	*Perm.*	0.150s	0.137s	0.115s
	2*DNN*	0.089s	0.092s	0.085s
	RANDN(20)	310.650s	168.09s	89.085s

types of HPC communication patterns. LFTI considers a comprehensive spectrum of the common communication traffics observed in HPC, classifies them into several pattern types and then, generates traffic workload samples for each pattern type. Given a network specification and routing information, LFTI first approximates the aggregate throughput of the network by taking the average of throughput measures resulting from several workloads of the same type. Then, the average aggregate throughput is normalized to the throughput of a crossbar switch that connects the same number of processing nodes as the given network does. The resultant metric is an LFTI *throughput index* corresponding to the given network and traffic pattern type pair. Hence, an LFTI index of 1 means that the performance of the interconnect is equivalent to a crossbar switch. The final LFTI output is a set of throughput indices that can express the throughput behaviors of a network for each type of traffic patterns separately. For further details, we refer the reader to the paper that introduces LFTI [11].

LFTI is used to compare throughput performances among different routing schemes on a given network specification. The network throughput resulting from MMF-MCF rate allocation can be a good theoretical benchmark for making such comparisons with LFTI. For networks other than fat-trees, calculating LFTI for MMF-MCF rate allocation is impossible due to the computational complexity of the *GEN* algorithm. However, for fat trees, we can use the newly developed algorithms to compute the MMF-MCF throughput performance as a reference for evaluating other fat-tree routing schemes. We note that the goal of this application is not to replace existing routing schemes with the optimal routing derived from MMF-MCF calculation, but to rather demonstrate an MMF-MCF based framework where the performance of any routing scheme, including and beyond the destination-mod-*k* routing, may be evaluated.

In this paper, we select a subset of the communication pattern types considered in the LFTI paper [11]. The communication patterns considered in this paper and their brief explanations are listed in Table 3. LFTI also uses different throughput indices for different node mapping schemes that determine physical communication patterns. We consider two mapping schemes in this paper, *whole system direct map* and *whole system random map*. With whole system direct map, it is assumed that the whole system runs one application with one communication pattern. The application processes are mapped to nodes using the identity function, allowing the node mapping scheme to exploit the network locality to improve throughput performance. With whole system random map, it is assumed that the whole system runs one application with one communication pattern and each application process is mapped to a random processing node with equal probability. This scheme characterizes the raw performance of the interconnect topology with no optimization in the node mapping.

To compute LFTI, it is imperative that the throughput of each communication pattern can be modeled efficiently. To evaluate the limitation of destination-mod-*k* routing, we compute the LFTI for each type of pattern assuming the destination-mod-*k* routing and MMF using the *DMK* algorithm. We then compare the fat-tree LFTI with destination-mod-*k* to the corresponding LFTI without routing constraints, in which case the parallelized *MMF_PGFT_OPT* algorithm with 4 threads is used to compute the throughput performance for a pattern.

The topologies used in this experiment are (1) $XGFT(3;18,18,36;1,18,18)$ a current generation large-scale fat-tree topology with full bisection bandwidth supporting 11,664 nodes, and (2) $XGFT(3;24,24,36;1,12,12)$: a slimmed fat-tree topology with 4:1 over-subscription ratio between the switch levels, supporting 20,736 nodes. The link capacities are assumed to be the same in both topologies. Figure 9 shows the LFTI indices for the full bisection bandwidth fat tree and Figure 10 shows the same for the slimmed fat tree. The LFTI indices calculated using the parallelized *MMF_PGFT_OPT* algorithm with 4 threads are

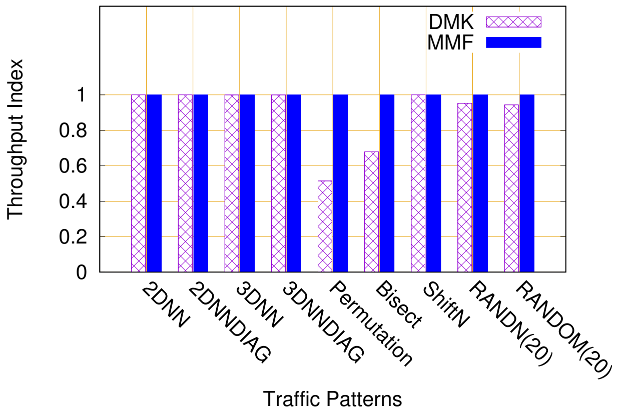
TABLE 3: HPC communication patterns used in evaluation

