Real-Time Fair Resource Allocation in Distributed Software Defined Networks

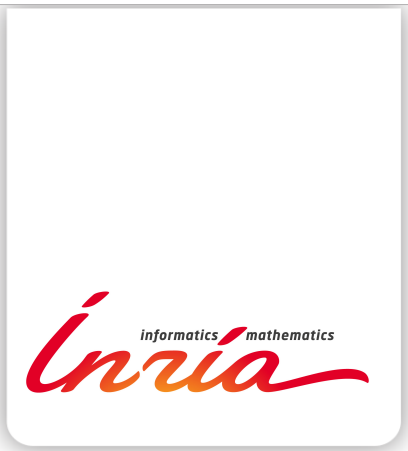
Zaid Allybokus, Konstantin Avrachenkov, Jérémie Leguay, Lorenzo Maggi

Real-Time Fair Resource Allocation in Distributed Software Defined Networks

Zaid Allybokus, Konstantin Avrachenkov, Jérémie Leguay, Lorenzo Maggi

RESEARCH REPORT
N° 9015
January 2017
Project-Team Neo

ISSN 0249-6399 ISRN INRIA/RR--9015--FR+ENG



Real-Time Fair Resource Allocation in Distributed Software Defined Networks

Zaid Allybokus*, Konstantin Avrachenkov†, Jérémie Leguay‡,

Lorenzo Maggi§

Project-Team Neo

Research Report n° 9015 — January 2017 — 20 pages

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 that tackles the fair resource allocation problem in a distributed SDN control architecture. Our 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 in real-time.

Key-words: Software-Defined Networking (SDN), Fair Resource Allocation, Network Utility Maximization (NUM) problem, α-fairness, Alternating Direction Method of Multipliers (ADMM), Distributed Algorithms, Distributed SDN Control Plane

* Huawei Technologies & Inria Sophia Antipolis
† Inria Sophia Antipolis
‡ Huawei Technologies
§ Huawei Technologies

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Partage équitable de ressources en temps réel dans les Software Defined Networks distribués

Résumé : La performance des réseaux informatiques est fortement liée au partage équitable de la bande-passante entre les différents flux. Lorsque la taille de ces flux varie constamment dans le temps, le problème de partage des ressources est non-trivial. Afin d'aborder ce problème, des techniques de partage pouvant réagir rapidement aux fluctuations de trafic sont désirables, en particulier pour le contrôle de grands réseaux avec des centaines de noeuds et des milliers de flux. Nous proposons un algorithme distribué qui s'attaque au problème de partage de ressources équitable dans le contexte des architectures Software-Defined Networks (SDN) distribuées. Cet algorithme génère en chaque instant des solutions convergeant vers le partage équitable en respectant toujours l'ensemble des contraintes, une propriété non satisfaite par les méthodes classiques de décomposition primale-duale. Grâce à la distribution des calculs, nous montrons que notre algorithme peut contrôler de grands réseaux en temps réel.

Mots-clés : Software-Defined Networking (SDN), Partage équitable de ressources, Maximisation de fonction d'utilité de réseau, α-fairness, Alternating Direction Method of Multipliers (ADMM), Algorithmes distribués, Distributed SDN Control Plane

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 [20] had envisioned the use of one central controller. However, for obvious scalability and resiliency reasons large networks already in production are considering a distributed SDN control plane [10]. Thus, a *logically centralized* network control plane may consist of multiple controllers each in charge of a SDN domain of the network and operating together, in a *flat* [18] or *hierarchical* [6] architecture, to achieve global tasks.

In this paper, we study the problem of finding a global fair resource allocation when the control plane is distributed over several domain controllers. More specifically, we consider the case where the size of flows evolves over time and bandwidth allocations have to be quickly adjusted towards the novel *fair* solution (in the sense of α-fairness defined by Mo et al. in [13]). In distributed SDN architectures, controllers operate with full information in their domain and communicate (e.g., system states or optimization variables) with adjacent domain controllers or a central gathering entity. Exchanges between controllers are expensive in terms of communication delay and overhead. In this setting, a distributed algorithm may not have enough time to converge to optimum before it has to provide a feasible answer due to the scale of networks. Therefore, the main challenge is to produce very quickly good quality feasible solutions. Local mechanisms such as Auto-Bandwidth [15] have been proposed to greedily and distributedly adjust the allocated bandwidth to support time-varying IP traffic in *Multi Protocol Label Switching* (MPLS) networks. However, they do not ensure fairness and do not optimize resources globally. On the other hand, centralized algorithms have been proposed to solve the problem but fail at quickly providing good and feasible solution in a distributed setting [12].

We propose a distributed algorithm that performs in real-time for the fair resource allocation problem in distributed SDN control planes. It is based on the *Alternating Direction Method of Multipliers* (ADMM) [2] that has captured a lot of attention recently for its separability and fast convergence properties. Our algorithm, called Fast Distributed (FD)-ADMM, is designed to be fully deployed over a distributed SDN control plane and permits controllers to handle their domain simultaneously while operating together in the fashion of a general distributed consensus problem to achieve a global optimal value. We show that this algorithm can function in real-time by iteratively producing a *feasible* (global) resource allocation strategy that converges to the α-fair optimal allocation. It produces very quickly (in fact, from the first iterations) good quality feasible solutions that permit to adjust in milliseconds bandwidth for flows that evolve quickly and need immediate response, a property that standard dual decomposition methods such as the one in [12] do not have. We argue that this property is crucial as the network state (e.g., flow size, flow arrival/departure, link/node congestions) may be affected by abrupt changes. Thus, we claim that 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. Therefore, we show that our algorithm is a good candidate for real-time fair resource allocation. To this aim, we compare the performances of the algorithm to the Lagrangian dual splitting method in [21, 12], a standard decomposition method that violates feasibility, demonstrates poor convergence rate, hence responds slower in real-time.

Moreover, we provide an explicit and adaptive tuning for the penalty parameter in FD-ADMM so that an optimal convergence rate can be approached on any instance execution. And

we show that projections can be massively parallelized link-by-link yielding a convergence rate of the algorithm that does not depend on the way the network is partitioned into domains.

The remainder of this paper is organized as follows. Section 2 surveys the related work around the real-time fair resource allocation problem. Sections 3 and 4 explicitly reformulate the fair resource allocation problem in the fashion of a general distributed consensus problem, addressed with the terminology of proximal algorithms. Section 5 discusses on an optimal tuning of the penalty parameter in FD-ADMM. Section 6 provides 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[1] has been the classic resource sharing principle [1] and has been studied extensively. The concept of *proportional fairness* and its weighted variants were introduced in [9]. Later, a spectrum of fairness metrics including the two former ones was introduced by Mo et al. in [13] as the family of α-fair utility functions. Some early notable work on max-min fairness includes [3], 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 [17], 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 [21], then McCormick et al. [12] tackle the α-fair resource allocation problem with a gradient descent applied to the dual of the α-fair resource allocation problem.

