Global Optimization for Hash-based Splitting

Paolo Medagliani*, Jérémie Leguay*, Mohammed Abdullah*, Mathieu Leconte*, Stefano Paris*
*Mathematical and Algorithmic Sciences Lab, France Research Center, Huawei Technologies Co. Ltd.
20 Quai du Point du Jour, 92100 Boulogne-Billancourt, France

***Abstract*—Load-balancing and network optimization in SDN networks require efficient flow splitting during the path computation phase. The way flow splitting is typically implemented in switches is to map the output of an hash function computed on the headers of incoming flows to the content stored in a Ternary Content Addressable Memory (TCAM), a very efficient but scarce resource. Although a large TCAM budget means that the flow distribution can more accurately model a fractional ideal, the distribution of flow volume amongst the paths is constrained in reality to use only a limited number of TCAM rows. In this paper, we present a flow splitting algorithm that maximizes the total number of demands allocated in the network according to the TCAM size constraints and, at the same time, minimize the total routing cost. Although the problem is NP-hard, we show through simulations that we can achieve good approximations of the optimal solution in a reasonable amount of time.**

I. Introduction

Network operators often use multi-path routing to load-balance traffic for the sake of reliability and performance. Spreading traffic over multiple paths to a given destination can indeed improve reliability in case of failures and increase the overall network utilization. Multi-path routing is generally applied to large flow aggregates. To maintain the best routing configuration in the network, the solution of a multicommodity flow problem with fractional flows can provide a feasible split configuration and can be computed in polynomial time [1]. However, in practice, deployments and hardware constraints make the multi-path routing problem, also called the *flow splitting problem*, NP-hard and thus challenging to solve. This paper particularly aims at finding feasible split configurations that maximize the overall network throughput.

The flow splitting problem is twofold: a *routing* problem decides which paths to use, and a *load-balancing* problem allocates portion of the flow aggregate to each path. To process packets at very high speed, most routers and switches implement *hash-based splitting* to spread flows over multiple forwarding rules. These rules, often called *buckets*, are stored in the expensive and power-hungry TCAM memory of network equipments [2]. The number of rules allocated to a path determines the traffic portion allocated to it. As a consequence, due to memory limitation, only a restricted number of splits are possible. Increasing the number of rules improves the split accuracy with regards to an ideal split target, but leads to a dramatic increase of memory consumption. This trade-off has to be handled globally so that the number of accepted flow aggregates is maximized.

The ECMP (Equal-Cost Multi-Path) [3] extension to OSPF is the most commonly used technique to load-balance traffic. It decouples the two problems (i.e., routing and load-balancing). The routing protocol first provides a set of equal cost paths. Then, ECMP equally splits flow aggregates over them based on a hash computed over packet-header fields. Albeit simple to implement, it assumes regular topologies with a high number of equal cost paths to a given destination and an equal amount of available capacity over them. In more general settings, uneven flow splitting may improve network utilization, and the use of unequal cost multi-paths techniques can further increase performance. However, these two approaches solve a local problem and do not attempt to maximize the overall network throughput.

Several propositions have emerged in recent years to make data plane elements programmable so that the control logic can be offloaded to external units. To name a few: Forces [4], PCEP [5] and OpenFlow [6]. This control and data plane separation creates an opportunity to implement more efficient routing processes than classical protocols, since the controller can take real-time decisions at a (logically) centralized place using an accurate view of the network. Moreover, emerging Software Defined Networking (SDN) controllers have a tremendous computational power compared to legacy embedded devices. This encourages the development of smarter network control planes using cutting-edge optimization and parallel computing techniques.

To cope with the above challenges, we start with formulating the global multi-path routing problem with bucket constraints as a Mixed Integer Linear Programming (MILP) problem in order to get some insight on its limiting factors. We call this the ***Multicommodity Constrained Flow Splitting (MCFS) problem***. We prove that it is NP-Hard, and present a practically-efficient approximation algorithm. Our algorithm leverages on linear relaxations to exploit the underlying structure of the problem.

Finally, we perform a thorough numerical evaluation of the algorithm considering a realistic network topology and traffic matrix. Numerical results highlight the strength of the approach in terms of execution time, admitted traffic and routing cost.

