Real-time scalable algorithms for alpha-fair resource allocation in software defined networks

Zaïd Allybokus

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Algorithmes distribués dédiés au calcul de l'allocation alpha-équitable en temps réel dans les réseaux SDN

Zaïd ALLYBOKUS

Institut National de Recherche en Informatique et en Automatique (INRIA)

Présentée en vue de l'obtention du grade de docteur en Sciences et Technologies de l'Information et de la Communication (STIC)
d'Université Côte d'Azur
Dirigée par : Dr. Konstantin Avrachenkov
Co-encadrée par : Dr. Jérémie Leguay
Soutenue le : 11 juin 2019

Devant le jury, composé de :
Prof. Konstantin Avrachenkov, Directeur de thèse, INRIA Sophia Antipolis
Prof. Walid Ben-Ameur, Rapporteur, Télécom SudParis
Dr. Eric Gourdin, Examinateur, Orange Labs
Prof. Adlen Ksentini, Rapporteur, EURECOM
Dr. Jérémie Leguay, Co-directeur de thèse, Huawei Technologies
Dr. Lorenzo Maggi, Co-directeur de thèse, Nokia Bell Labs
Prof. Guillaume Urvoy-Keller, Examinateur, Université de Nice Sophia Antipolis

anrt association nationale recherche technologie

Inria

HUAWEI

UNIVERSITÉ CÔTE D'AZUR

INVENTEURS DU MONDE NUMÉRIQUE

DISSERTATION

in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy
from University of Nice Sophia Antipolis

Zaïd Allybokus

Real-Time Scalable Algorithms for Alpha-Fair Resource Allocation in Software Defined Networks

Thesis to be defended on the 11th of June, 2019
before a committee composed of:

Reporters	Prof. Walid Ben-Ameur (Télécom SudParis, France)
	Prof. Adlen Ksentini (EURECOM, France)
Examiners	Dr. Eric Gourdin (Orange Labs, France)
	Prof. Guillaume Urvoy Keller
	(Université de Nice Sophia Antipolis, France)
Thesis Director	Prof. Konstantin Avrachenkov
	(INRIA Sophia Antipolis, France)
Thesis Co-Directors	Dr. Jérémie Leguay (Huawei Technologies, France)
	Dr. Lorenzo Maggi (Nokia Bell Labs, France)

Abstract

In this dissertation, we deal with the design of algorithms to tackle the α-fair resource allocation problem in real-time and distributed Software-Defined Networks (SDN). First, we define three major requirements that picture the challenges of real-time algorithms implementable in modern distributed SDN controllers. Those challenges are the ability to provide feasible resource allocations at all times, good transient solutions in terms of optimality gap that converge in an acceptable number of inter-controller communication rounds, and their ability of being massively parallelized independently of the network architecture. We use the Alternating Directions Method of Multipliers to design an algorithm that simultaneously, and unprecedentedly, tackles the three challenges. Motivated by a first study of the structural properties of the α-fair model, where we derive a lower bound on the optimal solution, we tune the penalty parameter of the augmented Lagrangian of the problem in order to optimize the algorithm's performance. We show that the algorithm can function in real-time when the traffic requirements can vary more or less abruptly. The variation of the traffic requirements are modeled by real-time varying coefficients of the optimization model that is solved on-the-fly and may represent various prioritization policies of the traffic (payment, traffic type, number of connections within a tunnel, *etc*). Then, we describe how to extend the algorithm to real world use cases with limited modifications to cope with multi-path load balancing and online adjustments. Furthermore, we address the problem of α-fairness when the environment is uncertain and the available amount of resources over the network links is known only through general density functions. The main focus there is, instead of feasibility, the notion of safety. We design a heuristic that polishes an outer relaxation of the problem, based on the sensitivity analysis of the static problem. In general, we are able to provide a safe and acceptably efficient solution by solving several static problems.

Keywords: Software-Defined Networks, Resource Allocation, Alpha-Fairness, Real-Time, Distributed Algorithms, ADMM, Convex Optimization.