In these works, no mention is made on the potential (in fact, systematic) feasibility violation of the sequences generated by those algorithms, which we believe is a matter that deserves attention in real-time setups. Motivated by this, recently the authors of [19] provide a feasibility preserving version of Kelly's methodology in [9]. 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 the case of polymatroidal and flow aggregating networks only. As a matter of fact, we contribute to this problem with 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 [11] 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 $O(\frac{1}{n})$ best known convergence rate of ADMM [7] failed to explain its empirical fast convergence until very recently, for instance in [5], 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 in the augmented Lagrangian formulation (see Section

[1] A resource allocation strategy is said to be max-min fair if no route can increase its allocation while remaining feasible without penalizing another route that has a smaller or equal allocation

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 [5], the authors provide a linear convergence proof that yields a convergence rate in a closed form that can be optimized with respect to the objectives parameters. Therefore, thanks to these works, an optimal tuning of ADMM for α-fair resource allocation is now available. Several papers use the distributivity of ADMM to design efficient algorithms solving consensus problems in e.g. model predictive control and congestion control, without however addressing this fundamental detail. In the simulations of [14] for instance, the authors try several choices of the penalty parameter and plot the best result found for each point.

Our contribution: In this paper, we reformulate the α-fair resource allocation problem and we design a distributed algorithm based on ADMM, called FD-ADMM. We show that this algorithm outputs at each iteration a feasible resource allocation strategy that converges to the unique optimum of the problem. We also provide an adaptive strategy to correctly tune the FD-ADMM penalty parameter and we show that projections can be massively parallelized on a link-by-link basis. Finally, we show how our algorithm outperforms the dual methods mentioned above in terms of feasibility preservation and responsiveness in dynamic scenarios.

3 Fair resource allocation problem

In this section, we formulate the fair resource allocation problem as a convex optimization problem. We then propose an efficient distributed algorithm based on ADMM to solve it. For clarity, we start off with basic definitions and properties.

3.1 Basic properties

Let $f : \mathbf{R}^n \to \bar{\mathbf{R}} = \mathbf{R} \cup \{\infty\}$ be a closed proper convex function. The set $\mathrm{dom}(f)$ denotes the domain of f, that is the set upon which f takes real values.

Fact 1. *Assume* $\mathrm{dom}(f) \neq \emptyset$*. Then, the following facts hold:*

(i) For $u \in \mathbf{R}^n$*,* $\lambda \in \mathbf{R}_{++}$*, the minimization problem*

$$u^*_\lambda = \arg\min_x \left\{ f(x) + \frac{1}{2\lambda}||u - x||^2 \right\}$$

admits a unique solution. The (λ*-scaled)* proximal operator *of* f *is the well-defined map* $\mathrm{prox}_{\lambda f} : u \to u^*_\lambda$.

(ii) Assume that f *takes the form* $f(x, y) = g(x) + h(y)$ *(write* $f = g \bigoplus h$*) where* $(x, y) \in \mathbf{R}^p \times \mathbf{R}^{n-p}$*. Then, for* $(u, v) \in \mathbf{R}^p \times \mathbf{R}^{n-p}$*,* $\mathrm{prox}_{\lambda f}(u, v) = \big(\mathrm{prox}_{\lambda g}(u), \mathrm{prox}_{\lambda h}(v)\big)$.

(iii) Assume that f *is the indicator function of a closed convex non-empty set* C*:*

$$f(x) = \begin{cases} 0 & x \in C \\ \infty, & x \notin C. \end{cases}$$

Then $\mathrm{prox}_{\lambda f} = P_C$*, the Euclidean projection onto* C.

The properties of Fact 1 will be useful to formulate an efficient proximal splitting method for our problem that we next define.

3.2 Problem reformulation

Let R be a set of connection requests over a network with a set J of capacitated links. Each link $j \in J$ has a total capacity of $C_j \in \mathbf{R}_+$. Each request r is represented by a route containing a subset of J that, without any confusion, we still denote as r. Hence, we adopt the simple notations $j \in r$ or $r \in j$ to say that link j belongs to the route r, or route r goes through link j, respectively. Given the set of requests and their corresponding utility function f_r, the network allocates bandwidth to all the requests in order to maximize the overall utility $f = \bigoplus_r f_r$, while satisfying feasibility, that is the link capacity constraints. Denote by x_r the capacity allocated to route r. Then, we have the following capacity constraint:

$$Ax \leq C \tag{1}$$

where $A = (a_{jr})_{jr}$ is the link-route incidence binary matrix:

$$a_{jr} = \begin{cases} 1 & \text{if } j \in r \\ 0 & \text{otherwise.} \end{cases}$$

The capacity allocation x is chosen according to the following Network Utility Maximization (NUM) problem:

$$\max_{x \geq 0, Ax \leq C} \sum_{r \in R} f_r(x_r), \tag{2}$$

where $f_r(\cdot)$ is a utility function, for instance, weighted α-fairness utility:

$$f_r^\alpha(x) = \begin{cases} w_r \frac{x^{1-\alpha}}{1-\alpha}, & \alpha \neq 1, \\ w_r \log(x), & \alpha = 1. \end{cases}$$

The utility functions are assumed to be non-decreasing, concave, non-identically equal to $-\infty$, and upper semi-continuous, as it is the case for the family of α-fair utility functions defined above.

From now on, we adopt the convex optimization terminology. Define for each $r \in R$ the convex cost functions $g_r = -f_r$. Then, $g = \bigoplus_r g_r$ is a convex closed proper function over $\mathbf{R}_+^{|R|}$. Let us introduce ι as the indicator function of the convex closed set $\{Ax \leq C, x \geq 0\}_x$ (see Fact 1). Then our problem can equivalently be formulated as:

$$\min_{x,z} \sum_{r \in R} g_r(x_r) + \iota(z), \tag{3}$$

$$\text{s.t.} \quad x - z = 0 \tag{4}$$

3.3 ADMM as an augmented Lagrangian splitting

The ADMM is related to the *augmented Lagrangian* dual splitting *Method of Multipliers* in the following way. Let us write the augmented Lagrangian with penalty $\frac{1}{\lambda}$ for problem (3) – (4):

$$L_{\lambda^{-1}}(x, z, u) = g(x) + \iota(z) + u^{\mathrm{T}}(x - z) + \frac{1}{2\lambda}||x - z||^2 \tag{5}$$

where u is the vector of Lagrange multipliers, $a^{\mathrm{T}}b$ is the Euclidean product of a and b and $||\cdot||$ the Euclidean norm. The method of multipliers consists in the following update rules:

$$(x^{k+1}, z^{k+1}) = \arg\min_{x,z} L_{\lambda^{-1}}(x, z, u^k) \tag{M1}$$

$$u^{k+1} = u^k + \frac{1}{\lambda}(x^{k+1} - z^{k+1}). \tag{M2}$$

However, ADMM permits to decouple the variables (x, z) in the optimization stage M1: instead of a global optimization over (x, z), we only optimize $L_{\lambda^{-1}}$ with respect to the variable x, then, given the new update of x, we optimize $L_{\lambda^{-1}}$ with respect to z. ADMM can thus be expressed in the proximal (λ-scaled) form, which we refer to as *centralized* ADMM (C-ADMM).