II. Related Work

Flow splitting over multiple paths has been studied extensively. Many works advocate weighted traffic distribution and relaxation of the equal cost constraint [7, 8] to increase network utilization. However, the ECMP (Equal-Cost Multi-Path) [3] extension to OSPF is still the most commonly used technique to load balance traffic. Research now focuses on efficient and practical ways to implement unequal cost and uneven flow splitting. To address this problem, recent extensions of ECMP such as Weighted-Cost Multi-Path (WCMP) [9] routing or Niagara [10] have been proposed for uneven flow splitting over equal cost paths. However, these two approaches solve a local problem and do not aim to maximize the overall network throughput.

Other proposals have been done for packet level splitting. FLARE [11] splits flows spatially using TCP sequence numbers. DeTail [12] relies on backpressure and requires modification of switches and the TCP receiver. LocalFlow [13] measures the rate of each active flows, and runs a scheduling algorithm at every single switch. It groups flows per destination, computes a fractional bin packing solution and rounds the solution to reduce the number of flows that are split.

In contrast with previous proposals, our solution works with legacy routers and for flow aggregates. Furthermore, by considering the global optimization problem, it can be easily extended to optimize several performance metrics in addition to network throughput and routing cost.

III. Problem Formulation

This section introduces the flow splitting problem.

A. System Model

Let us consider a network topology represented as a graph $G = (V, E)$. Each vertex $v \in V$ corresponds to a network node, while each directed arc $(u, v) \in E$ represents a network link, whose bandwidth and cost per unit of traffic are denoted by b_{uv} and c_{uv}, respectively.

A set $\mathcal{D} = \{D_1, D_2, \ldots, D_K\}$ of K demands must be routed through the network. We model each demand k as a tuple $D_k = (s_k, t_k, d_k, N_p^k)$, where parameters s_k and t_k represent the source and destination nodes, whereas d_k is the amount of traffic to be routed. To use more efficiently network resources, each demand can be served using multiple paths. However, in order to limit jitter in the flow aggregate, the split of the traffic over multiple paths (called flow splitting) can be performed only at the ingress device using at most N_p^k paths.

Flow splitting is implemented in SDN switches using *hash functions*, since they can be executed at the switching speed. For every incoming packet of a flow, a hash is computed over a portion of the packet header to determine which forwarding rule to apply. The set of forwarding rules determines a split configuration, characterized by the portion of the incoming flow that is sent over different paths. Hash-based flow splitting may deviate from the optimal fractional split solving the multicommodity flow problem. However, in practice, providing a perfect representation of the optimal split with sufficient precision is impossible as the number of forwarding rules that can be inserted into a TCAM memory is bounded.

Figure 1 illustrates how hash-based splitting works. Packet forwarding is decomposed in two sets of rules, stored in the *forwarding* and *group* tables. In the forwarding table, a first rule maps an incoming packet to its splitting group. In turn, the group table contains entries that equally split flows over next hops using a hash function. Unequal splitting over different paths is achieved by repeating next-hops in the group table. Entries in both tables are called *buckets*. The goal of an SDN controller is to compute a feasible bucket allocation to be applied inside each switch. Figure 1 shows an example where demand D_1 is split over 3 sub-paths with ratios ($\frac{1}{6}$, $\frac{1}{3}$, $\frac{1}{2}$). At intermediate nodes of each sub-path (e.g., node .4) no split is allowed and only one bucket is required to store a forwarding rule. The forwarding and group tables of a switch u contain respectively τ_u^F and τ_u^G buckets, under the constraint that their sum does not exceed the TCAM size.

Forwarding rules are stored in TCAM of hardware switches. This type of memory can perform high-speed packet lookups in a single clock cycle. However, these components are power-hungry and expensive. Therefore, SDN switches are usually equipped with small TCAM that can store from a few hundred to thousands entries [2], denoted as τ_u buckets. This limitation reduces the number of possible splits and can lead to unfeasible allocation of demands. Table I summarizes all parameters of the MCFS problem.

