Multi-Path Alpha-Fair Resource Allocation at Scale in Distributed Software Defined Networks

Zaid Allybokus, Konstantin Avrachenkov, Jérémie Leguay, Lorenzo Maggi

Multi-Path Alpha-Fair Resource Allocation at Scale in Distributed Software Defined Networks*

Zaid Allybokus*†, Konstantin Avrachenkov*, Jérémie Leguay†, Lorenzo Maggi†
†Huawei Technologies, France Research Center
{zaid.allybokus, jeremie.leguay, lorenzo.maggi}@huawei.com
*INRIA Sophia Antipolis
konstantin.avratchenkov@inria.fr

Abstract

The performance of computer networks relies on how bandwidth is shared among different flows. Fair resource allocation is a challenging problem particularly when the flows evolve over time. To address this issue, bandwidth sharing techniques that quickly react to the traffic fluctuations are of interest, especially in large scale settings with hundreds of nodes and thousands of flows. In this context, we propose a distributed algorithm based on the Alternating Direction Method of Multipliers (ADMM) that tackles the multi-path fair resource allocation problem in a distributed SDN control architecture. Our ADMM-based algorithm continuously generates a sequence of resource allocation solutions converging to the fair allocation while always remaining feasible, a property that standard primal-dual decomposition methods often lack. Thanks to the distribution of all computer intensive operations, we demonstrate that we can handle large instances at scale.

Index Terms

Software-Defined Networks; Multi-path Resource Allocation; Alpha-Fairness; Alternating Direction Method of Multipliers; Distributed SDN Control Plane; Distributed Algorithms.

1 Introduction

Software Defined Networking (SDN) technologies are radically transforming network architectures by offloading the control plane (e.g., routing, resource allocation) to powerful remote platforms that gather and keep a local or global view of the network status in real-time and push consistent configuration updates to the network equipment. The computation power of SDN controllers fosters the development of a new generation of control plane architecture that uses compute-intensive operations. Initial design of SDN architectures [37] had envisioned the use of one central controller. However, for obvious scalability and resiliency reasons, the industry has quickly realized that the SDN control plane needs to be partially distributed in large network scenarios [20]. Hence, although *logically* centralized, in practice the control plane may consist of multiple controllers each in charge of a SDN domain of the network and operating together, in a *flat* [35] or *hierarchical* [17] architecture. In Fig. 1, an example of flat architecture is illustrated. In hierarchical architectures, a master controller is placed on top of domain sub-controllers and keeps global view of the network and the sub-controllers to handle the message-passing protocols.

In this paper, we study the problem of computing a globally *fair* (in the sense of α-fairness defined by Mo and Walrand in [26], see Section 3) multi-path resource allocation in a distributed SDN scenario, where the control plane is distributed over several domain controllers. In this context, flows transiting in the network typically correspond to traffic aggregates of a customer or a class of customers. The traffic aggregates can be carried on one path or be split through many different paths connecting a source to a destination. Indeed, the multi-path traffic engineering can be preferred over single-paths to ensure routing robustness, low latency, or good load balancing for better performance, as explained in [21].

*This is the authors' edited copy of the article "Multi-Path Alpha-Fair Resource Allocation at Scale in Distributed Software Defined Networks" published in IEEE JSAC v.36(12), pp.2655-2666, December 2018.

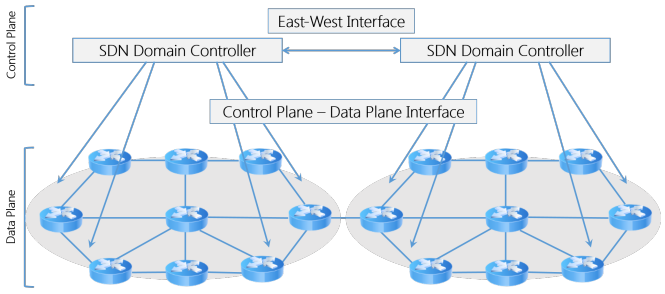

Fig. 1: Distributed SDN architecture.

In this paper, we study the problem of computing a globally *fair* (in the sense of α-fairness defined by Mo and Walrand in [26], see Section 3) multi-path resource allocation in a distributed SDN scenario, where the control plane is distributed over several domain controllers. In this context, flows transiting in the network typically correspond to traffic aggregates of a customer or a class of customers. We consider the traffic engineering use case where the size of flows evolves over time and the bandwidth reserved to each of them has to be quickly adjusted towards the novel fair solution.

In distributed SDN architectures [13][31], each controller has full information about its own domain. Although for fault-tolerance reasons, they could be composed of master and slave agents that act as a single entity [28], in this paper we only consider the distribution of the SDN control plane for scalability purposes: the control plane consists of multiple controllers each in charge of a single domain and a reduced portion of the global workload and operate together as a logically centralized controller. Therefore, each controller can communicate with adjacent peer controllers and/or with a central, upper-layer controller entity. However, exchanges between controllers are expensive in terms of communication delay and overhead [31]. This technological limitation translates directly into an algorithmic constraint: distributed algorithms for SDN have a limited budget in terms of the number of iterations to reach convergence, i.e. a near-optimal solution.

In distributed SDN architectures, as depicted in Fig. 1, each controller has full information about its own domain. Moreover, it can communicate with adjacent peer controllers and/or with a central, upper-layer controller entity. However, exchanges between controllers are expensive in terms of communication delay and overhead [31]. This technological limitation translates directly into an algorithmic constraint: distributed algorithms for SDN have a limited budget in terms number of iterations to reach convergence.

A second crucial property for any distributed algorithm for SDN is responsiveness. In fact, the network state may be affected by abrupt changes, e.g., flow size variation, flow arrival/departure, link/node congestion. In this case, convergence for the previous network state may not even be attained when a change occurs in the system. For this reason, it is often preferable to have a quick access to a good quality solution rather than a provably asymptotically optimal solution with poor convergence rate. Hence, it is crucial that the resource allocation computed by a distributed algorithm is *feasible, thus implementable*, at any iteration.

Also, modern SDN controllers [11] rely on grid computing technologies such as Akka [1] or Hazelcast [32], respectively for the two major open source SDN controllers OpenDayLight [3] and ONOS [2]. As a consequence, massively parallelizable algorithms for SDN should be preferable as more adapted and better likely to tackle scalability issues.

Also, modern SDN controllers rely on grid computing technologies such as Akka or Hazelcast [32] for compute intensive tasks. As a consequence, any distributed algorithm for SDN should be massively parallelized.

To recap, we identify two main requirements for a distributed algorithm for fair resource allocation, namely *i)* converging to a "good" fair solution in a small number of iterations and *ii)* producing *feasible* solutions at all iterations.

To recap, we identify three main requirements for a distributed algorithm for fair resource allocation, namely *i)* converging to a "good" fair solution in a small number of iterations, *ii)* producing *feasible* solutions at all iterations and *iii)* being massively parallelized.

We claim that none of the current methods that allocate resources in an SDN scenario is able to achieve the three aforementioned goals at the same time. Local mechanisms such as Auto-Bandwidth [29] have been proposed to greedily and distributedly adjust the allocated bandwidth to support time-varying IP traffic in *Multi Protocol Label Switching* (MPLS) networks. Auto-Bandwidth successfully tackles goals *ii)* and *iii)*, but not *i)*, as it neither ensures fairness nor optimizes resources globally. Also, classic primal-dual algorithms have been proposed to solve the α-fair resource allocation problem in distributed SDN scenarios, as in [25]. However, primal-dual algorithms are known to fail at providing feasible solutions at any iteration step, thus they fail at achieving goal *ii)*.

Recently in the optimization research community, the *Alternating Direction Method of Multipliers* (ADMM) [12] has captured the attention for its separability and fast convergence properties. We claim that ADMM offers new and yet unexploited possibilities to tackle concurrently the goals *i)*, *ii)* and *iii)*. Indeed, in this paper we show how ADMM serves our purposes, by allowing all controllers to handle their own domains simultaneously, while still converging to a global optimum in the fashion of a general distributed consensus problem.

Main contributions: We develop Fast Distributed ADMM (FD-ADMM, Algorithm 1) for the multi-path α-fair resource allocation problem over a distributed SDN control plane. It iteratively produces resource allocations that converge to the α-fair optimal allocation. Heavy computations, requiring projections on polytopes, can be massively parallelized on a link-by-link basis (Algorithm 1, line 7) in each domain. This yields a convergence rate (in terms of iteration count) that does not depend on the partitioning of the network into domains, that can therefore be done *independently*.

We showed [6] that our FD-ADMM algorithm can function in real-time, as *i)* close-to-optimal solutions are available since the *very first* iterations and *ii)* feasible allocations are available at *all* iterations (Proposition 1), a property that standard primal-dual decomposition methods generally lack. This permits to adjust within very short time the bandwidth of flows that evolve quickly and need immediate response.