Algorithm 1 Centralized ADMM (C-ADMM)

Input: Initial values z^0, v^0

1: **while** a suitable termination condition is not met **do**
2: $x^{k+1} = \text{prox}_{\lambda g}(z^k - v^k)$
3: $z^{k+1} = P(x^{k+1} + v^k)$
4: $v^{k+1} = v^k + x^{k+1} - z^{k+1}$
5: **end while**

In Algorithm 1, $P = \text{prox}_{\lambda\iota}$ is the projection on $\{Ax \leq C, x \geq 0\}_x$, and $v = \lambda u$ the λ-scaled dual variable. Now, the first step of Algorithm 1 (line 2) can be separated thanks to the separability property of the objective function, see Fact 1. In fact, g is fully separable, as $g(x) = \sum g_r(x_r)$. Thus, the proximal update of line 2 takes the trivially parallelized form:

$$\forall r \quad x_r^{k+1} = \text{prox}_{\lambda g_r}(z_r^k - u_r^k) \tag{6}$$

such that each local variable x_r can be computed separately.

Through expression (6), we are thus able to provide an efficient update rule for x, provided that the separate proximal computations are inexpensive. However, two main issues arise.

First, an update of the variable z in line 3 of Algorithm 1 requires full knowledge of the projection mapping, which in turn requires full information on the capacity set of the network. Thus, this global update rule represents an important limiting factor to the design of a fully distributed algorithm, which is our main design interest here to follow the distribution of SDN control planes.

Moreover, although the convergence of C-ADMM may only require some tens of iterations (see Section 6 for further details), it may be slow in terms of computation time due the successive application of a projection algorithm that would not scale with respect to the problem size. This also gives rise to a double loop algorithm where each iteration requires the convergence of a inner process that can be time-consuming. Indeed, computing the projection of a generic point onto a closed convex non-empty polyhedron is in general non-trivial. Hence, for general polyhedra, one has to operate alternate projections, summon quadratic programming solvers or use iterative algorithms such as the one in [8].

We address such issues in the next section, where we propose FD-ADMM, a distributed version of C-ADMM.

4 The general consensus form of ADMM: an efficient distributed algorithm design

In this section we show how to alleviate the cost of the global projection sub-routine in C-ADMM (line 3) by decomposing the formulation with respect to the network links of each SDN domain.

This scheme permits to reduce the size of the projections sub-problems that are addressed efficiently and that can run in parallel. The decomposition into domains can be orchestrated without any constraint. Unavoidably though, domains will need to exchange information as routes may traverse different domains.

4.1 Preliminaries

We organize the network into several domains $J_p, p = 1 \dots P$ such that $(J_p)_p$ forms a partition of the set of links J. Let R_p be the set of routes traversing the domain J_p via some link $j \in J_p$. More formally, $R_p = \{r \in R : \exists\, j \in J_p \text{ s.t. } j \in r\}$. Hence, $(R_p)_p$ forms a covering of R. Let ι_j denote the feasibility-preserving function for link $j \in J_p$, i.e.,

$$\iota_j(x) = \begin{cases} 0 & \text{if } \sum_{r \in j} x_r \leq C_j \\ \infty & \text{otherwise.} \end{cases} \tag{7}$$

Definition 1. Let $1 \leq q \leq n$ and $C > 0$. Let $\theta = \{\theta_1, \dots, \theta_q\} \subset [1, n]$, $|\theta| = q$. We define $S_\theta(C)$, as the set:

$$S_\theta(C) = \{x \in \mathbf{R}^n_+, \sum_{i=1}^{q} x_{\theta_i} \leq C\}.$$

The vector z_θ is defined as $(z_\theta)_i = 0$ if $i \notin \theta$, and $(z_\theta)_i = z_i$ otherwise. Thus, the linear projection of $S_\theta(C)$ onto $\{z_\theta, z \in \mathbf{R}^n\}$ is a simplex of dimension q embedded in $\mathbf{R}^n$.

Thus, for each $j \in J$, $S_j \overset{\text{def}}{=} S_{R_j}(C_j) \subset \mathbf{R}^{|R|}$ is the capacity set of the link j. Finally, for $j \in J$, $\textsc{Projection}(j, x)$ denotes the Euclidean projection of x onto S_j.

4.2 Consensus form

We can now reformulate our objective to a fully separable form. For ease of notation, the variable x will be written z_0 and we define $R_0 = R$. We also define an additional variable, $\tilde{z}_r$, that will represent the consensus value of z_{0r} found for each route r over all the domains handling the route r. We write $I_r = \{q \in [0, P] \quad r \in R_q\}$ to design the set of domain indices (including index 0) which r belongs to. The number $n_{pr} = |J_r \cap J_p|$ defines the number of links belonging to route r within domain J_p. In the same fashion as in Section 3.3, we plug the feasibility constraints into the objective. Each constraint being now handled separately, we can formulate Problem (3), (4) as follows:

$$\min \sum_{r \in R} g_r(z_{0r}) + \sum_{j \in J} \iota_j(z_0) \tag{8}$$

In order to obtain a separable objective and fully benefit from the separability property in Fact 1, we artificially create a copy of the variable z_0 for each link j. This variable will be handled by the unique domain J_p containing j. For each j, let $z_j \in \mathbf{R}^{|R|}$ be the copy of z_0 for link j.