Pattern	Description
2DNN	average throughput for all 2-dimensional nearest neighbor patterns with random 2D grid sizes
2DNNDIAG	average throughput for all 2-dimensional nearest neighbor with diagonals patterns with random 2D grid sizes
3DNN	average throughput for all 3-dimension nearest neighbor patterns with random 3D grid sizes
3DNNDIAG	average throughput for all 3-dimension nearest neighbor patterns with random 3D grid sizes
Permutation	average throughput for all permutation patterns
Bisect	average throughput for all bisection patterns; same as effective bisection bandwidth
Shift	average throughput for all shift patterns
RANDN(20)	average throughput for random patterns with 20 random destinations for each source node
RANDOM(20)	average throughput for random patterns, each consisting of *nprocs* $\times$ 20 randomly chosen SD pairs where *nprocs* is the number of processing nodes

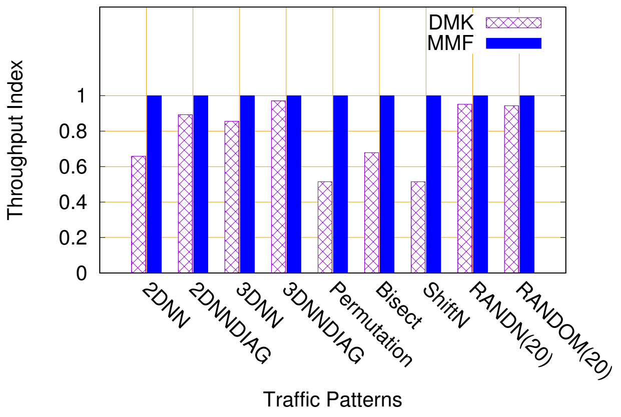
labeled as *MMF* in the figures, as any of our MMF-MCF implementations would produce the same set of indices.

For the full bisection bandwidth fat-tree (Figure 9), destination-mod-*k* achieves high performance for 2D and 3D stencil patterns with the whole system direct map scheme, but are not as effective for other patterns with global communication such as permutation and bisect when used with the whole system direct map scheme. This confirms observations from earlier studies [9], [11] that when node mapping is properly controlled, fat-trees with destination-mod-k routing can be very effective for many HPC traffic patterns. With random map, the patterns become more random; and fat-tree with destination-mod-k routing become less effective. On average, destination-mod-*k* achieves about 89% of the optimal routing with whole system direct map. With whole system random map, the performance gap between destination-mod-*k* and the optimal is wider for the stencil patterns and the shift pattern. On average, destination-mod-*k* is about 77% of the optimal. This indicates that destination-mod-*k* is sensitive to process mapping. Without routing constraints, the full bisection fat-tree achieves 100% performance of a crossbar switch while maintaining max-min fairness among traffic flows.

For the slimmed fat tree (Figure 10), the throughput indices are predictably lower due to over-subscription. Since slimmed fat trees do not have full bisection bandwidth, even the optimal interconnect performance without routing constraints is not equivalent to a crossbar switch. This makes crossbar switches a loose upper-bound to the performances of slimmed fat trees and we cannot infer much information about the performance of any routing by comparing to crossbar performances only. By comparing to the maximum obtainable MMF throughput on the same slimmed-down network with no routing constraint, Figure 10 provides a better estimate of the performance of destination-mod-*k* routing as compared to what can be achieved without the routing constraint. In other words, the LFTI results in Figure 10 quantifies the performance margin that can be improved by improving the destination-mod-*k* routing. On average, destination-mod-*k* routing on our slimmed