Table I: Notation for the MCFS problem

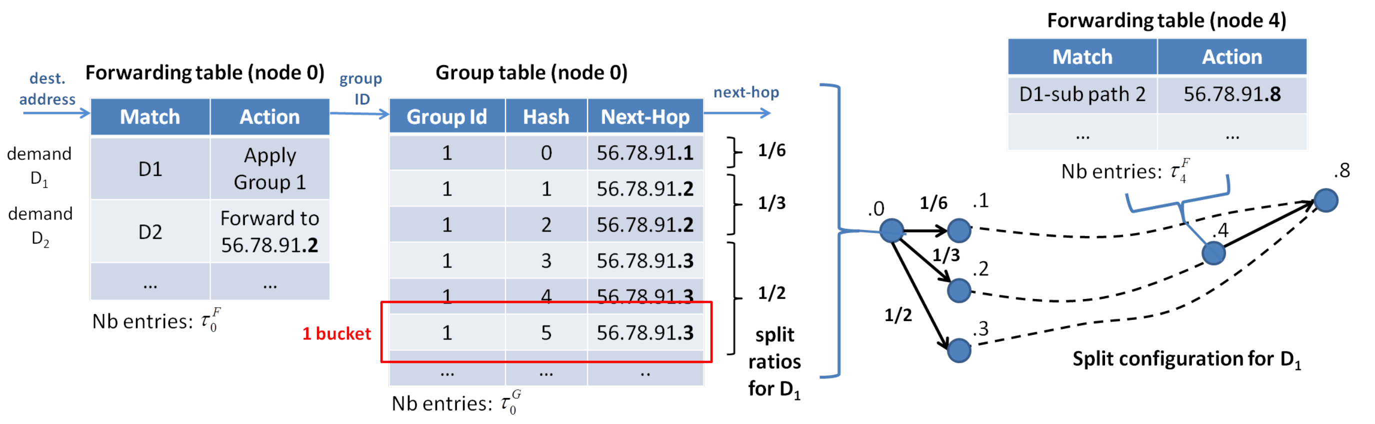
Parameters	Description
s_k	Source node for demand k
t_k	Destination node for demand k
d_k	Traffic generated by demand k
N_p^k	Maximum number of paths for demand k
c_{uv}	Cost of link (u, v)
b_{uv}	Capacity of link (u, v)
τ_u	TCAM size of node u

B. MINLP Formulation

The Multicommodity Constrained Flow Splitting (MCFS) problem that we address in this paper is to find as quickly as possible for a set of K demands a network-wide split configuration which maximizes network throughput. The primary objective is to accept as much traffic as possible, and the secondary objective is to minimize routing cost.

To this end, we define the following decision variables. Binary decision variables $x_{uv}^{kp} \in \{0, 1\}$ indicate the links that are selected to route demand k over one out of N_p^k paths $\left(p = \{1, \ldots, N_p^k\}\right)$. Specifically, $x_{uv}^{kp} = 1$ means that link (u, v) belongs to the p-th path used to route the demand k from its source to its destination. Real variables $y_{uv}^{kp} \in \mathbb{R}$ indicate the amount of traffic sent over edge (u, v) for demand k on its p-th path, whereas binary variables $h_{uv}^{kp} \in \{0, 1\}$ indicate if some traffic of demand k is actually routed over the edge (u, v) of the p-th path. In other words, $h_{uv}^{kp} = 1$ only if $x_{uv}^{kp} = 1$ and $y_{uv}^{kp} > 0$. As we will show, variables h_{uv}^{kp} are used to count the number of forwarding buckets τ_u^F at intermediate nodes, since due to the rounding of the flow caused by the bucket assignment at the source, some path might have zero flow.

Binary decision variables z_Q^k indicate the total number of buckets used for demand k ($z_Q^k = 1$ means that exactly Q



Forwarding table (node 0)

Match	Action
D1	Apply Group 1
D2	Forward to 56.78.91.**2**
...	...

Group table (node 0)

Group Id	Hash	Next-Hop
1	0	56.78.91.**1**
1	1	56.78.91.**2**
1	2	56.78.91.**2**
1	3	56.78.91.**3**
1	4	56.78.91.**3**
1	5	56.78.91.**3**
...	...	..

Forwarding table (node 4)

Match	Action
D1-sub path 2	56.78.91.**8**
...	...

Figure 1: Illustration of hash-based splitting. Forwarding and group table entries are shown for demands D_1 (split over 3 sub-paths) and D_2 (non split).

buckets are used for the splitting of demand k). Finally, given a value of Q buckets, the split of the bandwidth d_k requested by demand k is decided using integer variables $q_Q^{kp} \in \{0,\dots,Q\}$. Specifically, the split of the bandwidth d_k over the p-th path is realized according to the ratio $\frac{q_Q^{kp}}{Q}$, which is defined only when $z_Q^k = 1$ as illustrated in the following formulation.