The goal of the system is to agree on a common optimal value z_0, although each domain's information over z_0 relies on its own copy only. This agreement is enforced by adding a *consensus constraint* over all the copies. We can thus formulate our problem in matricial form under these conditions by extending the variable space to $\mathbf{R}^{|J||R|}$:

$$\min F(Z_0) + \sum_{j \in j} \iota_j(z_j) \tag{9}$$

$$\text{s.t.} \quad Z_0 - Z = 0 \tag{10}$$

$$Z = (z_1, \ldots, z_{|J|}), \quad Z_0 = (z_{01}, \ldots, z_{0|J|})$$

where $F(z_0, \ldots, z_0) = g(z_0)$ and $\text{dom}(F) = \{Z_0; z_{01} = \ldots = z_{0|J|}\}$. Applied to this formulation, ADMM update rules become parallelized in the sets R and in J. Creating a complete copy of all the variables for each domain is, nevertheless, of no use. Each domain indeed only needs information and manipulation over the only variables associated with the routes that they handle completely (the route is included in the domain's links) or partially (the route meets other domains). The variable z_0 does not need to be centralized or transmitted between controllers. Each domain controller may actually have a copy z_0 and perform the (low-cost) computation of their update rule (see line 9 in Algorithm 2 below) locally. Now, ι_j actually depends only on the sub-variable $(z_j)_{R_j} \stackrel{\text{def}}{=} (z_{jr})_{r \in R_j}$. We erase all the information that is irrelevant to region J_p: $z_j \in \mathbf{R}^{R_j}$. We can thus write the objective as follows:

$$\min G(z) = \sum_{r \in R} g_r(z_{0r}) + \sum_{j \in J} \iota_j((z_j)_{R_j}). \tag{11}$$

To sum up, we have artificially separated the objective function by creating a minimal number of copies of the primal variable z_0 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. To enforce an intra- (local) and inter- (global) domain consistent value of the appropriate allocation, consensus constraints are added to the problem. This new formulation can be interpreted as a multi-agent consensus problem formulation where route r has cost g_r, and link j has cost ι_j. As we separated the global objective on purpose, the separability property of the proximal operator thus gives the following:

$$\text{prox}_{\lambda G}(u) = \left((\text{prox}_{\lambda g_r}(u_{0r}))_r, (\textsc{Projection}(j, u_j))_j\right).$$

These considerations permit next to write our final efficient distributed consensus model where each agent only has access to local information.

4.3 Fast Distributed ADMM

Taking into account the respective dimension of each variable, the general consensus form of the problem can be expressed as:

$$\min \sum_{r \in R} g_r(z_{0r}) + \sum_{j \in J} \iota_j(z_j) \tag{12}$$

$$z_{jr} = z_{lr} \quad \forall r \in R_j \cap R_l \quad \forall j, l \in \{0\} \cup J \tag{13}$$

where $z_j = (z_{jr})_{r \in R_j} \in \mathbf{R}_+^{|R_j|}$. Applying ADMM to this formulation and using again Fact 1 gives, after simplification, Algorithm 2 (Fast Distributed (FD)-ADMM). After each iteration count k of the algorithm, each domain J_p receives minimal information from other domains based on which one is able to compute a local consensus value z_{pr}^{k+1} and a locally feasible value z_{*pr}^{k+1} to next send back to domains within I_r for all $r \in R_p$. To understand the formula for updating the consensus variables, one should notice that, the Euclidean projection of a point $y \in \mathbf{R}^n$ onto

Algorithm 2 Fast Distributed ADMM (FD-ADMM)

1: **procedure of** Domain p
2: Receive $z^k_{qr}, z^k_{*qr} \quad \forall q \in I_r \quad \forall r \in R_p$
3: Enforce $z^k_{*r} = \min_{q \in I_r} z^k_{*qr} \quad \forall r \in R_p$
4: $\tilde{z}^{k+1}_r = \frac{1}{|J_r|+1}\left(\sum_{q \in I_r} n_{qr} z^k_{qr} + z^k_{0r}\right) \quad \forall r \in R_p$
5: **for** $j \in J_p \cup \{0\}$ **do**
6: $u^{k+1}_{jr} = u^k_{jr} + z^k_{jr} - \tilde{z}^{k+1}_r \quad \forall r \in R_j$
7: $z^{k+1}_j = \text{Projection}(j, \tilde{z}^k - u^k_j)$
8: **end for**
9: $z^{k+1}_{0r} = \text{prox}_{\lambda g_r}(\tilde{z}^k_r - u^k_{0r}) \quad \forall r \in R_p$
10: Send $z^{k+1}_{pr} = \frac{1}{n_{pr}} \sum_{j \in J_r \cap J_p} z^{k+1}_{jr}$ and $z^{k+1}_{*pr} = \min_{j \in J_p} z^{k+1}_{jr}$ to domains $q \in I_r \quad \forall r \in R_p$
11: **end procedure**

the set $\{v_1 = v_2 = \ldots = v_n\}$ is simply $\frac{1}{n}\sum y_i$. Hence, if I denotes the indicator function of the feasible set (13), we have:

$$\forall r \in R \quad \text{prox}_{\lambda I}(u)_r = \frac{1}{|J_r|+1}\left(\sum_{l \in r} u_{lr} + u_{0r}\right).$$

This gives the simple update rules at lines 4 and 10. These updates rules are also simplified using the fact that the sum $\sum_{j \in r} u^k_{jr}$ is constant. It can thus be fixed to 0 by initialization. As Proposition 1 below will show it, the domain p can at each iteration k enforce a *globally* feasible allocation z^k_{*r} for each of the routes $r \in R_p$.

In FD-ADMM, the domains that do not share any route together do not have to communicate. The only communication procedures among the domain controllers are described at lines 2 and 10. In these steps, the domains gather from and broadcast to adjacent domains the sole information related to routes that they share in common. In particular, domains are blind to routes that do not traverse them, and can keep their internal routes secret from others.

In terms of overhead, we can easily evaluate the number of floats transmitted between each domain at each iteration. At each communication, domain J_p must transmit z_{pr} and z_{*pr} for each $r \in R_p$ to each other domain that r traverses. Hence, domain p transmits in total $2\sum_{q \neq p} |R_p \cap R_q|$ floats to the set of its peers. As a comparison, in a distributed implementation of the algorithm given in [12] and stated in Section 6, each domain p transmits in total $\sum_{q \neq p} |\{j \in J_p, \exists r \in R_q \text{ s.t. } j \in r\}|$ floats to the set of its peers, which is bounded by $(P-1)|J_p|$ as $|R|$ grows. Hence, potential drawbacks of this approach could include overhead or the potential feasibility violation by the iterate $\tilde{z}^k$. However, we have the following positive result.

Proposition 1. *FD-ADMM provides a sequence of feasible points that converges to the optimum.*

Proof. Consider the iteration number k and drop the superscript k for lightness. For any link j, we have by line 7 of Algorithm 2 that z_j is feasible in link j. That is, $\sum_{r \in j} z_{jr} \leq C_j$. Define $z_{*r} = \min_{j \in J_r} z_{jr}$. Then, for each link j:

$$\sum_{r \in j} z_{*r} \leq \sum_{r \in j} z_{jr} \leq C_j. \tag{14}$$

Thus, no capacity is violated by the allocation z_{*r}. At the optimum, the consensus is reached. Thus $(z^k_{*r})_k$ is a feasible sequence that converges to the optimum. □

The number z_{*r} introduced in Proposition 1 above in fact corresponds to the introduced variable of the same name in FD-ADMM. Thus, in a certain way, for sufficiently loaded and communicating domains (i.e. the $|R_p \cap R_q|$ 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.