(a) Whole system direct map

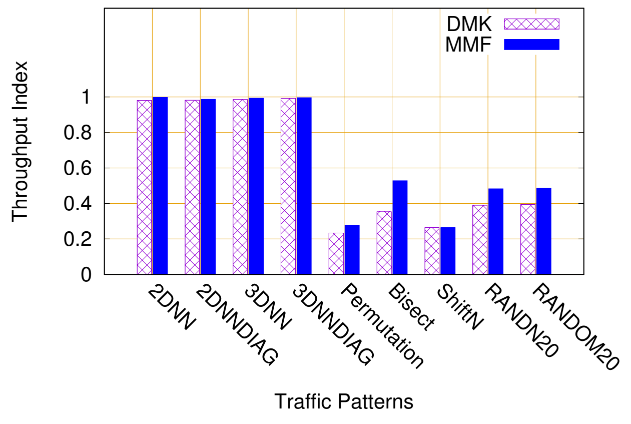


(b) Whole system random map

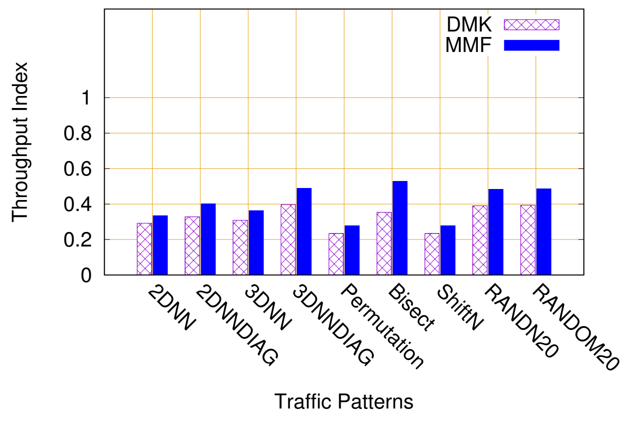
Fig. 9: LFTI summary of a 11,664-node fat-tree network ($XGFT(3;18,36,36;1,18,18)$)

fat-tree configuration achieves 89% of the optimal throughput performance with whole system direct map and 81% with the whole system random map.

For all of the experiments, the parallelized *MMF_PGFT_OPT* with 4 threads computes the LFTI much faster than *DMK*. Table 4 shows the statistics of the time and the average number of iterations taken to compute LFTI for the slimmed $XGFT(3;24,24,36;1,12,12)$ with the whole system random map. When calculating LFTI, we report the average time to solve one problem of each kind regardless of the problem size whereas the timing results in the preceding section showed the time to solve multiple problems for certain patterns. The statistics for other topologies and/or node mapping schemes are similar. As can be seen from the table, the number of iterations to solve each problem is much smaller for multi-path routing than for single-path routing. This partially explains why the parallelized *MMF_PGFT_OPT* computes the LFTI much faster than *DMK*. We note that the number of iterations only gives a rough estimation of the execution time. The final execution time also relies on how the rates are computed. For example, the algorithm that assigns rates to more flows in the



(a) Whole system direct map



(b) Whole system random map

Fig. 10: LFTI summary of a 20,736-node fat-tree network ($XGFT(3;24,24,36;1,24,24)$)

earlier iterations will out-perform an algorithm that assigns rates to a fewer number of flows in the earlier iterations since the execution time of each iteration depends on the number of actives flows in the iteration; and once the rate for a flow is computed, the flow is marked as inactive and will no longer be considered in future iterations. Nonetheless, the results in Table 4 shows that our parallelized *MMF_PGFT_OPT* is a practical scheme that can be used to compute LFTI for the current and future generation of large fat-tree topologies.

6. Conclusions

The max-min fair (MMF) communication rate establishes an upper bound on the communication performance achievable by altering only the routing algorithm. It is therefore a worthwhile metric to compute when designing and evaluating routing algorithms. However, while MMF is quick to compute for networks containing only a single path between any source and destination, it is extremely slow to compute for networks used in practice in high-performance computing (HPC) systems, all of which contain multiple paths between peers. This is because the best algorithm to date for solving the multiple-path case (a.k.a. MMF multi-commodity flow or MMF-MCF) is based on linear programming (LP), which is computationally infeasible at the billion-variable scale needed to represent a modern HPC system's network.

TABLE 4: Statistics of the timing and the average number of iterations to solve each problem in the calculation of the *LFTI* of a 20,736-node fat tree $XGFT(3;24,24,36;1,12,12)$ with the whole system random map (s:seconds)

pattern	DMK		Parallelized OPT	
	avg. iter.	time	avg. iter.	time
2DNN	7,559	58.22s	1,021	1.25s
2DNNDIAG	9,814	165.36s	2,123	7.75s
3DNN	8,637	85.37s	1,529	2.87s
3DNNDIAG	14,020	1,149.33s	3,288	91.98s
Permutation	857	3.92s	37	0.04s
Bisect	116	0.52s	52	0.03s
ShiftN	844	3.86s	39	0.04s
RANDN(20)	16,117	1,168.79s	4,531	104.88s
RANDOM(20)	16,121	1,180.27s	5,064	119.64s

The main conclusion to draw from our work is that it is possible to compute the MMF communication rate in the multi-path case in time comparable to—and at times, faster than—that needed to compute a single-path MMF rate by taking into account the network's specific topology instead of considering only the general case of arbitrary topologies. Via an empirical evaluation we have demonstrated that our algorithms, which target commonly used fat-tree topologies, are significantly faster than the existing generic solution.

A secondary conclusion is that destination-mod-k routing, which is the quintessential routing algorithm for fat-tree networks, achieves an average of about 77-89% of optimal routing (as determined by the MMF rate) across a suite of HPC communication patterns at a scale of over 10,000 network endpoints. This is the first time that the performance of destination-mod-k routing has been compared to optimal routing at such a large scale and is made possible by our algorithms' ability to so quickly compute MMF in the MMF-MCF case.