Moreover, we addressed in [5] the problem of the penalty parameter initialization in ADMM that is well-known to highly condition the convergence speed of the algorithm, by proposing a tuning based on a lower bound for the α-fair resource allocation problem that we derived.

This article is a synthesis of the two mentioned works [5] and [6] with three novel contributions:

- We extend the model presented in [6] to the setting of *multi-path* routing. Now, each flow can carry their traffic along several paths instead of only one.
- We also show that the algorithm can integrate a switching cost in order to address the related relevant problem of limiting flow reconfigurations. Although we expect FD-ADMM to continuously provide feasible iterates that respond to traffic variations in real-time, it is practically infeasible to reconfigure all the flows too often without harming the network stability and the overhead [30]. Therefore, we use ideas of sparse optimization [7] to make FD-ADMM sensitive to the cost of a reconfiguration of the bandwidth along a path, and hence operate a trade-off between fairness and switching cost.

- Finally, we evaluate numerically the performance of FD-ADMM over large scale instances made of Barabasi-Albert and Fat tree networks of up to hundreds of nodes and thousands of links, requests and paths, in a setting where several SDN domain controllers operate in parallel.
- To the best of our knowledge, we were the first to show how ADMM can help designing real-time distributed algorithms for computing α-fair resource allocations in distributed settings. We were also the first to address the penalty parameter tuning of ADMM in this situation. Those results are all extended to the multi-path setting in the present article.

The remainder of this paper is organized as follows. Section 2 surveys the related work around the fair resource allocation problem. Section 3 formulates the multi-path α-fair resource allocation problem and recalls the key ideas of ADMM. Section 4 introduces FD-ADMM, our distributed ADMM-based algorithm that benefits from the distribution of SDN controllers over multiple domains, that relies on a reformulation of the problem in the fashion of a general distributed consensus problem. Section 5 extends the model to account for the introduction of the switching cost. Section 6 provides large scale simulations that validate our approach and finally, Section 7 concludes the paper.

2 Related work

The concept of fair resource allocation has been a central topic in networking. Particularly, *max-min* fairness has been the classic resource sharing principle [10] and has been studied extensively. The concept of *proportional fairness* and its weighted variants were introduced in [19]. Later, a spectrum of fairness metrics including the two former ones was introduced in [26] as the family of α-fair utility functions.

Some early notable works on max-min fairness include [14], where the authors propose an asynchronous distributed algorithm that communicates explicitly with the sources and pays some overhead in exchange for more robustness and faster convergence. Later in [33], a distributed algorithm is defined for the weighted variant of max-min fair resource allocation problem in MPLS networks, based on the well-known property that an allocation is max-min fair if and only if each *Label-Switched Path* (LSP) either admits a *bottleneck link* amongst its used links or meets its maximal bandwidth requirement (see Definition 4 there of a bottleneck link). The problem of Network Utility Maximization (NUM) was also addressed with standard decomposition methods that could give efficient and very simple algorithms based on gradient ascent schemes performing their update rules in parallel. In this context, Voice [38], then McCormick et al. [25], tackle the α-fair resource allocation problem with a gradient descent applied to the dual of the problem.

In the works described above, no mention is made on the potential (in fact, systematic) feasibility violation of the sequences generated by those algorithms, which is a crucial matter in distributed SDN settings. Regarding this topic, the authors of [22] employ damping techniques to avoid transient infeasibility while reaching the max-min fair point, but cannot guarantee feasibility at all times, especially in dynamic settings. Also motivated by this, more recently the authors of [36] provide a feasibility preserving version of Kelly's methodology in [19]. Their algorithm introduces a slave that gives at each (master) iteration an optimal solution of a weighted proportionally fair resource allocation problem that is explicitly addressed in only the two cases of polymatroidal and flow aggregating networks. In fact, our paper contributes to this problem by proposing an efficient *real-time* version of the slave process, for any topology, preserving feasibility at each (slave) iteration. Amongst approximative approaches, one can quote the very recent work [24] where a multiplicative approximation for $\alpha \neq 1$ and additive approximation for $\alpha = 1$ is provably obtained in poly-logarithmic time in the problem parameters. Moreover, starting from any point, the algorithm reaches feasibility within poly-logarithmic time and remains feasible forever after. The algorithm described in our paper solves the problem optimally and reaches feasibility as from the first iteration from any starting point.

The work around ADMM is currently flourishing. The $\mathcal{O}(\frac{1}{n})$ best known convergence rate of ADMM [18] failed to explain its empirical fast convergence until very recently in [16], where global linear convergence rates are established in four scenarios of the strongly convex case. ADMM is also well-known for its performance that highly depends on the parameter tuning, namely, the penalty parameter λ in the augmented Lagrangian formulation (see Section 3.3 below). An effective use of this class of algorithms cannot be decoupled from an accurate parameter tuning, as convergence can be extremely slow otherwise. Thus, in the same paper [16], the authors provide a linear convergence proof that yields a convergence rate in a closed form that can be optimized with respect to the problem parameters. Therefore, thanks to these works, we derived in [6] an adaptive tuning of ADMM, and in [5] a satisfactory initialization of the penalty parameter for the α-fair resource allocation problem. Several papers use the distributivity of ADMM to design efficient distributed algorithms solving consensus formulations for e.g. model predictive control [27] and resource allocation in wireless virtual networks [23] but do not address this fundamental detail.

3 Multi-path fair resource allocation problem

In this section, we define the multi-path α-fair resource allocation problem and formulate it as a centralized convex optimization problem. We also introduce the basic principles of ADMM and then present the centralized algorithm it yields. We will see in what this centralized version does not fit our distributed setting, which motivates the more detailed decomposition of this article.

3.1 Presentation

We define the network as the set of its links $\mathcal{J}$. Each network link $j \in \mathcal{J}$ has a total capacity of $c_j \in \mathbf{R}_+$.

Let $\mathcal{R}$ be a set of *connection requests*, or shortly, *requests*, over the network. We regard each request r as a communication demand between a source node and a destination node of the network, provided with a set P_r of several paths connecting the two end nodes. We assume the paths are pre-computed once and for all and do not change. In this article, only the

bandwidth allocation along fixed paths is considered and as such, the path computation dimension of the problem goes beyond the scope of this study. We model the paths as subsets of $\mathcal{J}$, that we assume form a physical path on the network's real topology. Thus, in the multi-path setting, the link capacities will be shared by the different requests in $\mathcal{R}$ with a bandwidth allocation along one or many of their paths. Let $\mathcal{P} \coloneqq \cup_r P_r$ be the set of all established paths. For each path $p \in \mathcal{P}$, we define J_p as the set of links that form p. For example, if the path p contains the set of links j_1, j_2 and j_3, then $J_p = \{j_1, j_2, j_3\}$.

The aim is to compute an optimal bandwidth allocation with respect to the α-fairness metric defined in [26], that we report below.

Definition 1 ((w,α)-fairness, [26])**.** Let $\alpha \in \mathbf{R}_+$. Let $F \subset \mathbf{R}_+^n$ be a non-empty feasible set not reduced to $\{\mathbf{0}\}$. Let $\boldsymbol{w} \in \mathbf{R}_+^n$ and $\boldsymbol{y}^* \in F$. We say that $\boldsymbol{y}^*$ is $(\boldsymbol{w},\alpha)$-*fair* (or simply α-fair when there is no confusion on $\boldsymbol{w}$) if the following holds:

$$\forall r \in [1,n], \quad y_r^* > 0 \quad \text{and} \quad \forall \boldsymbol{y} \in F, \quad \sum_{r=1}^{n} w_r \frac{y_r - y_r^*}{y_r^{*\alpha}} \leq 0.$$

Equivalently, $\boldsymbol{y}^*$ is $(\boldsymbol{w},\alpha)$-fair if, and only if $\boldsymbol{y}^*$ maximizes the α-fair utility function f^α defined over $F - \{0\}$:

$$f^\alpha(\boldsymbol{y}) = \sum_{r=1}^{n} f_r^\alpha(y_r),$$

where

$$f_r^\alpha(y_r) = \begin{cases} w_r \log(y_r) & \text{if } \alpha = 1, \\ w_r \frac{y_r^{1-\alpha}}{1-\alpha} & \text{otherwise.} \end{cases}$$

The success of α-fairness is due to its generality: in fact, for $\alpha = 0, 1, 2, \infty$ it is equivalent to max-throughput, proportional fairness, min-delay, and max-min fairness, respectively [34]. The introduction of the weight vector $\boldsymbol{w}$ permits to assign to a request a level of priority: the larger w_r, the higher the system's incentive to allocate the available bandwidth to request r. In general, w_r can model the number of connections of the same type of a connection request r or the amount of backlogged traffic for this request, for example.