5 Implementation and algorithm tuning

In this section, we discuss two major points in the design of FD-ADMM. First, we precise and justify the choice of the procedure PROJECTION. Next, we derive an explicit adaptive update of the reciprocal penalty parameter λ that permits to accelerate the convergence of FD-ADMM on any instance.

5.1 On the Euclidean projection onto a simplex

In Section 4 we advocated a link-wise separation of the formulation because it is non-trivial to project an arbitrary point onto an arbitrary closed convex polyhedron. After all, SDN domains operating in parallel could each apply central projections onto their local capacity set and obtain a locally feasible allocation. This procedure would force the global resource allocation to wait, at each iteration, for the parallel projections execution that can take time. However, the projection onto a simplex can be done with an exact method whose complexity is dominated by the one of sorting a list of the size of its dimension. In average, sorting a list of length q is done in $O(q \log q)$. Hence, by operating instead a link-by-link projection, the controllers save a huge amount of time by providing an (generically infeasible) approximate projection point z_{pr} and deriving a locally feasible allocation z_{*pr} (see Algorithm 2 line 10). Although the quality of the global iterate z_* may be altered by further distribution of the projection, the point is quickly generated. Paradoxically enough, FD-ADMM therefore fully adapts to any network distribution into domains *because* it functions by link, regardless of the network partition into domains. The algorithm we use for PROJECTION in FD-ADMM is presented for instance in [4] in which the authors also give a correctness proof and performance demonstration. It permits to provide an efficient update for each domain J_p.

5.2 Estimating the optimal parameter λ

It is well-known that the reciprocal penalty parameter λ highly conditions the convergence speed of ADMM. An inaccurate tuning can indeed lead to a very slow convergence. For appropriate problems, it is possible to use a result proven in [5] to compute an optimal reciprocal penalty parameter. Before stating it, we remind the following.

Definition 2. Let $f : \mathbf{R}^n \to \bar{\mathbf{R}}$ be a differentiable function. Then, f is *strongly convex* with modulus σ if

$$(\nabla f(x) - \nabla f(y))^{\mathrm{T}}(x - y) \geq \sigma ||x - y||^2, \quad \forall x, y \in \mathrm{dom}(f).$$

Definition 3. Let $f : \mathbf{R}^n \to \bar{\mathbf{R}}$ be a differentiable function. Then, f is *Lipschitz* with modulus L if

$$|f(x) - f(y)| \leq L|x - y|, \quad \forall x, y \in \mathrm{dom}(f).$$

We are now ready to state the result in [5] which will help us tune FD-ADMM to optimize its convergence performance.

Theorem 1 ([5]). *Assume that the following problem:*

$$\begin{aligned} &\min F(x) + G(z) \\ &s.t.\ Mx - Pz = 0 \end{aligned} \tag{M}$$

has a saddle point, and both objective functions are convex. Assume that M has full row rank, and that F is σ-strongly convex and has a L-Lipschitz gradient. Then, the sequence of iterates (primal and dual concatenated) of ADMM converges linearly with rate $(1+\delta)^{-1}$, where[2]

$$\delta = 2\left(\frac{||M||^2}{\lambda\sigma} + \frac{L\lambda}{\lambda_{\min}(M^{\mathrm{T}}M)}\right)^{-1}$$

and $\frac{1}{\lambda}$ is the penalty parameter in the augmented Lagrangian form (see Section 3.3).
Corollary: *The optimal reciprocal penalty parameter is*

$$\lambda_* = \sqrt{\frac{||M||^2\lambda_{\min}(M^{\mathrm{T}}M)}{\sigma L}}.$$

Unfortunately, the result above cannot directly be applied to our general consensus formulation. Indeed, its matricial formulation does not provide a full-row rank matrix M. The problem to which the theorem applies is actually the centralized problem (3) - (4). We therefore derive a reciprocal penalty parameter selection for the centralized problem, and use it as a tool to estimate a satisfactory parameter for FD-ADMM. The following fact expresses the coefficients of interest in our problem.

Fact 2. *The function $g = \bigoplus_r g_r$ is σ-strongly convex and has L_d-Lipschitz gradient with:*

$$\sigma = \alpha \min_r \frac{w_r}{B_r^{\alpha+1}}, \quad L_d = \alpha \max_r \frac{w_r}{d_r^{\alpha+1}}$$

on any compact subset of $\mathrm{dom}(g)$ of the form $K_d = \{x \geq d, Ax \leq C\}$, for $d \gg 0$, and $B_r = \min_{j\in r} C_j$.

Proof. Consider the case $g = g_r$. We start with the calculus of σ. We write $g(x) = -w\frac{x^{1-\alpha}}{1-\alpha}$, when $\alpha \neq 1$, and $g(x) = -w\log(x)$ when $\alpha = 1$. Suppose $\alpha > 1$. For $y < x \in \mathrm{dom}(g)$, we have:

$$\begin{aligned} (\nabla g(x) - \nabla g(y))(x-y) &= w(\frac{1}{y^\alpha} - \frac{1}{x^\alpha})(x-y) \\ &= \frac{w}{x^\alpha y^\alpha}(x^\alpha - y^\alpha)(x-y) \\ &= \frac{w}{x^\alpha y^\alpha}\alpha c^{\alpha-1}(x-y)^2), \quad (y < c < x) \\ &= \frac{\alpha w}{x^\alpha y}\left(\frac{c}{y}\right)^{\alpha-1}|x-y|^2 \\ &\geq \frac{\alpha w}{x^\alpha y}|x-y|^2 \\ &\geq \frac{\alpha w}{t^{\alpha+1}}|x-y|^2, \end{aligned}$$

where the third equality is just an application of the mean value theorem. The case $\alpha < 1$ is handled likewise by integrating $x^{\alpha-1}$ into the parenthesis instead of $y^{\alpha-1}$ in the fourth line. The

[2] $\lambda_{\min}$ is the smallest eigenvalue of a positive matrix, and $||M||$ is the operator norm

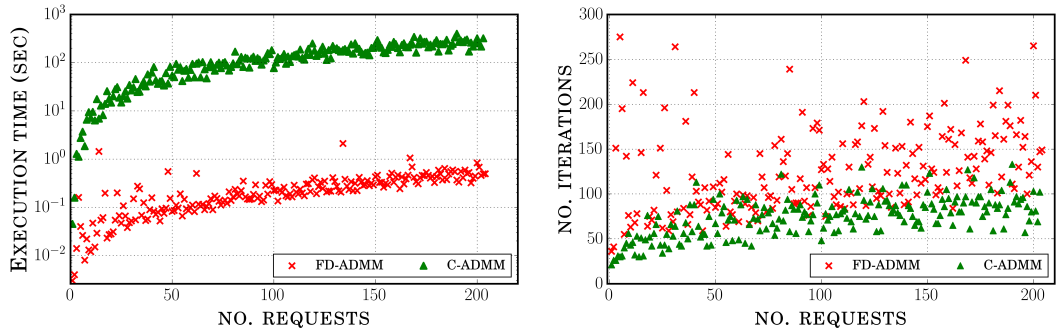


Figure 1: C-ADMM against FD-ADMM: execution time and iteration count.

case $\alpha = 1$ is straightforward. By plugging in x, y an appropriate sequence, say, respectively, $t - \frac{1}{n}$ and $t - \frac{2}{n}$, one can see that this bound is tight.