A natural avenue for future research would be to investigate ways to extend the concepts utilized in this paper to other networks used in modern HPC systems such as meshes/tori and dragonflies. Nevertheless, by being able to rapidly compute max-min fair rates for multi-commodity flows on fat trees it is finally possible to evaluate routing algorithms at scale not only in terms of how much better they are than prior routing algorithms but also in terms of how close they come to the best possible routing algorithm for the given network.

References

[1] E. Strohmaier, H. Simon, J. Dongarra, and M. Meuer, "Top500 supercomputing sites," Nov. 2016. [Online]. Available: http://www.top500.org/lists/2016/06

[2] Cisco Systems, Inc., "Virtualized multiservice data center (VMDC) 3.0 design guide," Dec. 2012. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/VMDC/3-0/DG/VMDC_3-0_DG.pdf

[3] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: A scalable fault-tolerant layer 2 data center network fabric," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 39–50. [Online]. Available: http://doi.acm.org/10.1145/1592568.1592575

[4] H. P. Hayden, "Voice flow control in integrated packet networks," Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Cambridge, MA, USA, Jun. 1981. [Online]. Available: http://hdl.handle.net/1721.1/15891

[5] D. P. Bertsekas and R. Gallager, *Data Networks*, 2nd ed. Prentice Hall, Jan. 1992.

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

[7] D. Nace, L. Nhat Doan, O. Klopfenstein, and A. Bashllari, "Max-min fairness in multi-commodity flows," *Computers and Operations Research*, vol. 35, no. 2, pp. 557–573, Feb. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.cor.2006.03.020

[8] S. R. Öhring, M. Ibel, S. K. Das, and K. Mohan J, "On generalized fat trees," in *Proceedings of the 9th International Parallel Processing Symposium*. Santa Barbara, California, USA: IEEE Computer Society, Apr. 25–28, 1995, pp. 37–44.

[9] E. Zahavi, "Fat-tree routing and node ordering providing contention free traffic for mpi global collectives," *Journal of Parallel and Distributed Computing*, vol. 72, no. 11, pp. 1423–1432, 2012.

[10] L. Dagum and R. Menon, "OpenMP: An industry-standard API for shared-memory programming," *IEEE Comput. Sci. Eng.*, vol. 5, no. 1, pp. 46–55, Jan. 1998. [Online]. Available: http://dx.doi.org/10.1109/99.660313

[11] X. Yuan, S. Mahapatra, M. Lang, and S. Pakin, "LFTI: A new performance metric for assessing interconnect designs for extreme-scale HPC systems," in *Proceedings of the IEEE 28th International Parallel and Distributed Processing Symposium*, ser. IPDPS '14. IEEE, 2014, pp. 273–282. [Online]. Available: http://doi.acm.org/10.1145/1555754.1555783

[12] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Trans. Comput.*, vol. 34, no. 10, pp. 892–901, Oct. 1985. [Online]. Available: http://dl.acm.org/citation.cfm?id=4492.4495

[13] Mellanox Technologies, "Deploying HPC Cluster with Mellanox InfiniBand Interconnect Solutions Reference Design Rev 1.3," http://www.mellanox.com/related-docs/solutions/deploying-hpc-cluster-with-mellanox-infiniband-interconnect-solutions-archive.pdf, Jan. 2017.

[14] F. Shahrokhi and D. W. Matula, "The maximum concurrent flow problem," *J. ACM*, vol. 37, no. 2, pp. 318–334, Apr. 1990. [Online]. Available: http://doi.acm.org/10.1145/77600.77620

[15] T. Hoefler, T. Schneider, and A. Lumsdaine, "Optimized routing for large-scale InfiniBand networks," in *17th Annual IEEE Symposium on High Performance Interconnects (HOTI 2009)*, Aug. 2009.

[16] N. Jain, A. Bhatele, X. Ni, N. J. Wright, and L. V. Kale, "Maximizing throughput on a dragonfly network," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 336–347. [Online]. Available: https://doi.org/10.1109/SC.2014.33

[17] N. Megiddo, "On the complexity of linear programming," *Advances in Economic theory*, pp. 225–268, Jan. 1987.

[18] IBM Corp., "IBM ILOG CPLEX Optimization Studio." [Online]. Available: http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud

[19] G. Rodríguez, C. Minkenberg, R. Beivide, R. P. Luijten, J. Labarta, and M. Valero, "Oblivious routing schemes in extended generalized fat tree networks," in *Proceedings of the 2009 IEEE International Conference on Cluster Computing, August 31 - September 4, 2009, New Orleans, Louisiana, USA*, 2009, pp. 1–8. [Online]. Available: http://dx.doi.org/10.1109/CLUSTR.2009.5289145