Sharing policy. We define the *link-path* incidence matrix $\mathbf{A} \in \mathbf{R}^{|\mathcal{J}|\times|\mathcal{P}|}$, and the *request-path* incidence matrix $\mathbf{B} \in \mathbf{R}^{|\mathcal{R}|\times|\mathcal{P}|}$, as the following:

$$A_{jp} = \begin{cases} 1 & \text{if } j \in J_p \\ 0 & \text{otherwise} \end{cases}, \text{ and } B_{rp} = \begin{cases} 1 & \text{if } p \in P_r \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

The multi-path α-fair bandwidth sharing policy can be defined as follows. For each request r, the network can attribute a bandwidth of $x_p \geq 0$ along one or more of its established paths $p \in P_r$. The *path-wise allocation* $\boldsymbol{x}$ is then defined as the vector $(x_p)_{p\in\mathcal{P}}$. The *aggregate bandwidth* of request r, $y_r \geq 0$, represents the total bandwidth allocated to request r, that is,

$$y_r \coloneqq (\mathbf{B}\boldsymbol{x})_r = \sum_{p\in P_r} x_p. \tag{2}$$

The goal of the network is to find a path-wise allocation $\boldsymbol{x}$ in such a way that the aggregate bandwidth vector $\boldsymbol{y} \coloneqq (y_r)_{r\in\mathcal{R}}$ maximizes the α-fair metric. The path-wise allocation $\boldsymbol{x}$ should also respect the link capacity constraints of the network, that read:

$$\mathbf{A}\boldsymbol{x} \leq \boldsymbol{c} \Leftrightarrow \sum_{p:j\in J_p} x_p \leq c_j \quad \forall j \in \mathcal{J}. \tag{3}$$

Therefore, we have the problem definition below:

Definition 2. The *multi-path α-fair allocation problem* is the problem of finding a path-wise bandwidth allocation $\boldsymbol{x}$ (and therefore its corresponding aggregate bandwidth allocation $\boldsymbol{y} = \mathbf{B}\boldsymbol{x}$) such that the function f^α is maximized with respect to $\boldsymbol{y}$ and the link capacity constraints $\mathbf{A}\boldsymbol{x} \leq \boldsymbol{c}$ are respected.

It is well-known that the function f^α admits a unique maximizer over any convex closed bounded set. In our case, this means that the optimal aggregate bandwidth allocation $\boldsymbol{y}^*$ is unique. We draw the reader's attention to the fact that the fairness is measured upon the aggregate bandwidth allocation $\boldsymbol{y}$, not upon the path-wise allocations: one request may be allocated resources along several paths which do not need to be "fair" to each other. Remarkably, for the unique optimal aggregate bandwidth allocation $\boldsymbol{y}^*$, there may[1] be several path-wise allocations that verify the equation $\mathbf{B}\boldsymbol{x} = \boldsymbol{y}^*$.

3.2 Problem formulation

In the rest of this article, we adopt the convex optimization terminology. Define for each $r \in \mathcal{R}$ the convex cost function $g_r^\alpha(y_r) \coloneqq -f_r^\alpha(y_r)$. Then, $g^\alpha(\boldsymbol{y}) \coloneqq \sum_{r\in\mathcal{R}} g_r^\alpha(y_r)$ is a convex, closed and proper[2] function over $\mathbf{R}_+^{|\mathcal{R}|}$. We introduce ι as the *convex indicator function* of the capacity constraints:

$$\iota(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \mathbf{A}\boldsymbol{x} \leq \boldsymbol{c}, \boldsymbol{x} \geq 0 \\ \infty, & \text{otherwise.} \end{cases}$$

Then the multi-path α-fair allocation problem (Def. 2) can equivalently be formulated as the following convex program:

1. The unicity of $\boldsymbol{y}^*$ follows from the strict convexity of f^α as a function of $\boldsymbol{y}$. As the function $\boldsymbol{x} \mapsto f^\alpha(\mathbf{B}\boldsymbol{x})$ is *not generically* strictly convex with respect to the path-wise variable $\boldsymbol{x}$, the unicity of a path-wise optimum is no more guaranteed.
2. The term *closed* stands for lower semi-continuous and *proper* means non-identically equal to ∞.

$$\min_{\boldsymbol{y},\boldsymbol{x}} \sum_{r\in\mathcal{R}} g_r^{\alpha}(y_r) + \iota(\boldsymbol{x}), \tag{4}$$

$$\text{s.t.} \quad \boldsymbol{y} - \mathbf{B}\boldsymbol{x} = 0. \tag{5}$$

3.3 ADMM as an augmented Lagrangian splitting

The Alternating Directions Method of Multipliers is directly applicable to the form of the convex optimization problem (4)–(5). To do so, we write the augmented Lagrangian function of the problem, for a given penalty parameter $\lambda^{-1} > 0$, where $\boldsymbol{u}$ is the vector of Lagrange multipliers associated to the constraints (5):

$$L_{\lambda^{-1}}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{u}) = g^{\alpha}(\boldsymbol{y}) + \iota(\boldsymbol{x}) + \boldsymbol{u}^{\mathrm{T}}(\boldsymbol{y} - \mathbf{B}\boldsymbol{x}) + \frac{1}{2\lambda}\|\boldsymbol{y} - \mathbf{B}\boldsymbol{x}\|^2 \tag{6}$$

where for two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, $\boldsymbol{a}^{\mathrm{T}}\boldsymbol{b}$ is the Euclidean product of $\boldsymbol{a}$ and $\boldsymbol{b}$ and $\|\cdot\|$ is the Euclidean norm.

Unlike the classic method of multipliers (see for instance [9], Chapter 3), where the augmented Lagrangian is minimized jointly with respect to $(\boldsymbol{x},\boldsymbol{y})$ each time the dual variable $\boldsymbol{u}$ is updated (see Eq. (7c) below), the ADMM consists of minimizing the augmented Lagrangian $L_{\lambda^{-1}}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{u})$, alternatively with respect to $\boldsymbol{x}$ and $\boldsymbol{y}$. After some straightforward algebra, one can derive the update rules that represent each of the alternate minimizations, and summarize, through Eqs. (7a)–(7c) below, one iteration of ADMM that is to be repeated till a suitable termination condition is satisfied.

$$\begin{aligned}\boldsymbol{y} &\longleftarrow \operatorname{argmin} L_{\lambda^{-1}}(\boldsymbol{x},\cdot,\boldsymbol{u})\\ &= \operatorname*{argmin}_{\boldsymbol{y}} g^{\alpha}(\boldsymbol{y}) + \frac{1}{2\lambda}\|\boldsymbol{y} - (\mathbf{B}\boldsymbol{x} - \lambda\boldsymbol{u})\|^2 \qquad (7a)\\ \boldsymbol{x} &\longleftarrow \operatorname{argmin} L_{\lambda^{-1}}(\cdot,\boldsymbol{y},\boldsymbol{u})\\ &= \operatorname*{argmin}_{\boldsymbol{x}} \iota(\boldsymbol{x}) + \frac{1}{2\lambda}\|\mathbf{B}\boldsymbol{x} - (\boldsymbol{y} + \lambda\boldsymbol{u})\|^2 \qquad (7b)\\ \boldsymbol{u} &\longleftarrow \boldsymbol{u} + \frac{1}{\lambda}(\boldsymbol{y} - \mathbf{B}\boldsymbol{x}) \qquad (7c)\end{aligned}$$

We refer to this algorithm as the *centralized* algorithm.

In the light of the above update rules, one can directly remark that the centralized algorithm assumes that the function ι is globally accessible (see Eq. (7b)). The function ι is the indicator function of the network's capacity set, hence contains and necessitates full knowledge of the network topology and load. However, we assume that our distributed setting cannot afford such global knowledge. Indeed, the SDN controllers have the sole topology information of the underlying sub-network they have been assigned to. Thus, they are able to perform local computations only and have to operate together to achieve global consensus, which forbids the function ι from being globally available.

Remark 1. The centralized algorithm (7a)–(7c) benefits from the best presently known convergence rate. Indeed, formulation (4)–(5) belongs to a class of problems with strongly convex objectives having Lipschitz gradient that respects the assumptions of Theorem 1 in [16] and as such, the convergence of ADMM applied to solve them is linear with a known explicit rate. Unavoidably, we will sacrifice this convergence speed guarantee in order to abide by the rules of the distributed SDN settings.

In the next section, we show how to break down the topology information with respect to the SDN distribution, by benefiting from the distributive properties of ADMM.

4 Multi-agent consensus formulation

In this section, we show how to skirt the need for the global topology information by decomposing the formulation with respect to the network links of each SDN domain. The global knowledge of the topology being not affordable in the distributed SDN control plane, the decomposition permits to respect the locality of information of the different domain controllers. As we will see, our decomposition will break down the function ι by partitioning the network topology into different domains, and will naturally induce a partition of the set of requests over the network. This will therefore partition the global information and distribute it among the domains accordingly. We assume the domains can operate only with their private information, and any other information needed that belong to another controller needs to be gathered through inter-domain communication. The partition into domains can be orchestrated at the discretion of the SDN architect. Unavoidably though, domains will need to exchange information as paths may traverse many of them.