As for the Lipschitz factor, similarly, take $s < y < x$, we have:

$$\begin{aligned} |\nabla g(x) - \nabla g(y)| &= w|\frac{1}{y^\alpha} - \frac{1}{x^\alpha}| \\ &= \frac{w}{x^\alpha y^\alpha}|x^\alpha - y^\alpha| \\ &= \frac{w}{x^\alpha y^\alpha}\alpha c^{\alpha-1}|x - y| \\ &\qquad (y < c < x) \\ &\leq \frac{\alpha w}{s^{\alpha+1}}|x - y|, \end{aligned}$$

where the last line is obtained in the same fashion as for the calculus of σ, for each case $\alpha < 1$, $\alpha > 1$. The case $\alpha = 1$ is straightforward. Now, consider $g = \bigoplus w_r g_r$. For $x, y \in \mathrm{dom}(g)$,

$$\begin{aligned} & (\nabla g(x) - \nabla g(y))^{\mathrm{T}} (x - y) \\ = & \sum_r (\nabla g_r(x_r) - \nabla g_r(y_r))(x_r - y_r) \\ \geq & \alpha \sum_r \frac{w_r}{B_r^{\alpha+1}}|x_r - y_r|^2 \\ \geq & \alpha \min_r \frac{w_r}{B_r^{\alpha+1}} \sum_r |x_r - y_r|^2 \\ = & \alpha \min_r \frac{w_r}{B_r^{\alpha+1}}||x - y||^2. \end{aligned}$$

The derivation of L_d is similar. □

The last difficulty in choosing the optimal reciprocal penalty parameter for our problem is thus to correctly evaluate the Lipschitz modulus. One should keep in mind that ∇g is Lipschitz on any compact subset of the form $K_d, d \gg 0$, and *not globally* on the feasible set. We call d a *disagreement point* according to *bargaining theory* terminology. A disagreement point d represents the minimal values for an allocation of each route, that is, when the feasible set is reduced to the form K_d defined above. A natural choice of a disagreement point could be the feasible point z_* that the first iterations provide. Generically, there is *a priori* no guarantee that

the set K_{z^*} contains the optimum, but, we remark that at least in the first iterations, the use of z_* provides a good approximation of the best reciprocal penalty parameter. The analytical evaluation of this phenemon goes beyond the scope of this paper and we keep it for future work. Thus, we update λ in an adaptive fashion in the beginning of the algorithm with the help of those points. We remark that, at the very least in our case study, operating this update several times in the beginning and then fixing λ for the rest of the execution provides a good performance in terms of optimal convergence speed. In the next section, an appreciation of the typical phenomenon is illustrated (Figure 2). In all our simulations, we use the simple following update scheme to estimate the optimal penalty parameter at each execution of the algorithm.

Penalty adaptation scheme. At each iteration number below a threshold τ, denote by p the last output of a feasible point. Then the new reciprocal penalty parameter we choose is:

$$\lambda_* = \frac{1}{\alpha}\left(\min_{r\in R}\frac{w_r}{B_r^{\alpha+1}}\max_{r\in R}\frac{w_r}{p_r^{\alpha+1}}\right)^{-\frac{1}{2}}.$$

After τ iterations, do not update λ.

In our numerical evaluations we will set $\tau = 30$. Thus, FD-ADMM is now fully tuned and we are ready to demonstrate its performance in the next section, in terms of convergence speed in real-time scenarios.

6 Performance analysis

We now evaluate numerically FD-ADMM in terms of its convergence properties. More specifically, in Section 6.1 we compare the performance of FD-ADMM and C-ADMM in offline scenarios where the optimum is desired. In Section 6.2 we evaluate FD-ADMM in real-time scenarios, where good and feasible solutions are needed on-the-fly as weights w_r vary over time. In order to benchmark the transient properties of FD-ADMM we use the standard Lagrangian dual decomposition approach (LAGR) for single-path routing in [21, 12, 16], that we recall in Algorithm 3. We here assume that domain controllers operate in synchronous mode. In this case, the decomposition into domains has no impact on FD-ADMM performance, as projection is on a link-basis. All simulations are made for the proportional fairness objective functions ($\alpha = 1$). In this case, one has:

$$\begin{aligned} x = \text{prox}_{\lambda f^1}(u) &\Leftrightarrow x \geq 0 \text{ and } x^2 - ux - \lambda w = 0 \\ &\Leftrightarrow x = \frac{u + \sqrt{u^2 + 4\lambda w}}{2}. \end{aligned}$$

The algorithms under investigation were evaluated using BT's 21 CN network topology[3], containing 106 nodes and 474 links. The requests were generated by computing the shortest path between randomly chosen sources and destinations.

6.1 Algorithm design

Evaluating the alleviation of the compute-intensive parts of C-ADMM was a key concern to motivate and validate the distribution to FD-ADMM. To this aim, we show in Figure 1 the computation time and iteration count for those two algorithms on small instances for a number

[3]We would like to thank the authors of [12] for their willingness to share the BT 21 CN topology dataset.

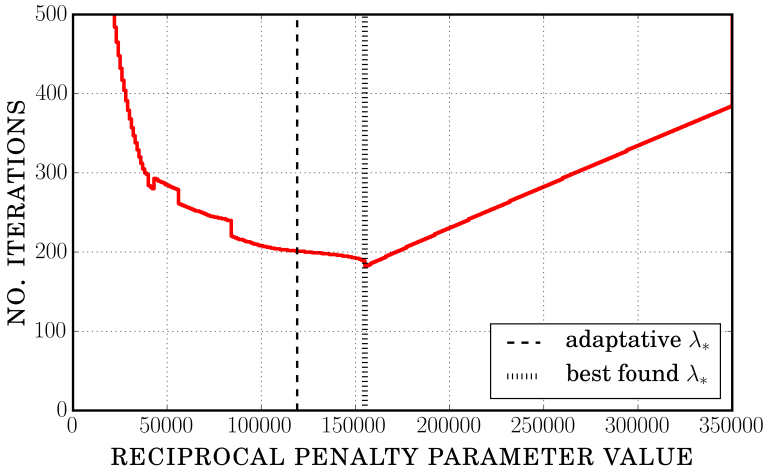


Figure 2: Convergence rate of FD-ADMM vs. reciprocal penalty parameter. The adaptive approximation demonstrates sufficient accuracy.