The MCFS problem turns out to be a multi-commodity flow problem with additional constraints on the number of used paths and the portion of flow assigned to selected paths. In particular, MCFS can be formulated as follows:

$$\max \quad \zeta \sum_{k=1}^{K} \sum_{p=1}^{N_p^k} \sum_{(s_k;v)\in E} y_{s_k v}^{kp} - \sum_{k=1}^{K} \sum_{p=1}^{N_p^k} \sum_{(u,v)\in E} c_{uv} y_{uv}^{kp} \tag{1}$$

$$s.t. \quad \sum_{(s_k;v)\in E} x_{s_k v}^{kp} = 1, \qquad \sum_{(u;s_k)\in E} x_{u s_k}^{kp} = 0, \tag{2}$$

$$\sum_{(t_k;v)\in E} x_{t_k v}^{kp} = 0, \qquad \sum_{(u;t_k)\in E} x_{u t_k}^{kp} = 1, \tag{3}$$

$$\sum_{(u,v)\in E} x_{uv}^{kp} - \sum_{(v;u)\in E} x_{vu}^{kp} = 0, \qquad \forall u \in V \setminus \{s_k, t_k\}, \tag{4}$$

$$\sum_{k=1}^{K} \sum_{p=1}^{N_p^k} y_{uv}^{kp} \le b_{uv}, \tag{5}$$

$$y_{uv}^{kp} = d_k x_{uv}^{kp} \sum_{Q=1}^{\theta} \frac{q_Q^{kp}}{Q}, \tag{6}$$

$$\sum_{p=1}^{N_p^k} q_Q^{kp} = Q z_Q^k, \qquad \sum_{Q=1}^{\theta} z_Q^k = 1, \tag{7}$$

$$\sum_{\substack{k\in K:\\ s_k=u}} \sum_{Q=1}^{\theta} Q z_Q^k + \sum_{\substack{k\in K:\\ s_k\neq u}} \sum_{p=1}^{N_p^k} \sum_{(u,v)\in E} h_{uv}^{kp} \le \tau_u, \tag{8}$$

$$h_{uv}^{kp} \le x_{uv}^{kp}, \qquad h_{uv}^{kp} \le q_Q^{kp}, \qquad h_{uv}^{kp} \ge \frac{y_{uv}^{kp}}{d_k}, \tag{9}$$

$$x_{uv}^{kp}, h_{uv}^{kp} \in \{0,1\}, \qquad y_{uv}^{kp} \in \mathbb{R}, \tag{10}$$

$$q_Q^{kp} \in \{0,\dots,Q\}, \qquad z_Q^k \in \{0,1\}, \tag{11}$$

where ζ is some sufficiently large constant and $\theta = \max_u\{\tau_u\}$ is the maximum number of rules that can be stored in a TCAM. When unspecified, the range of the variables is as follows:

$$k \in K, \qquad u \in V, \qquad p \in \{1,\dots,N_p^k\}, \qquad (u,v) \in E.$$

The objective function (1) in the above formulation maximizes the bandwidth accepted in the network, while the second term minimizes the routing cost among all feasible solutions with maximum accepted bandwidth. To this end, it suffices to set $\zeta = |K||V| \max_{k\in K} N_p^k \cdot \max_{(u,v)\in E} c_{uv}$, so that increasing the amount of accepted bandwidth will always be prioritized over reducing the routing cost.

Constraints (2) to (4) are flow conservation constraints that permit to route the traffic of each demand from its source s_k to its destination t_k over a set N_p^k paths. They also prevent cycles forming at the source or target.

Constraints (5) guarantee that the amount of traffic transmitted on each directed link (u,v) does not exceed its capacity b_{uv}. Constraints (6) compute the fractional flow y_{uv}^{kp} (i.e., the bandwidth assigned to path p) as the product of the path variable x_{uv}^{kp}, which indicates whether path p of demand k uses edge (u,v), and the fraction of the demand k that is allocated to path p, namely $d_k \sum_{Q=1}^{\theta} \frac{q_Q^{kp}}{Q}$. These inequalities are nonlinear, since they involve the product of variables x_{uv}^{kp} and q_Q^{kp}. However, we will show below how they can be replaced by a set of linear constraints in order to make the overall formulation ILP.