4.1 Preliminaries

We suppose that the network is split into M domains, where each domain m is assigned to a set of links $\mathcal{J}_m \subset \mathcal{J}$. In other words, $(\mathcal{J}_m)$ forms a *partition* of the set of links $\mathcal{J}$. Let $\mathcal{P}_m$ be the set of paths traversing the domain $\mathcal{J}_m$ via some link $j \in \mathcal{J}_m$.

We partition the set of requests $\mathcal{R}$ in the following way. Each request r has one source node that belongs to one unique domain $m(r)$. Define, for each domain m, the set of requests originating from it as $\mathcal{R}_m = \{r \in \mathcal{R} \text{ s.t. } m = m(r)\}$. Notice that $(\mathcal{P}_m)$ forms a covering of $\mathcal{P}$, and $(\mathcal{R}_m)$ forms a partition of $\mathcal{R}$.

Now, each domain m can define a set of (private, as explained later) indicator functions for each of its links: let ι_j denote the indicator function for link $j \in \mathcal{J}_m$, i.e.,

$$\iota_j(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \sum_{p:j\in J_p} x_p \le c_j \text{ and } x_p \ge 0 \ \forall p \text{ s.t. } j \in J_p \\ \infty & \text{otherwise.} \end{cases} \tag{8}$$

Remark 2. The function ι_j depends only on the variables x_p such that $j \in J_p$.

To simplify the algorithm design, we eliminate the matrix $\mathbf{B}$ and the variable $\boldsymbol{y}$ from the formulation by plugging in directly the equation (2) into the functions g_r^α. We therefore redefine the (now private) domain functions g_m^α as:

$$g_m^\alpha : \mathbf{R}_+^{\mathcal{P}} \to \mathbf{R}, \boldsymbol{x} \mapsto \sum_{r\in\mathcal{R}_m} g_r^\alpha\Big(\sum_{p\in P_r} x_p\Big), \tag{9}$$

Remark 3. In fact, the function g_m^α depends only on the variables x_p, $p \in P_r$, for $r \in \mathcal{R}_m$.

4.2 Private variables and consensus form

As from now, the controller of domain m is the sole owner of the now private indicator functions $\iota_j, j \in \mathcal{J}_m$ and $g_m^a lpha$. In order to evaluate its private function g_m^α, domain m defines a variable $\boldsymbol{x}^m$ as its private copy of the variable $\boldsymbol{x}$. As g_m^α depends only on a reduced number of variables (see Rem. 3), one only needs to define $\boldsymbol{x}^m = \left((x_p^m)_{p\in P_r}\right)_{r\in\mathcal{R}_m}$.

In the same way, each link j belonging to domain m is associated to a private copy $\boldsymbol{z}^j$ of the vector $\boldsymbol{x}$ through which the controller will evaluate the indicator function ι_j. Likewise, thanks to Rem. 2, we only need to define $\boldsymbol{z}^j = (z_p^j)_{p:j\in J_p}$, to avoid creating unnecessary variables.

The notation $\boldsymbol{z}^m$ will from now on be adopted to refer to the collection of variables $\{\boldsymbol{z}^j\}_{j\in\mathcal{J}_m}$.

In short, the minimal information that a domain m needs is:

1) the SDN-domain topology $\mathcal{J}_m$ and the load on each link $j \in \mathcal{J}_m$, that is, the sub-matrix $(A_j)_{j\in\mathcal{J}_m}$.
2) the knowledge of the objective functions g_r^α for each $r \in \mathcal{R}_m$.

Following the above notations, we can reformulate the objective (4) as the following:

$$\sum_{m=1}^{M} h^m(\boldsymbol{x}^m, \boldsymbol{z}^m), \tag{10}$$

where the functions h^m are defined as:

$$h^m(\boldsymbol{x}^m, \boldsymbol{z}^m) \coloneqq g_m^\alpha(\boldsymbol{x}^m) + \sum_{j\in\mathcal{J}_m} \iota_j(\boldsymbol{z}^j).$$

Finally, the controllers will enforce a consensus value among all the copies of a same variable via the following (global) indicator function:

$$\chi((\boldsymbol{x}^m)_m, (\boldsymbol{z}^m)_m) = \begin{cases} 0 & \text{if } z_p^j - x_p^m = 0 \\ & \forall m, \forall p \in P_r, \forall r \in \mathcal{R}_m, \forall j \in J_p \\ \infty & \text{otherwise.} \end{cases} \tag{11}$$

Hence, the new function to minimize to solve our problem is simply $\sum h^m + \chi$. We separate it further to end up with the classic 2-block form applicable to ADMM by introducing special copies $\boldsymbol{x}'^m$ and $\boldsymbol{z}'^m$ of the collections of variables $\boldsymbol{x}^m$ and $\boldsymbol{z}^m$, respectively for the composite term χ. Considering the present setting, one can verify that our problem takes the equivalent separated form:

$$\min \quad \sum_{m=1}^{M} h^m(\boldsymbol{x}^m, \boldsymbol{z}^m) + \chi((\boldsymbol{x}'^m)_m, (\boldsymbol{z}'^m)_m) \tag{12}$$

$$\text{s.t.} \quad \boldsymbol{z}^j = \boldsymbol{z}'^j \quad \forall j \in \mathcal{J} \tag{13}$$

$$\boldsymbol{x}^m = \boldsymbol{x}'^m \quad \forall m = 1, \dots, M. \tag{14}$$

To recap, *we have artificially separated the objective function by creating a minimal number of copies of the variable $\boldsymbol{x}$ in order to fully distribute the problem*. Now, instead of a global resource allocation variable, several copies of the variable account for how its value is perceived by each link of each domain. This new formulation can be interpreted as a multi-agent consensus problem formulation where domain m has cost h^m and has to agree on the values it shares with other domains. To enforce an intra- (local) and inter- (global) domain consistent value of the appropriate allocation, consensus constraints (11) are added to the problem. Those consensus constraints are now handled by the agent χ that will represent the communication steps required at each iteration between domains.

4.3 Update rules

In the same fashion as in Section 3.3, we write the augmented Lagrangian form of Formulation (12)–(14), and obtain, after some simplification, Algorithm 1 (Fast Distributed (FD)-ADMM). The variables $\boldsymbol{u}^m$ and $\boldsymbol{v}^m$ are the dual variables associated to the constraints (13) and (14) respectively.
Alg. 1 can be summarized into the three following steps:

Stage 1) Objective minimization – solve :

$$\underset{(\boldsymbol{x},\boldsymbol{z})}{\operatorname{argmin}} \sum_{m=1}^{M} \Bigg\{ h^m(\boldsymbol{x}^m, \boldsymbol{z}^m) + \frac{1}{2\lambda}\left(\|\boldsymbol{x}^m - \boldsymbol{x}'^m + \lambda \boldsymbol{v}^m\|^2 + \|\boldsymbol{z}^m - \boldsymbol{z}'^m + \lambda \boldsymbol{u}^m\|^2 \right) \Bigg\} \tag{15}$$

Stage 2) Consensus operation – setting $\boldsymbol{x}^m$ and $\boldsymbol{z}^m$ as the result of Stage 1), solve :

$$\underset{(\boldsymbol{x}',\boldsymbol{z}')}{\operatorname{argmin}} \chi((\boldsymbol{x}'^m)_m, (\boldsymbol{z}'^m)_m) + \frac{1}{2\lambda} \sum_{m=1}^{M} \left(\|\boldsymbol{x}^m - \boldsymbol{x}'^m + \lambda \boldsymbol{v}^m\|^2 + \|\boldsymbol{z}^m - \boldsymbol{z}'^m + \lambda \boldsymbol{u}^m\|^2 \right) \tag{16}$$

Stage 3) Dual variables update – see Alg. 1 lines 6 and 9.

For presentation purposes we presented those three stages in Alg. 1 in the order 2), 3), 1) to start with the information gathering and end with the information broadcasting of each domain controller. Of course, it has no incidence on the algorithm behavior as long as the three steps are executed in the correct order.

By separability of Eq. (15), the objective minimization stage boils down to the parallel minimization of each term of the sum by the corresponding domain controller. In details, the controller of domain m can minimize the term of index m in the sum (15) with respect to $\boldsymbol{x}^m$ and $\boldsymbol{z}^m$, *simultaneously*, meaning that the minimization operations with respect to $\boldsymbol{x}^m$ and $\boldsymbol{z}^m$ are *independent*.