Résumé

Dans cette thèse, nous étudions la conception d'algorithmes dédiés au calul de l'allocation de ressources α-équitable en temps réel dans les réseaux Software-Defined Networks (SDN) distribués. En premier lieu, nous définissons trois besoins majeurs établissant les enjeux des algorithmes en temps réel implémentable dans les controlleurs distribués SDN. Ces enjeux sont la disponibilité de solutions faisables à tout moment, une qualité transitoire acceptable en termes d'écart à l'optimum, une convergence en un nombre raisonnable de tours de communications entre les differents controlleurs, ainsi qu'une facilité des algorithmes à être massivement parallèles, indépendamment de l'architecture SDN du réseau. Nous utilisons les outils de l'Alternating Directions Method of Multipliers afin de définir une classe d'algorithmes qui, sans précédent, répondent simultanément à ces enjeux. À la lumière des propriétés structurelles du modèle de l'allocation α-fair, nous calculons une borne inférieure sur la solution optimale et l'utilisons afin d'ajuster le paramètre de pénalité du Lagrangien augmenté du problème dans le but d'optimiser la performance des algorithmes. Nous montrons que l'algorithme est capable de fonctionner en temps réel lorsque les exigences du trafic varient de façon plus ou moins brute. La variation des exigences du trafic est modelisée par la variation en temps réel de certains coefficients du modèle d'optimisation qui est résolu à la volée. Ces coefficients représentent en pratique des politiques de priorité variées au sein du trafic (paiement, type de trafic, nombre de connections à l'intérieur d'un chemin, *etc*). Ensuite, nous décrivons comment étendre l'algorithme à des scenarios réels avec des modifications minimes, afin de prendre en compte l'équilibrage en multi-chemin des flots et l'ajustement de la bande passante en temps réel. Par ailleurs, nous répondons au problème de partage de ressources α-équitable lorsque l'environnement admet des incertitudes sur la quantité de ressources disponibles sur chaque lien, connue uniquement au travers de fonctions de densités générales. L'axe prioritaire est alors, au lieu de la faisabilité, la notion de fiabilité. Nous concevons alors une heuristique qui affine une approximation extérieure du problème en se basant sur l'analyse de sensibilité du problème statique. En toute généralité, nous arrivons à fournir une solution fiable et acceptable en termes d'efficacité en résolvant quelques problèmes statiques.

Mots-clés: Software-Defined Networks, Allocation de Ressources, Alpha-Equité, Temps Réel, Algorithmes Distribuées, ADMM, Optimisation Convexe.

To my parents...

Acknowledgments

The present work would never have been possible without my supervisor, Dr. Konstantin Avrachenkov, and my two co-advisors, Dr. Jérémie Leguay and Dr. Lorenzo Maggi. This is why, first and foremost, I would like to acknowledge my deepest gratitude and direct my warmest thanks to them. Your benevolent help and expertise have been the building blocks of this thesis that I had an extreme pleasure to write while working under your supervision.

I also wish to express my gratitude to all the colleagues who contributed greatly to making my working environment a happy place that I was always keen to go to in the morning. I am convinced that none of you need to be explicitely named in order to recognize yourselves.

Also, I would like to thank my family for their constant cheer all thorough those three years.

Thank you mother, for always believing in me. You are the person with the greatest indirect contribution to this work, as you always make me want to become a better version of myself.

Contents

List of Figures

Chapter 1

Introduction

SOFTWARE DEFINED NETWORKING (SDN) technologies are radically transforming the network architecture of data centers, network overlays, and carrier networks [1, 2, 3]. They provide programmable data planes that can be configured from a remote controller platform. They are implemented on top of commodity servers that provide a tremendous computational power compared to legacy network devices and use modern distributed computing technologies (e.g., loosely consistent databases, parallel computing, consensus algorithms, transactional updates) to gather a global view of the network status and push, in real-time, consistent configuration updates to network equipments. While initial SDN deployments focused mainly on automated provisioning of tunnels and virtual networks, essentially virtualizing known concepts, it becomes clear today that SDN presents an opportunity for a paradigm shift in network optimization. Departing from the classical autonomic routing systems we migrate to a new centralized routing paradigm, where a powerful controller is aware of the static parameters and time-varying conditions, and centrally orchestrates the entire network reaching unprecedented levels of efficiency. Indeed, the necessary ingredients are in place: (i) the power of modern SDN controllers, and (ii) the recent advances in optimization and machine learning to produce the desired outcome.