The set of constraints (7) restricts the bucket configuration by defining how many buckets are used for each path p assigned to a demand k. In particular, the second constraint in (7) selects only a single total number of buckets Q to be used for demand k among the θ possibilities at most, and the first constraint guaranties that this number Q is split among the paths used by demand k. Because only one value of Q is active at a time, the variables q_Q^{kp} can be non-zero only for one value of Q in (7). Therefore, the expression $\sum_{Q=1}^{\theta} \frac{q_Q^{k,p}}{Q}$ in (6) gives the fraction of demand k which is allocated to path p.

Constraints (8) guarantee that the forwarding rules defined either for the splitting of a demand originated at node u, or the paths that are passing through node u, do not exceed the TCAM capacity of u. Note that for counting how many buckets need to be reserved for forwarding rules we cannot sum over all demands the variables x_{uv}^{kp} that indicate the activation of an outgoing link for a path p. Indeed, the optimal solution might select a path with zero flow on it (i.e., $x_{uv}^{kp} = 1$ and $y_{uv}^{kp} = 0$). For this reason, we use binary variables h_{uv}^{kp} in the second summation of (8) and we further defined constraints (9) to set variable $h_{uv}^{kp} = 1$ only if some traffic of demand k is routed to the destination through path p on the link (u, v). Finally, constraints (11) define the domains of our decision variables.

We observe that the above formulation is for a network with directed links. However, we could easily recover a formulation with undirected links by interpreting (u, v) as being the opposite direction of (v, u) and replacing y_{uv}^{kp} by $y_{uv}^{kp} + y_{vu}^{kp}$ in constraints (5).

After having introduced the Mixed Integer Non Linear Programming (MINLP) formulation for the MCFS problem, we now point out that constraint (6) can be linearized by replacing it with the following sets of linear constraints:

$$\begin{aligned} &y_{uv}^{kp} \leq d_k x_{uv}^{k,p}, && y_{uv}^{kp} \leq d_k \textstyle\sum_{Q=1}^{\theta} \frac{q_Q^{kp}}{Q}, \\ &y_{uv}^{kp} \geq 0, && y_{uv}^{kp} \geq d_k \textstyle\sum_{Q=1}^{\theta} \frac{q_Q^{kp}}{Q} - d_k(1 - x_{uv}^{kp}) \end{aligned}$$

Unfortunately, even the derived MILP problem is NP-hard. In fact, it is hard even to approximate within practically useful bounds.

Proposition 1. *MCFS cannot be approximated to within constant factors unless $P = NP$.*

Proof. By setting $b_{uv} = 1$ for all $uv \in E$, $N_p^k = d_k = 1$ for all $k \in K$, and giving large enough bucket capacity at each node (e.g., K is sufficient), we see that the *maximum edge-disjoint paths* (EDP) problem reduces to MCFS. For both directed and undirected graphs, EDP is NP-hard to approximate to within constant factors [14] $\square$

IV. Our approach

To solve the MCFS problem we have developed a meta-heuristic referred to as *Iterative Relaxation with Scaling and Rounding* (IRSR).

A. Iterative Relaxation with Scaling and Rounding (IRSR)

As the resolution of the MILP is unfeasible from a computational point of view, we consider the linear relaxation by removing path and bucket constraints. Solving the resulting LP, which we call the *Relaxed LP* (RLP), will give a set of allocations for a subset of the demands. The drawback of this approach is that a demand D_k may end up employing more than N_p^k paths, and since the volume of flow on each path is fractional, it will also generally not be consistent with bucket constraints placed in the original MILP. Thus, these violations need to be corrected to get a feasible solution.

Suppose demand D_k was admitted in the solution of RLP and suppose that $\hat{N}_p^k > N_p^k$ paths were allocated to it. Consider the $\hat{N}_p^k - N_p^k$ paths with the least amount of D_k's bandwidth allocated to them. We delete them and uniformly redistribute their bandwidth onto the remaining N_p^k paths. Following this *path redistribution*, we proceed with *bucket rounding* to attempt finding a feasible solution within a buckets budget given to each demand by a *bucket oracle*.

Of course, in performing the above, it may be that edge capacities get violated. We deal with this in two ways: firstly, before solving RLP, we multiply edge capacities throughout the network by $(1-\alpha)$, where $\alpha \in [0, 1)$ is a parameter. Constricting capacities in this way may force the RLP to choose more expensive paths, but it gives us leeway in performing the subsequent redistribution and rounding phases (which are, of course, performed with the actual edge capacities). In fact, since we don't know *a priori* what a good value of α is, we attempt a set of them and choose the one that gives the best outcome in terms of allocated bandwidth after the bucket rounding phase. This can be done in parallel as SDN platforms have large computational power.