The corresponding update rules can be stated as follows:

Stage 1) a) The minimization over $\boldsymbol{x}^m$:

$$\underset{\boldsymbol{x}^m}{\operatorname{argmin}} \sum_{r\in\mathcal{R}_m} \left\{ g_m^{\alpha}(\boldsymbol{x}^m) + \frac{1}{2\lambda} \sum_{p\in P_r} \|x_p^m - x_p'^m + \lambda v_p^m\|^2 \right\}.$$

This problem is separable with respect to $r \in \mathcal{R}_m$. Hence, we minimize it term by term. A term of index r of the above sum is differentiable, strictly convex (because of the 2-norm term) and coercive (near 0 and ∞) with respect to the positive variables $(x_p^m)_{p\in P_r}$. Therefore, a unique minimum exists and is obtained at the unique critical point. By differentiation, the critical point verifies:

$$x_p^m - x_p'^m + \lambda v_p^m = \frac{\lambda w_r}{(\sum_{r\in P_r} x_p^m)^{\alpha}}.$$

Thus, by setting $y_r \coloneqq \sum_{r\in P_r} x_p^m$, one has:

$$y_r^{\alpha}(x_p^m - x_p'^m + \lambda v_p^m) = \lambda w_r. \tag{17}$$

Summing Eq. (17) over $p \in P_r$, a necessary condition for y_r is:

$$y_r^{\alpha+1} - y_r^{\alpha} \sum_{p\in P_r} \{x_p' - \lambda v_p^m\} - \lambda w_r = 0. \tag{18}$$

A quick study of the left hand side of Eq. (18) shows that there is a unique positive solution. Therefore, it is an uni-variate unconstrained problem that can be solved efficiently with standard root finding algorithms (particularly when α is an integer, it boils down to finding the unique positive root of a polynomial of degree $\alpha + 1$). Eq. (17) in turn yields the path-wise update rule:

$$x_p^m = \frac{\lambda w_r}{y_r^{\alpha}} + x_p'^m - \lambda v_p^m. \tag{19}$$

Stage 1) b) The minimization over $\boldsymbol{z}^m$:

$$\underset{\boldsymbol{z}^m}{\operatorname{argmin}} \sum_{j\in\mathcal{J}_m} \left\{ \iota_j(\boldsymbol{z}^j) + \frac{1}{2\lambda} \sum_{p:j\in J_p} \|z_p^j - z_p'^j + \lambda u_p^j\|^2 \right\}.$$

Likewise, this problem is separable with respect to $j \in \mathcal{J}_m$. Hence, we minimize it term by term. The minimization of the term j boils down to operating the Euclidean projection of the point $\varphi^j = (z_p'^j - \lambda u_p^j)_{p:j\in J_p}$ onto the capacity set of the link j. That is, we find the closest point[3] to φ^j lying in the set $\{(z_p^j)_{p:j\in J_p} \geq 0 \text{ s.t. } \sum_p z_p^j \leq c_j\}$. This set is in fact a simplex of

3. With respect to the 2-norm.

Algorithm 1 Fast Distributed ADMM (FD-ADMM)

1: **procedure of domain** m
2:
STAGE 2):
3: Collect $\dot{\boldsymbol{z}}^q$ from neighbors
4: FORM $\tilde{\boldsymbol{z}}_p^m = \frac{1}{|J_p|+1}\left(\sum_{n:p\in\mathcal{P}_n} \dot{\boldsymbol{z}}_p^n\right)$ ▷ $\forall r \in \mathcal{R}_m, p \in P_r$
5:
STAGES 3) AND 1):
6: $u_p^j \leftarrow u_p^j + \frac{1}{\lambda}(z_p^j - \tilde{z}_p^m)$ ▷ $\forall p$ s.t. $j \in J_p$
7: Update $\boldsymbol{z}^j$ as the Euclidean projection of $(\tilde{z}_p^m - \lambda u_p^j)_{p:j\in J_p}$ onto the capacity set of link j (see **1. b)** ▷ $\forall j \in \mathcal{J}_m$
8:
9: $v_p^m \leftarrow v_p^m + \frac{1}{\lambda}(x_p^m - \tilde{z}_p^m)$
10: Update x_p^m as in Eq. (19) ▷ $\forall r \in \mathcal{R}_m, p \in P_r$
11:
BUILD INFORMATION TO BE SENT TO NEIGHBORS:
12: $\dot{z}_p^m \leftarrow \sum_{j\in J_p\cap\mathcal{J}_m} z_p^j$ ▷ $\forall p \in \mathcal{P}_m$
13: $\dot{z}_p^m \leftarrow \dot{z}_p^m + x_p^m$ ▷ $\forall r \in \mathcal{R}_m$
14: Send $\dot{\boldsymbol{z}}^m$ to neighbors
15: **end procedure**

dimension $n_j = \mathrm{Card}(\{p : j \in J_p\})$ and radius c_j, and this operation can be done [15] with a complexity dominated by the one of sorting a list of length n_j (thus, $\mathcal{O}(n_j \log(n_j))$ in average).

Stage 2) The consensus operation: As for the consensus operation stage, it suffices[4] to build the consensus point $\tilde{\boldsymbol{z}} \in \mathbf{R}^{\mathcal{P}}$, where:

$$\tilde{z}_p = \frac{1}{1+|J_p|}\left(x_p^{m(r)} + \sum_{\substack{m=1\dots M \\ j\in\mathcal{J}_m\cap J_p}} z_p^j\right) \quad \forall r, \forall p \in P_r. \tag{20}$$

In order to produce this point, domain m communicates its contribution $\dot{\boldsymbol{z}}^m$ to the sum encountered in the average (20) (l.12). Then the actual consensus value for each component p is recovered by dividing by the number of existing copies of variables with index p. Let r be the request such that $p \in P_r$. Then, one can check that for each path p, this number equals $|J_p| + 1$: one copy per link and one copy for the domain m such that $m(r) = r$ (l.4).

Communication among domain controllers: In FD-ADMM, *only domains that do share a path together have to communicate*. The communication procedures among the domain controllers are described between the lines 12 and 14. In these steps, the domains gather from and broadcast to adjacent domains the sole information related to paths that they have in common. In particular, domains are blind to paths that do not visit them, and can keep their internal paths secret from others. In details, after each iteration of the algorithm, each domain m receives the minimal information from other domains such that m is still able to compute a local consensus value $\tilde{z}^m$ (l. 4). Next, domain m sends back to the neighboring domains a contribution $\dot{z}^m$ so that they can recover a consensus value of the path-wise allocation.

Communication overhead: In terms of overhead, we can easily evaluate the number of floats transmitted between each pair of domain at each iteration. At each communication, domain m must transmit $\dot{z}_p^m$ for each path $p \in \mathcal{P}_m$ to each other domain that p traverses. The variable $\tilde{z}$ does not need to be centralized or transmitted between controllers. Each domain controller may actually have a copy of it recovered locally at negligible cost (see l.4). Note that the value of $|J_p|$ in l.4 can be recovered by each domain through a *unique* message passing at the establishment of the path p, therefore we can dismiss the global nature of this information. Hence, domain m transmits in total $\sum_{n\neq m} |\mathcal{P}_n \cap \mathcal{P}_m|$ floats to the set of its peers. As a comparison, in a distributed implementation of the standard dual algorithm given in [25], each domain m would transmit in total $\sum_{n\neq m} |\{j \in \mathcal{J}_m, \exists p \in P_n \text{ s.t. } j \in J_p\}|$ floats to the set of its peers, which is bounded by $(M-1)|\mathcal{J}_m|$ as $|\mathcal{P}|$ grows.

Practical implementation. Domain controllers implementing FD-ADMM are communicating bandwidth allocation decisions to SDN switches using standard protocols such as OpenFlow, PCEP (Path Computation Element Protocol) or BGP-LS (Border Gateway Protocol for Link State) available at their south-bound interface. To exchange information with other domain controllers about optimization variables or network states, i.e. east-west communications, domain controllers may use the iSDNi or IOCONA interfaces, in the case of the two most popular open source SDN controllers, OpenDayLight [3] and ONOS [2] respectively.

Feasibility preservation: A potential drawback of the distributed approach is the potential feasibility violation by the iterate $\tilde{z}$. However, we have the following positive result.

Proposition 1. *At each iteration of FD-ADMM, the point $\boldsymbol{z}^\dagger$, defined as $z_p^\dagger = \min_{j\in J_p} z_p^j,\ \forall p \in \mathcal{P}$, is feasible, and $\mathbf{B}\boldsymbol{z}^\dagger$ converges to the optimal aggregate bandwidth allocation.*

4. The minimization of Eq. (16) can be done with a straightforward calculus that we ommit here. Moreover, we explain in [6] how to obtain this simplified expression.

Proof. See [6], Proposition 1. □

Thus, in a certain way, for sufficiently loaded and communicating domains (i.e. the $|\mathcal{P}_m \cap \mathcal{P}_n|$ are large enough) we sacrifice some overhead (counted on a per iteration basis) compared to standard dual methods, but in exchange for anytime feasibility, a major feature that dual methods do not generically provide.