A typical success of deployment solutions for bringing SDN to the real-world is B4 [4], the world-wide Wide Area Network (WAN) of Google, among the largest in the Internet. It interconnects its data centers to one another to replicate data in real-time between individual campuses. Now, its network will reach the general public with Espresso [5], an extension of Google's SDN that will allow to choose from where to serve individual users and dynamically adjust paths and operate load balancing based on real-time measurements of how network connections are performing, and not on individual IP addresses and shortest paths as traditionally.

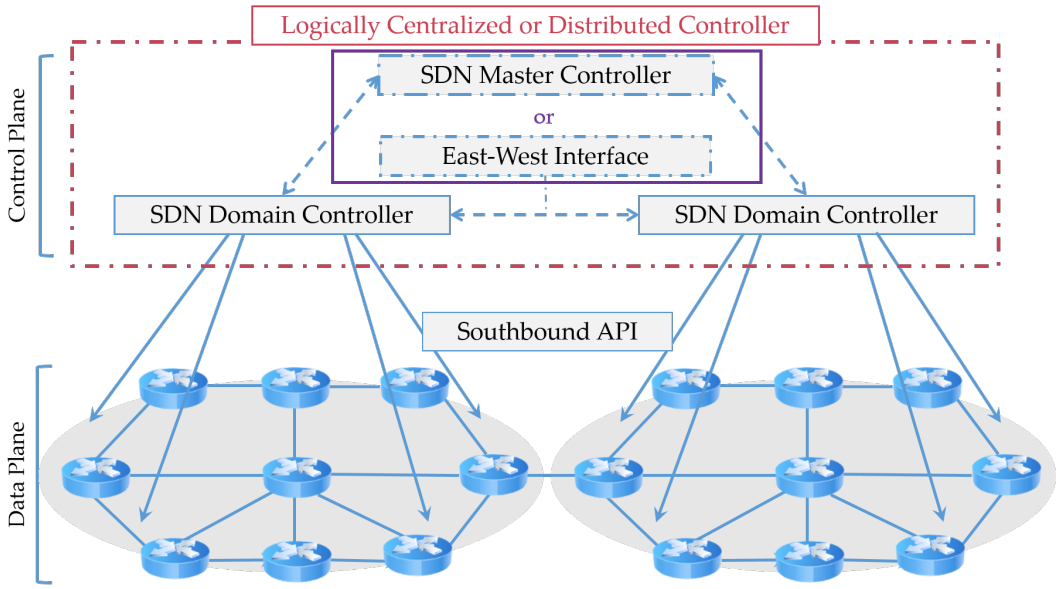


Figure 1.1 | Logically Centralized or Distributed SDN architecture.

1.1 Modern distributed SDN controllers

The computation power of SDN controllers fosters the development of a new generation of control plane architecture that uses compute-intensive operations.

The design of SDN architectures [6] envision the use of a central controller. However, the myth of *one* physical controller has long been interpreted as a fundamental limit to the ideological scaling and robustness of the SDN standard, as the ability of one controller to handle a growing number of requests and devices, hence its scalability has its own practical limits. Thus, obvious scalability questions [7], clearly imply that the SDN control plane, possibly *logically centralized*, is in fact partially distributed in large network scenarios [1]. Hence, in practice the control plane may consist of multiple controllers each in charge of an SDN domain consisting of a portion of the physical network that they each control. They thus operate together, in a *flat* [8] or *hierarchical* [9] architecture, to achieve global tasks. In Figure 1.1, an example of SDN architecture is illustrated. The design of general SDN architectures with multiple SDN controllers permits one to define different levels of logical distribution. In a logically centralized architecture, whether flat or hierarchical, the domain controllers operate together as workers in a distributed system, but push their decisions in a common database to always be consistent. A major drawback with this model is its requirement in terms of extensive and precise synchronization between all the sub-controllers (the domain controllers) through the SDN master controller (the centralized database). Many studies advocate [10] that this is not optimal for very large data center networks (DCNs) and WANs. Thus, a logically distributed architecture may also be helpful in highly distributed networks. Typically, the flat architecture (Figure 1.1) may

correspond to a logically distributed SDN control plane where the master controller is eliminated and replaced with individual databases at each controller level, in which case, East-West communication protocols are required to share relevent information between controllers. The East-West communication protocol is not yet standardized and represents today areas of active research [10, 11, 12].