The second way we deal with capacity violations is through iteration: demands which get rejected for violating capacity constraints become the input in another round of the algorithm. We keep iterating until there is no improvement in the number of allocations or all the demands are allocated.

We detail this process below. It should be noted that we solve the RLP using Column Generation (CG) [15], which is an approach that is often effective in coping with LPs with a large number of variables. However, CG is not necessary, and any algorithm for solving LPs may be used instead.

(1) Scaling Scale edge capacities by $(1 - \alpha)$.
(2) Relaxed LP (RLP) Remove path number and bucket constraints, relax integer variables and solve.
(3) Intermediary buckets For each (fully or partially) allocated demand D_k, remove one bucket entry from each node on each path in the allocation (routing a path through a TCAM costs one entry).
(4) Bucket oracle Use the oracle on the resulting network and demand set to get a bucket budget $\mathcal{B}$ for each demand.
(5) Path redistribution For each (fully or partially) allocated D_k, if $\hat{N}_p^k > N_p^k$ paths were allocated to it, then uniformly redistribute $\hat{N}_p^k - N_p^k$ excess paths on the most heavily allocated N_p^k.
(6) Bucket rounding For each allocated D_k, for $b = 1, \ldots, \mathcal{B}_k$ redistribute the flow amongst the allocated (at most N_p^k) sub-paths of D_k in units of d_k/b so as to reproduce the former allocation as faithfully as possible.
(7) Filtering Check which demands now violate edge capacity constraints. Perform this sequentially, always checking against the network residual to the demands already allocated.
(8) Update residual buckets Update bucket utilization at

both source nodes and intermediate nodes for the demands that were allocated during this iteration.

(9) Iteration All demands rejected in the previous phase become the input to a reiteration of this algorithm. allocated.

It should be noted that bucket rounding is a non-trivial problem. Our approach is as follows: for each allocated D_k we allocate buckets one at a time, and in doing so, build up a per-path flow profile for D_k based on these buckets. If we are going to add exactly b buckets in total, then each bucket adds d_k/b to the flow of the path it is assigned to. The bucket is placed randomly per a probability distribution derived from the flow profile induced by the current bucket allocation, and that given by RLP solution. A larger difference of these profiles on a particular path makes it more likely a bucket will be assigned to that path.

B. Bucket Oracle

We use the following *oracle-based* procedure for bucket allocations. The goal here is to decide an initial maximum number of buckets for each demand so as not to exceed the TCAM size and to keep flexibility for the subsequent rounding step, so that we can split with reasonable accuracy the demands which require it. In this work, splits are only performed at source nodes of the demands. Intermediate nodes on a path only need one bucket to be able to forward packets.

To leave space for demands on intermediate nodes, we a priori reduce the TCAM capacity τ_u. In the rest of the paper, we consider a capacity reduction of 50%.

To make sure that this remaining budget of buckets is not exceeded when performing splits, we split this budget into individual bucket budgets $\mathcal{B}_k$ for each demand originating at node u, such that $\sum_{k\in K_u} \mathcal{B}_k = \widetilde{\tau}_u$, where $K_u := \{k \in K : s_k = u\}$. In order to set the values of $\mathcal{B}_k$, we leverage mainly on the intuition that large demands require more buckets to achieve a reasonable split accuracy. Therefore, we will allocate individual bucket budgets $\mathcal{B}_k$ in order to maximize a fairness criterion between demands which gives higher weight to larger demands. In particular we choose to maximize proportional fairness among demands in our bucket allocation, i.e., we set $\{\mathcal{B}_k\}_{k\in K_u}$ such that it approximately solves

$$\max_{\substack{\mathcal{B}_k\in\mathbb{N},\ \forall k\in K_u\\ \sum_{k\in K_u}\mathcal{B}_k\le\widetilde{\tau}_u}} \sum_{k\in K_u} d_k \log \mathcal{B}_k,$$

Optimizing the above fairness criterion can be approximately done in a simple way: we can define a Lagrangian multiplier λ for the constraint $\sum_{k\in K_u} \mathcal{B}_k \le \widetilde{\tau}_u$, iteratively assign buckets based on λ using $\mathcal{B}_k = \max\left\{1, \frac{d_k}{\lambda}\right\}$ and adjust λ using dichotomy search to converge towards using up the whole bucket budget.