Penalty parameter initialization: It is well known that the penalty parameter λ highly conditions the convergence speed of ADMM. In [5], it has been shown numerically that the optimal penalty λ_* in terms of convergence speed for the (centralized) algorithm given in the update rules (7a)–(7c) provides a satisfactory performance of FD-ADMM. The analysis of this aspect of the design of FD-ADMM goes beyond the scope of this article, but in this paragraph, we adapt a multi-path version of Theorem 1 in [5] where a lower bound on the (path-wise) α-fair bandwidth allocation in the case of single paths is derived. It is shown there how this lower bound permits to define a natural penalty parameter initialization that boosts the algorithm performance. Here, we generalize the result of [5, Th. 1] to our case of multi-path routing, and establish a lower bound on the *aggregate bandwidth* allocation.

To do so, one needs the following definitions: we define, for each request r, the set $\mathcal{R}(r)$ of *other requests that visit some link used by* r: $\mathcal{R}(r) \coloneqq \left\{ s \in \mathcal{R} : \left(\cup_{q \in P_s} J_q\right) \cap \left(\cup_{p \in P_r} J_p\right) \neq \varnothing \right\}$. Lastly, we define the *utopic*[5] aggregate bandwidth allocation a_r of a request r, and its *local midpoint value* ϱ_r, respectively, as the following:

$$a_r \coloneqq \max_{\substack{x \geq 0, \mathbf{A}\boldsymbol{x} \leq \boldsymbol{c}, \\ \forall s \in \mathcal{R}(r) - \{r\}, (\mathbf{B}\boldsymbol{x})_s = 0}} (\mathbf{B}\boldsymbol{x})_r, \quad \varrho_r = \frac{w_r}{\sum_{s \in \mathcal{R}(r)} w_s} a_r. \tag{21}$$

Proposition 2. *Let* $r_\dagger \coloneqq \operatorname{argmin}_{s \in \mathcal{R}} \varrho_s$ *be the request with the smallest local midpoint value. Then, one can bound from below the optimal* aggregate bandwidth *allocation* $\boldsymbol{y}^* \geq \boldsymbol{d}$ *where:*

- *if* $\alpha \geq 1 \quad d_r = \varrho_{r_\dagger}^{1-1/\alpha} \varrho_r^{1/\alpha}$
- *if* $0 < \alpha \leq 1 \quad d_r = \left(\dfrac{w_r a_r}{\sum_{s \in \mathcal{R}(r)} w_s a_s^{1-\alpha}} \right)^{1/\alpha}$.

Proof. See [5], Theorem 1. □

Using this lower bound, we now generalize the penalty parameter that is formulated in [5]:

Penalty parameter initialization:

$$\lambda_* = \alpha \left(\min \frac{w_r}{a_r^{\alpha+1}} \max \frac{w_r}{d_r^{\alpha+1}} \right)^{-\frac{1}{2}}. \tag{22}$$

5 Reconfiguration with switching costs

In this section, we show that our FD-ADMM formulation is flexible enough to permit to solve a related relevant problem, by simply modifying the private objective functions h^m. We assume a traffic is already established with a current resource allocation, and that its requirements can vary on-the-fly. One can model a variation of the traffic requirements by a change in priorities between flows via a variation of the weight vector $\boldsymbol{w}$, the computation of a new path for an existing (or not) request, the elimination of a path for a request, etc. Under these circumstances, FD-ADMM can continuously generate feasible solutions to adapt the path-wise allocation to the new requirements in real-time. In fact, by doing so, the controllers may improve the optimality gap, and thus satisfy the demands with a better fairness measure as they evolve. However, enforcing a new resource allocation too often requires overwhelming flow reconfigurations rules that can cause Quality-of-Service degradation or system instability [30]. Therefore, we introduce a switching cost to limit the number of reconfigurations. The goal for the controllers will thus be to perform a trade-off between fairness and switching cost.

The introduction of a switching cost into the objective function can be of interest to enforce hard constraints onto the number of reconfigured paths. To be more specific, let $\boldsymbol{x}^0$ be a feasible path-wise allocation and assume the actual resource allocation of the demands follows $\boldsymbol{x}^0$. Now, the traffic demands have changed and the network has to recompute a new path-wise allocation $\boldsymbol{x}^*$ with fair aggregate bandwidth allocation $\mathbf{B}\boldsymbol{x}^*$, to respond to the traffic requirements. Assume the network has a budget of $\kappa > 0$ reconfigurations. According to the fairness policy of the network, the allocation should be updated in order to maximize the new fairness metric, without exceeding this budget:

$$\|\boldsymbol{x}^0 - \boldsymbol{x}^*\|_0 \leq \kappa, \Leftrightarrow \sum_p \mathbf{1}(x_p^0 \neq x_p^*) \leq \kappa, \tag{23}$$

where $\|u\|_0 = \operatorname{Card}\{p, u_p \neq 0\}$ is called the *zero-norm*[6] of a vector and denoted ℓ_0. Adding the constraint of Eq. (23) into the problem gives rise to a problem structure with integral constraints, and goes beyond the scope of this work. We consider here a relaxation of this problem that is still tractable with our method.

We can control the zero-norm (23) by adding the most natural sparsity inducing penalty induced by the θ-scaled ℓ_1-norm $\theta\|\boldsymbol{x}^0 - \boldsymbol{x}^*\|_1$, where θ is a positive parameter. The ℓ_1-norm is well known to be the fittest convex relaxation of the ℓ_0-norm, for the simple reason that the ℓ_1-ball is the convex hull of the set of points $\{\boldsymbol{v} \text{ s.t. } \|\boldsymbol{v}\|_0 \leq 1\}$. We therefore consider the problem described in Eqs. (12), (13), (14), with an extended expression of the function h^m:

5. In fact, the utopic path-wise and aggregate bandwidth allocations correspond to the path-wise and the aggregate bandwidth allocations, respectively, that a request can get if all other requests get an allocation of 0, which justifies the term *utopic*.

6. This is an abuse of terminology as it is not a norm.

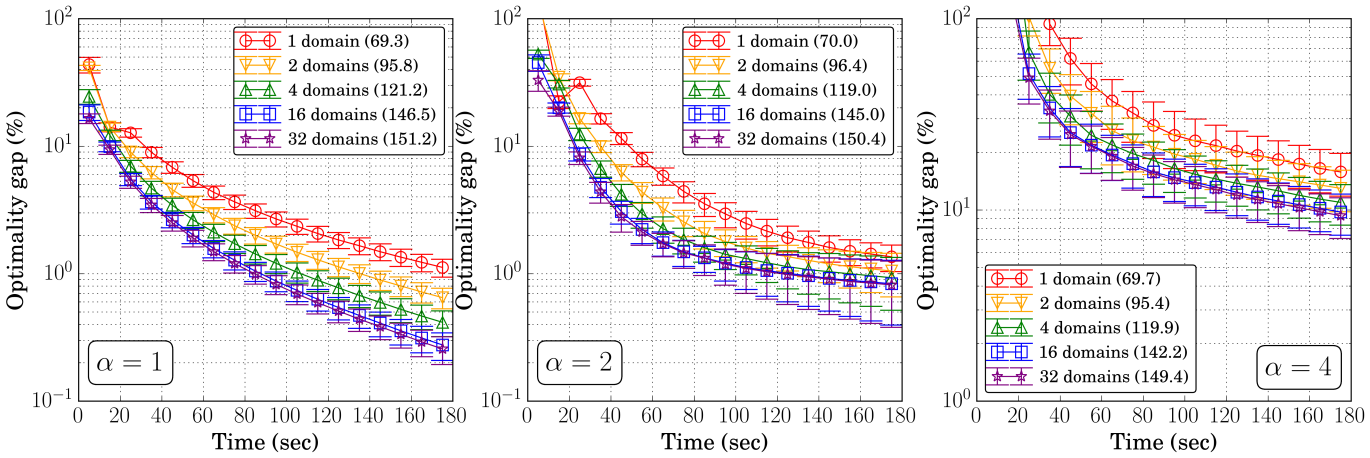

Fig. 2: **Convergence with domain distribution.** (Barabasi-Albert Graphs) Optimality gap over time within a deadline of 3 minutes for different numbers of domains. The average number of achieved iterations appear in the legend (inside parenthesis).

$$h^m(\boldsymbol{x}^m, \boldsymbol{z}^m) = g_m^\alpha(\boldsymbol{x}^m) + \theta \sum_{r \in \mathcal{R}_m} \sum_{p \in \mathcal{P}_r} |x_p^m - x_p^{m0}| + \sum_{j \in \mathcal{J}_m} \iota_j(\boldsymbol{z}^m), \tag{24}$$

where $\boldsymbol{x}^{m0}$ is the copy of $\boldsymbol{x}^0$ for domain controller m.

With this extended formulation, the changes in FD-ADMM only occur in the optimization stage 1) (Eq.(15)). The term-wise minimization of the functions h^m now also takes into account an incentive for each domain m to stay near the point $\boldsymbol{x}^{m0}$. Of course, a proper tuning of the parameter θ is necessary to enforce the real budget κ. The larger θ, the smaller the number of re-sized paths. We show this effect in the next section, dedicated to the experimentations.