1.2 Challenges

In distributed SDN architectures [13, 14, 12], each controller has full information about its own domain. In order to ensure fault-tolerance, each of them could be composed of master and slave agents that act as a single entity [15]. To maintain consistency, each controller can communicate with adjacent peer controllers and/or with a central, upper-layer controller entity. All in all, the multiple controller-based architecture permits to split the global workload of the control plane on a domain basis while always maintaining a global view of the network equipment and information such as topology, flows, traffic patterns. This unprecedented ability of software-based networks makes it now possible to dynamically control flows carrying traffic while fulfilling global objectives.

In this thesis, we are focused on the problem of bandwidth sharing between flows in SDN that can now be performed while optimizing *globally* joint objectives such as network efficiency, fairness between flows, robustness against failures, *etc*. The faculty of SDN to address such a problem globally lies beneath the ability of the distributed control plane to solve global optimization problems (general network utility maximization problems instances) with algorithms that can now be implemented and run on the controllers that can cooperate to solve them. Remarkably, an optimization problem that can be formalized and distributed following the actual distribution into network domains would require the domain controllers to solve local sub-problems (associated to their own domain), and to communicate together to achieve consensus on the variables that they have in common.

However, exchanges between controllers are expensive in terms of communication delay and overhead [14]. 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.* an optimal solution.

A second crucial property for a 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. Those changes fall under the notion of a *real-time* scenario. In this case, the convergence of a resource sharing strategy for the previous network state may not even be attained when a change occurs in the system. For this reason, especially in real-time scenarios, it is often preferable to have a quick access to a good quality solution rather than a provably asymptotically optimal solu-

tion with poor convergence rate. In particular, to address best the dynamic nature of the network state, it is highly desirable that the resource allocation computed by a distributed algorithm is *feasible*, that is, satisfies the physical constraints posed by the reality of limited resources, at any iteration.

Also, modern propositions of SDN controllers [16] rely on grid computing technologies such as Akka [17] or Hazelcast [18], respectively for the two major open source SDN controllers OpenDayLight [19] and ONOS (Open Network Operating System) [20]. As a consequence, massively parallelizable algorithms for SDN should be preferable as more adapted and better likely to tackle scalability issues.

1.3 Our resource allocation challenge

To recap, we identify three main requirements for a distributed algorithm for optimal resource allocation, namely:

1. *Real-time:* converging to a "good" solution in a small number of iterations,

2. *Feasibility:* producing *feasible* solutions at all iterations,

3. *Distributivity:* 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 [21] 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 2. and 3., but not 1., as it does not optimize resources globally. Also, classic primal-dual algorithms have been proposed to solve the general network utility maximization problem and might show as candidates in distributed SDN scenarios, as in [22]. However, primal-dual algorithms are known to fail at providing feasible solutions at any iteration step, thus they fail at achieving goal 2.

Recently in the optimization research community, the *Alternating Direction Method of Multipliers* (ADMM) [23] 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 1., 2. and 3. Indeed, this dissertation will 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.

In numerous use cases, it is advocated that a good fitness measurement for a resource allocation can take the form of a separable function formed by the aggregate sum of individual utilities of each flow. Separable functions are very prone to nearly unconditional distribution of optimization problems, as we will see in this dissertation. We study a particular instance of

the general network utility maximization problem dedicated to allocate resources while maximizing a certain separable objective: the α- fair resource allocation problem. The definition and presentation of this notion are carried out in Chapter 2. The major interest of this allocation notion is in its generality as it covers well-known notions of fairness (notably, *max-min fairness*, *minimum potential delay*, *proportional fairness*), encompassing them all into a general family of functions forming a spectrum of fairness metrics parameterized by the positive real number α.