Subsequently, after each demand is processed, the oracle is called again, operating on the basis of the remaining demands and the remaining (unused) buckets.

V. Performance Evaluation

This section illustrates the performance of our algorithm to solve the MCFS problem in a realistic network scenario.

A. Experimental Methodology

We used a network topology generated with the random-connection model [16]. To mimic real world topologies like GEANT, we considered 60 nodes, a nodal degree of 3, link capacity of 40 GB, and link costs according to a uniform distribution in $[1;10]$. We consider several number of demands $k \in \{600;2100\}$ with random source and destination pair. The requested bandwidth d_k is generated according to a Zipf distribution of exponent $a = 0.5$. The maximum size of the demands is set to 48 GB, to depict the presence of elephant flows transferred in the network. The average demand size, that depends on the number of generated demands, ranges between 5.3 GB and 2 GB. As SDN switches are usually equipped with small TCAM to store from a few hundred to thousands entries, we considered a TCAM size of 3000 for each node.

We compare IRSR with upper and lower bounds for the total amount of accepted traffic. The upper bound consists of RLP, the linear relaxation of the problem without constraints and splittable flows. The lower bound is obtained by solving the unconstrained multi-commodity flow problem with integer flows with column generation and randomized rounding (*Integer Linear Problem without Flow Splitting*, ILP No FS). The third considered benchmark is a greedy Successive Shortest Path (SSP) algorithm without flow splitting. In this case, for each demand we compute the shortest path on the residual graph: either the demand is accepted and the network capacity updated or it is rejected if there is no spare capacity.

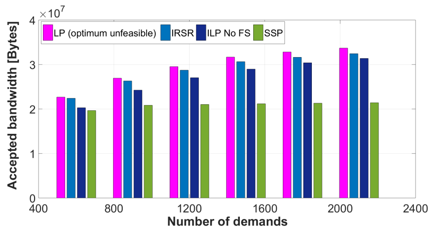
As performance metrics, we consider the *proportion of accepted traffic* (i.e., the fraction of traffic successfully admitted with respect to the input load), the *routing cost per unit of accepted traffic* (i.e., the ratio between the total accepted bandwidth and the cost for routing the accepted traffic [1]), and the *execution time* of each algorithm to compute a solution. Each points depicted in the plots illustrated in this section is the average of 10 independent experiments. As the upper bound gives an unfeasible solution from the point of view of flow splitting, we will consider it only for the accepted bandwidth performance.

Results of the IRSR approach have been obtained by considering different factors of scaling $\alpha = \{0.005, 0.01\}$. As this scaling operation can be easily parallelized, the execution time for IRSR will be evaluated based on a parallel implementation. We implemented the algorithms in C++ and ran simulations on a Linux Server with 40 Intel(R) Xeon(R) CPU E5-2690 v2 3.00GHz cores with 25MB cache and 192 GB of RAM.

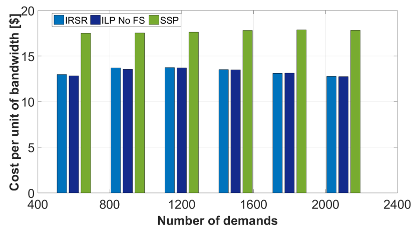
B. Performance Analysis

Figure 2 shows the total accepted bandwidth, the average routing cost per unit of accepted traffic and the execution time of the proposed algorithms as a function of the number of demands (up to 2100). We first observe that IRSR can allocate almost all the demands offered to the network when we

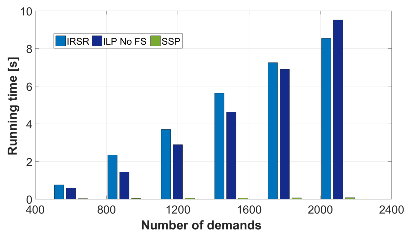
[1]We point out that the *cost* of a demand on a link through which it is routed is defined as the product of the bandwidth of the demand and the cost per unit of bandwidth of the link.



(a) Accepted bandwidth



(b) Routing cost



(c) Execution time

Figure 2: Accepted bandwidth (a) and execution time (b), and (c) average routing cost as a function of the number of demands.

compare the total accepted traffic with RLP. When the number of demands is low (corresponding to an average demand size of 5.3 GB) IRSR benefits from the possibility of splitting to keep the number of rejected demands low. Vice versa, ILP No FS and SSP reject some demands because they cannot split elephant flows. When the number of connections increases, IRSR still performs very close to RLP. ILP No FS still has good performance because instead of big demands it allocates smaller flows, which have lower size and can fit in the network without being split. Instead, as SSP is based on shortest path routines, it quickly experiences performance degradation as it always tries to use the cheapest path.