6 Simulations

This section is dedicated to the experimentation of FD-ADMM. First of all, we demonstrate, on large instances with hundreds of nodes and thousands of requests and paths, the gains achievable with the distribution of the workload among several domain controllers (Fig. 2). We will not focus on absolute performance evaluation in terms of time, but in demonstrating the benefits of the distribution of the SDN centralized controller into SDN domain controllers that split the workload and build global solutions through consensus. Secondly, we analyze the behavior of FD-ADMM implemented as the extension described in Sec.5 (Fig. 4). All the simulations are executed under the three main sharing policies: proportional fairness ($\alpha = 1$), minimum potential delay ($\alpha = 2$) and max-min fairness (with an approximation $\alpha = 4$). In previous works [6], the performance of FD-ADMM with respect to convergence speed, feasibility preservation, and responsiveness in variable traffic requirement situations was extensively compared with the one of the classic Lagrangian method in [19]. We do not display these results here, and refer the curious reader to the aforementioned work.

6.1 Setting

Two simulations[7] were run over two types of networks. The one type of network was generated following the model of Barabasi-Albert with minimal degree 4, and the other was a Fat Tree with a number pods $k = 16$. We give here a description of the generated instances for the first simulation.

The Barabasi-Albert networks contained 500 nodes, which gave problems with 3968 resources. Over this network, instances of requests were created between sources and destinations chosen uniformly at random with a number of established paths from 2 to 4 between the pair of nodes. The instances contained $5,000$ requests (that represented approximately $11,000 - 15,000$ paths in total).

The Fat Tree (with 16 pods) networks contained $1,345$ nodes, which gave problems with $3,136$ resources. We modeled a connection to the Internet via the fat tree by adding a root node connected to the core nodes of the tree. For each server, we generated two connections to the root node through 4 paths each, and to another server chosen uniformly at random through 4 paths. This gave problems with $3,072$ requests and therefore $12,288$ paths.

The second simulation was executed on a small network (generated with the model of Barabasi-Albert with same minimal degree 4) with 100 nodes made of 768 links, and each instance contained 500 requests each with 1 to 2 established paths (approximately $700 - 800$ paths).

In all the simulations, the network capacities were fixed at an equal value (100) and unless specifically mentioned otherwise, the weights w were fixed at a unique value 1. We created the domains by splitting the set of network links into a number (equal to the desired number of domains) of equally sized subsets, taking into account the network topology so

7. The curious reader can find the source code an run the simulations in [4].

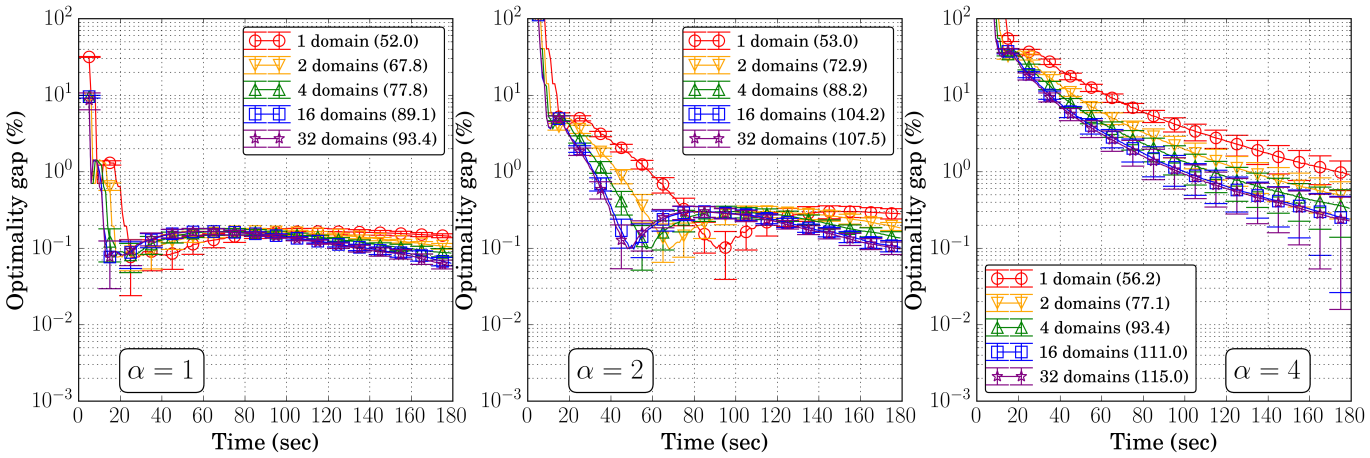

Fig. 3: **Convergence with domain distribution.** (Fat Tree Graphs) Optimality gap over time within a deadline of 3 minutes for different numbers of domains. The average number of achieved iterations appear in the legend (inside parenthesis).

that each domain remains connected. We assumed all the domains performed their update rule in a synchronized manner, meaning that an iteration was achieved (and the variable $\tilde{z}$ in stage 2) updated) *when all the controllers were done with their respective work*. The direct extension of distributed ADMM-based algorithms to the asynchronous setting is possible and its convergence is studied and demonstrated for instance in [13]. We keep simulations in the aforementioned case for future work.

6.2 Results

Convergence with domain distribution. We evaluated the gains achievable by the SDN distribution by plotting, for each partition of the network following Sec. 4 into a number of controllers within $\{1, 2, 4, 16, 32\}$, the achieved number of iteration and the optimality gap[8] with time under a finite time deadline. In our implementation, each controller performs its private update rules under one specific thread at each iteration. As the update rules for each controller are also massively parallelized, there is still a lot of room for optimizing absolute time performances. Also, the consensus operation stage is here done without exploiting the parallelism. In practice, domain controllers may be themselves equipped with hundreds of cores and perform multi-threads computations that can crush the computation times down to several orders of magnitude smaller. The deadline was fixed to three minutes for all instances. The results are shown in Fig. 2 (Barabasi-Albert Graphs) and Fig. 3 (Fat Tree Graphs). Each point represented corresponds to an average over 15 and 10 instances (for Fig. 2 and Fig. 3, respectively) of the same characteristics along with a 95%-confidence interval estimated following the t-distribution model.

The results show that the distribution of the computation permits to operate more than twice faster (according to the average achieved number of iterations) and thus to reach the same optimality gap faster. For instance, for $\alpha = 1$ in Fig. 2, in order to reach a gap below 2%, a single controllers needs more than 2 minutes whereas 32 controllers together need less than one minute. This reduction of the time can be even more dramatic when the update rules within each domain, as well as the consensus operation stage, are done in parallel.

The communication delay being dependent of the SDN implementation, we condense the information concerning the overall communication delay of controllers into the number of performed iterations. According to recent work on this topic [8], east-west interfaces between domain controllers can permit one to perform inter-controller communications with low latency. With the fast convergence that is demonstrated in Fig. 2 in terms of iteration count, one can see that the communication delay does not deteriorate the acceleration achieved with the distribution of the workload among the domain controllers.

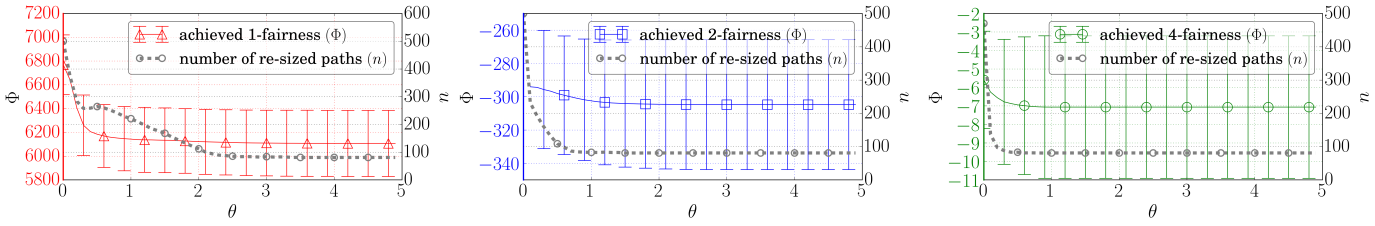

Fig. 4: **Switching costs.** Number of re-sized paths (n) and achieved fairness (Φ) versus the switching cost θ.