of requests ranging from 1 to 200. The centralized projection in C-ADMM is executed using the variation of Hildreth's projection algorithm on general polyhedra in [8]. When convergence is desired, a precise stopping criterion for FD-ADMM is available, as the optimality gap can be upper-bounded by the primal and dual residuals, see [2]. In our case, evaluating those residuals results in computing the absolute variation of two consecutive values of $\tilde{z}$, and the consensus accuracy[4] $\max_{r\in R_j}^{j\in J} |z_{jr} - \tilde{z}_r|$. This is a first advantage for FD-ADMM implementation as no robust stopping criterion is available for standard gradient descent. When an optimality gap is computed, we thus consider a 10^{-6}-approximation by FD-ADMM as the reference for all tested algorithms. In Figure 2, we illustrated, on a small instance with 200 requests, the number of iterations of FD-ADMM to reach convergence for a various number of the parameter values, in order to evaluate our *adaptive* scheme's accuracy with respect to the empirically *best found* parameter. It shows that our approximation of λ_* is fairly satisfactory. In Figure 1, FD-ADMM shows that distributing the consensus over the links exchanges several more iterations for a reduction of the compute time by two orders of magnitude for small instances. Hence, the distribution does not seem to cost too much convergence rate. Not surprisingly, the use of a central projection sub-routine makes C-ADMM impossible to scale. The convergence criterion used in Figures 1 and 3 is modest (10^{-1}). Finally, we plotted a notable behavior of FD-ADMM in Figure 3. One can imagine a link between the convergence rate and the mean link load, i.e.,

[4]One can choose any other norm in $\bigoplus \mathbf{R}^{|R_j|}$.

Algorithm 3 Lagrangian-based gradient descent (LAGR)

Input: Initial positive values $u_j(0)$
while a suitable termination condition is not met **do**
$x_r(t) = \arg\max_{x_r \geq 0} \{f_r(x_r) - x_r \sum_{j:j\in r} u_j(t)\} \quad \forall r$
$u_j(t+1) = u_j(t) - \frac{u_j(t)}{2C_j}(C_j - \sum_{r:j\in r} x_r(t)) \quad \forall j$
$t \leftarrow t+1$
end while

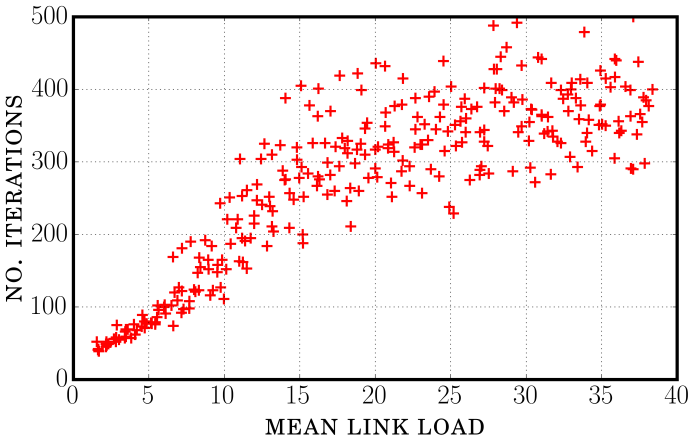


Figure 3: Iteration count for FD-ADMM vs the mean link load (average value of the $|R_j|$).

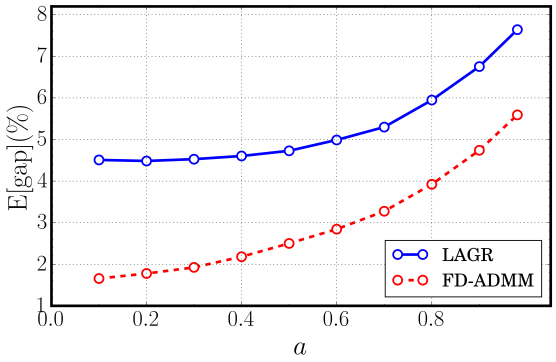


Figure 4: Average optimality gap E[gap] vs. the variation amplitude a.

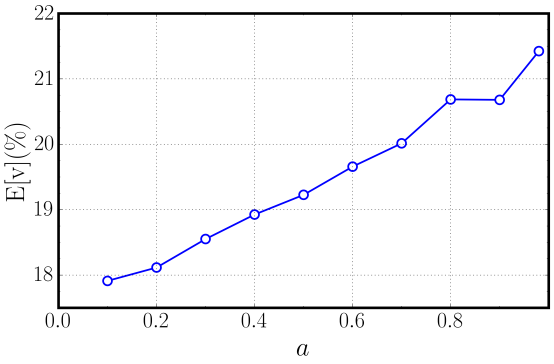


Figure 5: Average percentage of violated constraints E[v] by LAGR vs. the variation amplitude a.

$\frac{1}{|J|}\sum|R_j|$. This conjecture requires further investigation that we keep for future work.

6.2 Comparison against Lagrangian method

We now compare the proposed FD-ADMM algorithm against the classic LAGR Algorithm 3, see [21, 12, 16]. To this aim we perform two experiments, in real-time and static scenarios, respectively.

We start by evaluating the real-time responsiveness of FD-ADMM by considering a small scenario where 200 routes are established and the weights $(w_r^t)_{r\in R, t\in 0\dots T}$ vary over discrete time t, following the formula:

$$w_r^{t+1} \in [(1-a)w_r^t, (1+a)w_r^t] \quad a \in [0,1],$$

where at each event t, w_r^t is chosen uniformly within the above interval in which a determines the amplitude of the weight variation. In Figure 4 we illustrated the average optimality gap of the two algorithms achieved over 20 events with 10 iterations between each event. We observe that FD-ADMM outperforms LAGR in terms of optimality gap, although the performance of both algorithms is fairly acceptable. However, remarkably, FD-ADMM remains always feasible

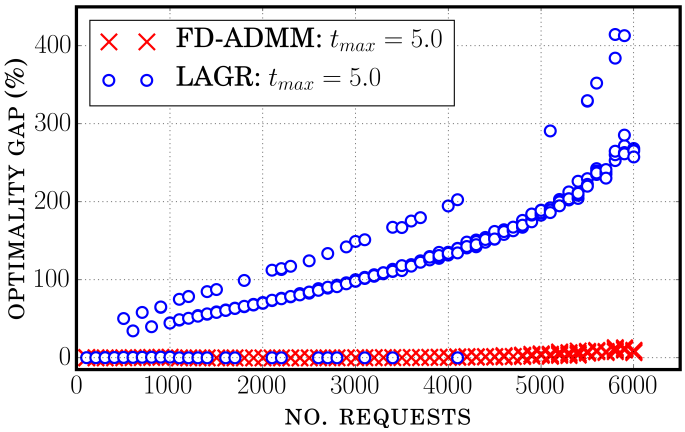


Figure 6: Optimality gap of the best feasible point found after 5 seconds runtime.

whereas LAGR constantly violates the constraints as weights w_r change in real-time. Figure 5 shows the percentage of constraints of the problem that are violated for each value of the amplitude a. In fact, LAGR iteratively approaches the fair resource allocation from the outside of the feasible set. This drawback is commonly amended by projecting the solution onto the feasible set. However, this is not doable in our distributed setting, as projection requires costly on-the-fly operations that require full topological information. For such reasons, we claim that the standard LAGR algorithm is not well suited for computing real-time fair allocations in a distributed SDN setting.