Considering the cost per unit of bandwidth, due to the underlying CG routine, there is no difference between IRSR and ILP No FS, as the cheapest links are used to allocate the demands. After $K = 1200$, despite big demands can be rejected, the cost per unit of bandwidth decreases as cheapest links are packed allocating the smallest flows. SSP instead, uses more expensive links as soon as it introduces bottlenecks on the cheapest ones. For such a reason, it turns out that it always has a higher routing cost even in the case where it has bandwidth acceptance performance comparable to ILP No FS.

Finally, considering the execution time, IRSR and ILP No FS have very similar performance. The largest part of the time is consumed by running the CG algorithm. Despite IRSR runs on the top a rounding routine that is not present in ILP No FS, the execution time does not increase significantly.

VI. Concluding remarks

We presented a flow allocation problem for networks with forwarding rules held in TCAM, whose scarcity creates a trade-off between the precision of the solution allocated into the network and the number of demands that can be accepted. Finding an optimal path allocation that takes into account the use of buckets is NP-hard even to approximate within constant factors. Thus, we proposed an approach for maximizing the accepted bandwidth while respecting capacity and bucket constraints. As far as we are aware, this paper is the first work that takes into account global path allocation and TCAM-based flow splitting at the same time.

We evaluated through simulations the performance of the proposed algorithm in terms of accepted bandwidth, routing cost and execution time. Results show that IRSR finds feasible split configurations which maximize the accepted throughput and minimize the routing cost. In the presence of elephant flows whose size is comparable to link capacity, our approach allows to achieve accepted traffic performance which is very close to that of the relaxed problem.

Extensions of this work will consider the interaction between a global optimization achieved periodically, such as the one presented in this work, and local rules executed in real-time to correct the allocation of individual flows to buckets. The assumption behind hash-based splitting is that individual flows are several orders of magnitude smaller than the aggregate. However, there might be situations where a few individual flows are large and local rules may be required to limit the deviation from the target split ratio decided globally.

References

[1] T. H. Cormen, *Introduction to algorithms*. MIT press, 2009.

[2] K. Kannan and S. Banerjee, "Compact TCAM: Flow entry compaction in TCAM for power aware SDN," in *Distributed Computing and Networking*. Springer, 2013, pp. 439–444.

[3] D. Thaler, "Multipath issues in unicast and multicast next-hop selection. internet engineering task force: RFC 2991," 2000.

[4] A. Doria, "Forwarding and Control Element Separation (ForCES) Protocol Specification," RFC 5810, IETF (March 2010).

[5] J. P. Vasseur, "Path Computation Element (PCE) Communication Protocol (PCEP)," RFC 5440, IETF (March 2009).

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, Mar. 2008.

[7] S. Prabhavat, H. Nishiyama, N. Ansari, and N. Kato, "On load distribution over multipath networks," *Communications Surveys & Tutorials, IEEE*, vol. 14, no. 3, pp. 662–680, 2012.

[8] G. M. Lee and J. Choi, "A survey of multipath routing for traffic engineering," *Information and Communications University, Korea*, 2002.

[9] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers," in *ACM EuroSys*, 2014.

[10] N. Kang, J. Reumann, A. Shraer, and J. Rexford, "Efficient Traffic Splitting on SDN switches," Tech. Rep., 2015.

[11] S. Sinha, S. Kandula, and D. Katabi, "Harnessing tcps burstiness using flowlet switching," *Proceedings of ACM Hot-Nets*, 2004.

[12] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "Detail: reducing the flow completion time tail in datacenter networks," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 139–150, 2012.

[13] S. Sen, D. Shue, S. Ihm, and M. J. Freedman, "Scalable, optimal flow routing in datacenters via local link balancing," in *ACM CoNEXT*, 2013.

[14] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis, "Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems," in *ACM STOC*, 1999.

[15] G. Desaulniers, J. Desrosiers, and M. Solomon, Eds., *Column generation*, ser. GERAD 25th anniversary series. New York: Springer, 2005.

[16] M. D. Penrose, "On a continuum percolation model," *Advances in applied probability*, pp. 536–556, 1991.