8. The optimal solution was obtained by running FD-ADMM till the convergence is provably obtained. This can be done by observing the values of the primal and dual residuals of the problem at each iteration. It is shown [12, Ch. 3.3.1] that these values bound the optimality gap of the iterate at each iteration. The residual values are thus used as a robust convergence detector. We consider the optimum was reached when the residual values dropped below 10^{-2} (modest but satisfactory convergence)

Switching costs. In the second experiment, we set x^0 as an optimal path-wise allocation for the actual setting of the weights $\boldsymbol{w}(\equiv \mathbf{1})$. Then, we simulate a new traffic requirement by choosing at random a new weight vector $\boldsymbol{w}_1$ within $[1, 10]$. Thus, FD-ADMM runs to find the new optimally fair aggregate bandwidth allocation, but, the value of θ forces a trade-off between fairness and switching cost. We ran FD-ADMM till convergence was provably obtained to a precision 10^{-2} and analyzed the number of paths that had been re-sized, for different values of θ. The minimization problem in stage 1) was solved optimally at each iteration using a standard optimization package for unconstrained problems with a tolerance 10^{-2} (as the closed form of (19) is no longer available for positive θ). To evaluate numerically the zero-norm of vectors, we used the function $N_\epsilon(x) = \mathrm{Card}(\{p : |x_p| > \epsilon\})$ where ϵ is a desired level of precision. This precision was fixed to the convergence tolerance of our experimentation, that is, $\epsilon = 10^{-2}$. At the optimum, the objective therefore splits up into a sum $-\Phi + \theta\Psi$, where $\Psi \coloneqq \sum_{p,|x_p - x_p^0|>\epsilon} |x_p - x_p^0|$, the number of terms in that sum being equal to $n \coloneqq N_\epsilon(\boldsymbol{x} - \boldsymbol{x}^0)$ and Φ is the achieved fairness $\sum_r f_r^\alpha(y_r)$ for the optimal aggregate bandwidth allocation $\boldsymbol{y} = \mathbf{B}\boldsymbol{x}$. The results are shown in Fig. 4. Likewise, each point represented corresponds to an average over 15 instances of the same characteristics along with a 95%-confidence interval estimated following the t-distribution model.

As expected, the larger the configuration cost θ, the smaller the number of reconfiguration. The results show that for a desired number of reconfiguration, it is possible to chose an appropriate value of θ to enforce it. Also, it can be seen that the reconfiguration cost does not deteriorate dramatically the system's sharing policy performance. This means it is possible to reconfigure small subsets of paths on-the-fly (to the limit of what is feasible in terms of reconfiguration budget), and still enforce a satisfactorily fair policy, that will be ultimately optimally fair if the traffic requirement stabilizes. In the figures, we only showed the points for small values of θ – this, along with the small precision level ϵ, explain the apparition of a plateau below which it seems impossible to go. In reality, even larger values of θ permit one to accomplish a very low (down to zero) number of reconfigurations. We do not plot the points for larger values of θ in order to focus on the decrease of the reconfiguration number in the very beginning.

7 Conclusion

We designed an algorithm, FD-ADMM, that solves optimally the multi-path α-fair resource allocation problem. We showed that FD-ADMM is fully distributed and that its implementation is suitable to distributed SDNs, regardless of the actual distribution of the networks into domains. The massively separable sub-problems given by FD-ADMM are efficiently solvable by the SDN domain controllers equipped with massively parallel hardware capable of addressing each update rule simultaneously as multiple-threads, thus reducing considerably the computation time and improving dramatically the responsiveness of the algorithm while providing in real-time feasible solutions. It was demonstrated numerically that the distribution of the SDN control permits to accelerate the overall system efficiency by attaining an equivalent optimality gap in highly reduced time. This shows that FD-ADMM scales naturally with the problems size, exploiting the computational power of modern SDN controllers. We also showed that the FD-ADMM extends easily to account for a switching cost per path, and that a trade-off between fairness and switching cost can be operated to preserve the system stability without deteriorating too much the resource allocation efficiency. In the future, we wish to specify the trade-off by providing theoretical bounds on the cost θ for this specific problem in order to respect any maximum allowed reconfiguration budget κ in the multi-path setting.

References

[1] Akka, https://akka.io.
[2] The onos project, https://onosproject.org.
[3] Opendaylight, https://www.opendaylight.org.
[4] Source code, https://github.com/zaidallybokus/multi_path.
[5] Allybokus, Z., Avrachenkov, K., Leguay, J., and Maggi, L. (2017a). Lower bounds for the fair resource allocation problem. In *Proc. IFIP Performance*.
[6] Allybokus, Z., Avrachenkov, K., Leguay, J., and Maggi, L. (2017b). Real-time fair resource allocation in distributed software defined networks. *Proc. ITC 29*.
[7] Bach, F., Jenatton, R., Mairal, J., Obozinski, G., et al. (2012). Optimization with sparsity-inducing penalties. *Foundations and Trends® in Machine Learning*.
[8] Benamrane, F., Mamoun, M. B., and Benaini, R. (2017). New method for controller-to-controller communication in distributed sdn architecture. *International Journal of Communication Networks and Distributed Systems*.
[9] Bertsekas, D. P. (2014). *Constrained optimization and Lagrange multiplier methods*. Academic press.
[10] Bertsekas, D. P., Gallager, R. G., and Humblet, P. (1992). *Data networks*. Prentice-Hall International Series.
[11] Blial, O., Ben Mamoun, M., and Benaini, R. (2016). An overview on sdn architectures with multiple controllers. *Journal of Computer Networks and Communications*, 2016.
[12] Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*.
[13] Chang, T.-H., Hong, M., Liao, W.-C., and Wang, X. (2016). Asynchronous distributed admm for large-scale optimization—part i: Algorithm and convergence analysis. *IEEE Transactions on Signal Processing*.
[14] Charny, A., Jain, R., and Clark, D. (1995). Congestion control with explicit rate indication. In *Proc. of IEEE ICC*.
[15] Chen, Y. and Ye, X. (2011). Projection onto a simplex. *arXiv preprint arXiv:1101.6081*.
[16] Deng, W. and Yin, W. (2016). On the global and linear convergence of the generalized alternating direction method of multipliers. *Journal of Scientific Computing*.
[17] Hassas Yeganeh, S. and Ganjali, Y. (2012). Kandoo: a framework for efficient and scalable offloading of control applications. In *Proc. of ACM HotSDN*.

[18] He, B. and Yuan, X. (2012). On the o(1/n) convergence rate of the douglas-rachford alternating direction method. *SIAM J. Numer. Anal*.
[19] Kelly, F. P., Maulloo, A. K., and Tan, D. K. (1998). Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society*.
[20] Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proc. of the IEEE*.
[21] Kumar, P., Yuan, Y., Yu, C., Foster, N., Kleinberg, R., Lapukhov, P., Lim, C. L., and Soulé, R. (2018). Semi-oblivious traffic engineering: The road not taken. In *USENIX NSDI*.
[22] Lee, T.-J. and Veciana, G. D. (1998). A decentralized framework to achieve max-min fair bandwidth allocation for atm networks. In *IEEE GLOBECOM 1998*, pages 1515–1520 vol.3.
[23] Liang, C. and Yu, F. R. (2015). Distributed resource allocation in virtualized wireless cellular networks based on admm. In *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*.
[24] Marasevic, J., Śtein, C., and Zussman, G. (2016). A fast distributed stateless algorithm for alpha-fair packing problems. In *Proc. of ICALP), vol.55, pp.54–1*.
[25] McCormick, B., Kelly, F., Plante, P., Gunning, P., and Ashwood-Smith, P. (2014). Real time alpha-fairness based traffic engineering. In *Proc. of ACM HotSDN*.
[26] Mo, J. and Walrand, J. (2000). Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking (ToN)*.
[27] Mota, J. F., Xavier, J. M., Aguiar, P. M., and Püschel, M. (2012). Distributed admm for model predictive control and congestion control. In *Proc. of IEEE CDC*.
[28] Obadia, M., Bouet, M., Leguay, J., Phemius, K., and Iannone, L. (2014). Failover mechanisms for distributed sdn controllers. In *NOF*. IEEE.
[29] Palle, U., Dhody, D., Singh, R., Fang, L., and Gandhi, R. (2016). Pcep extensions for mpls-te lsp automatic bandwidth adjustment with stateful pce. Technical report, IETF. Work in Progress.
[30] Paris, S., Destounis, A., Maggi, L., Paschos, G. S., and Leguay, J. (2016). Controlling flow reconfigurations in sdn. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE.
[31] Phemius, K., Bouet, M., and Leguay, J. (2014). Disco: Distributed multi-domain sdn controllers. In *Proc. IEEE NOMS*.
[32] Shin, J., Kim, T., Lee, B., and Yang, S. (2017). Iris-hisa: Highly scalable and available carrier-grade sdn controller cluster. *Mobile Networks and Applications*.
[33] Skivée, F. and Leduc, G. (2004). A distributed algorithm for weighted max-min fairness in mpls networks. In *International Conference on Telecommunications*. Springer.
[34] Srikant, R. and Ying, L. (2013). *Communication networks: an optimization, control, and stochastic networks perspective*. Cambridge University Press.
[35] Stallings, W. (2013). Software-defined networks and openflow. *The internet protocol Journal*.
[36] Sundaresan, R. et al. (2016). An iterative interior point network utility maximization algorithm. *arXiv preprint arXiv:1609.03194*.
[37] Vaughan-Nichols, S. J. (2011). Openflow: The next generation of the network? *Computer*.
[38] Voice, T. (2006). Stability of multi-path dual congestion control algorithms. In *Proc. of Valuetools*. ACM.