In our last experiment we test the two algorithms under a static scenario, where the weights w_r do not vary over time and LAGR has enough time to find at least one feasible solution. In Figure 6 we compare the optimality gap of the *best feasible* solutions found after 5 seconds runtime by FD-ADMM and LAGR, for different instance sizes over BT topology. We observe that FD-ADMM obtains a close-to-optimal feasible solution for all the instance sizes (from 100 to 6000 requests), while LAGR is still far from the optimum especially when the instance becomes large.

To recap, in this section we have demonstrated by experimentation that FD-ADMM reacts quickly to unpredictable network variations, while preserving the feasibility of the solutions computed iteratively. We then claim that FD-ADMM is a good candidate for real-time fair resource allocation in distributed SDN scenarios.

7 Conclusions and future work

In this paper we addressed the real-time fair resource allocation problem in the context of a distributed SDN control plane architecture. Our main contribution is the design of a distributed algorithm that continuously generates a sequence of feasible solutions and adapts to any partitioning of the network into domains. We reformulated the α-fair resource allocation problem in the fashion of a general consensus problem to derive the FD-ADMM algorithm. This algorithm can be massively parallelized on several processors that manage different regions of the network, hence fully benefiting from the computing resources of SDN controllers in distributed architectures. We also provided a strategy for a sufficiently accurate estimation of the penalty parameter of FD-ADMM that boosts its convergence. Finally, we compared FD-ADMM to a

standard dual Lagrangian decomposition method (LAGR) and we demonstrated how the former is more adapted to a real-time situation where bandwidth has to be adjusted on-the-fly. In fact, FD-ADMM ensures a smaller optimality gap since the very first iterations and, most importantly, it produces a feasible fair allocation at all iterations.

As a next step, we envision to adapt FD-ADMM to the case where multiple candidate paths are available for each request. We also would like to bound the number of bandwidth adjustments over time, as flow programming is often slow on hardware equipment. Additionally, the convergence properties of FD-ADMM require further investigations to possibly reach a result comparable to the linear convergence property of C-ADMM. Finally, it would be interesting to derive efficient calculus for the case of general α-fairness.

References

[1] Dimitri P Bertsekas, Robert G Gallager, and Pierre Humblet. *Data networks*, volume 2. Prentice-Hall International Series, 1992.

[2] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[3] Anna Charny, Raj Jain, and David Clark. Congestion control with explicit rate indication. In *Proc. of IEEE ICC*, 1995.

[4] Yunmei Chen and Xiaojing Ye. Projection onto a simplex. *arXiv preprint arXiv:1101.6081*, 2011.

[5] Wei Deng and Wotao Yin. On the global and linear convergence of the generalized alternating direction method of multipliers. *Journal of Scientific Computing*, 66(3):889–916, 2016.

[6] Soheil Hassas Yeganeh and Yashar Ganjali. Kandoo: a framework for efficient and scalable offloading of control applications. In *Proc. of ACM HotSDN*, 2012.

[7] Bingsheng He and Xiaoming Yuan. On the o(1/n) convergence rate of the douglas-rachford alternating direction method. *SIAM J. Numer. Anal.*, 50(2):700–709, April 2012.

[8] Alfredo N Iusem and Alvaro R De Pierro. A simultaneous iterative method for computing projections on polyhedra. *SIAM Journal on Control and Optimization*, 25(1):231–243, 1987.

[9] Frank P Kelly, Aman K Maulloo, and David KH Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society*, 49(3):237–252, 1998.

[10] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proc. of the IEEE*, 103(1):14–76, 2015.

[11] Jelena Marasevic, Clifford Stein, and Gil Zussman. A fast distributed stateless algorithm for alpha-fair packing problems. In *Proc. of ICALP*, volume 55, pages 54–1, 2016.

[12] Bill McCormick, Frank Kelly, Patrice Plante, Paul Gunning, and Peter Ashwood-Smith. Real time alpha-fairness based traffic engineering. In *Proc. of ACM HotSDN*, pages 199–200, 2014.

[13] Jeonghoon Mo and Jean Walrand. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking (ToN)*, 8(5):556–567, 2000.

[14] João FC Mota, João MF Xavier, Pedro MQ Aguiar, and Markus Puschel. Distributed admm for model predictive control and congestion control. In *Proc. of IEEE CDC*, 2012.

[15] Udayasree Palle, Dhruv Dhody, Ravi Singh, Luyuan Fang, and Rakesh Gandhi. PCEP Extensions for MPLS-TE LSP Automatic Bandwidth Adjustment with Stateful PCE. Internet-Draft draft-dhody-pce-stateful-pce-auto-bandwidth-09, Internet Engineering Task Force, November 2016. Work in Progress.

[16] Daniel Pérez Palomar and Mung Chiang. A tutorial on decomposition methods for network utility maximization. *IEEE Journal on Selected Areas in Communications*, 24(8):1439–1451, 2006.

[17] Fabian Skivée and Guy Leduc. A distributed algorithm for weighted max-min fairness in MPLS networks. In *International Conference on Telecommunications*, pages 644–653. Springer, 2004.

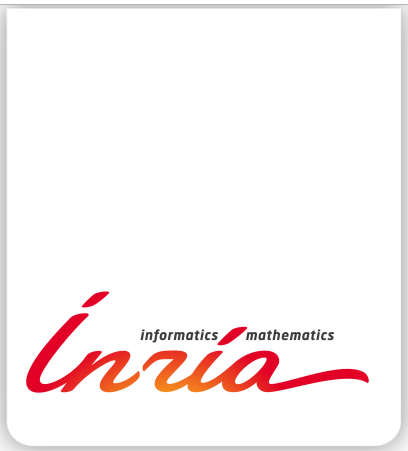
[18] William Stallings. Software-defined networks and openflow. *The internet protocol Journal*, 16(1):2–14, 2013.

[19] Rajesh Sundaresan et al. An iterative interior point network utility maximization algorithm. *arXiv preprint arXiv:1609.03194*, 2016.

[20] Steven J Vaughan-Nichols. Openflow: The next generation of the network? *Computer*, 44(8):13–15, 2011.

[21] Thomas Voice. Stability of multi-path dual congestion control algorithms. In *Proc. of Valuetools*, page 56. ACM, 2006.

Contents



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399