In a first part of this thesis (Chapters 2– 4), as the reader may have guessed already, we use the framework of ADMM to build a model that carries out an answer to the question of how to efficiently compute distributively optimally real-time α-fair resource allocation strategies for a set of flows over a network, keeping in mind that this is now possible to operate with a global view of a network, unprecedentedly provided by emergent SDN technologies. The model is designed to tackle the fair resource allocation problem over a given, and fixed, network of links each having a fixed maximum capacity.

The notion of feasibility is central to this dissertation. Indeed, networks provide bandwidth to all types of traffic from a finite amount of available resources at each link called *capacity*. Therefore, efficient load balancing is critically dependent of active measurement of the available bandwidth on routes carrying out the traffic. A spectacular example is video streaming, that nowadays represents the largest portion of the Internet traffic, which perceived (by the user) quality is highly dependent on adaptive rate of service relying on measurement of the real-time network load. There is not yet a unified protocol for available bandwidth measurement in SDN and various packet probing techniques such as [24] are being proposed and can measure the available bandwidth on whole paths, or on a per-link basis.

In the definition of our real-time α-fair allocation model, we are therefore conducted to assume that the actual capacity of the links can be known by the SDN controller, only through those measurements. This is why in a second part (Chapter 5), we introduce the notion of uncertainties in the link capacities to account for their possible fluctuations in real-time. In this kind of scenario, it is common to see robust solutions that would find the best bandwidth sharing strategy under the worst-case scenario in order to guarantee strict feasibility. However, the conservative approach is often too pessimistic in practice and prone to overly deteriorated solutions, thus too much rigidity in feasibility can lead to excessively sub-optimal solutions in practice, and cause unnecessary service degradation. In this context, we use our algorithms to allocate bandwidth to flows while operating a trade-off between α-fairness and strict feasibility.

1.4 Thesis outline

We now outline the main parts of this thesis.

Chapter 2 is dedicated to an introduction on the concept of fairness in

resource sharing systems. We review the major steps that led to the construction and the definition of the notion of α-fairness. Further, we define the α-fair resource allocation problem as a convex optimization problem and study its structural properties. Specifically, on a generic instance of the α-fair resource allocation problem, we build a formula that corresponds to a lower bound on the allocation of each flow of the problem, that is, we derive a minimal value that each flow is guaranteed to get considering the structure of the instance in hand. This lower bound will show great practical interest in the design and tuning of our algorithms.

In Chapter 3, we build our algorithm that computes the α-fair allocation distributively. We show that the distribution method of our algorithm is compatible with any organization of the SDN controller (logically centralized or distributed) into domains. Our algorithm, FD-ADMM (Fast-Distributed ADMM), provides feasible iterates at any iteration and we show its suitability to real-time scenarios when the traffic instance (the objective function of the optimization problem only) varies on-the-fly, by comparing its performance to the one of a classic dual decomposition method.

Chapter 4 reviews some technical refinements of FD-ADMM and is organized in two parts. The first part discusses on an optimal tuning of FD-ADMM, which performances are highly dependent on the initial tuning of a parameter introduced in its design. In this part, we use the results from Chapter 2 to tune the algorithm and show that we obtain near-optimal performance in practice in terms of convergence rate. In a second part, we extend the model to two practical use cases. Firstly, the algorithm can address resource allocation instances where more than one path is available for each flow, while the algorithm will choose itself on which path to allocate bandwidth to them so as to maximize the overall fairness. Secondly, we introduce switching costs to be concatenated to the objective that permit the algorithm to jump from one allocation to another while not exceeding a bandwidth re-configuration budget (*i.e.*, a maximum number of bandwidth allocations that are allowed to be re-sized at the same time) in the case where the traffic requirements have varied from one epoch to another. These two optional extensions are presented as practical solutions to improve the network's efficiency by avoiding congested links (the multi-path setting) and to improve the stability for individual paths (the switching costs setting). In both cases, we show explicitely the changes in the update rules of the algorithm.

Lastly, Chapter 5 deals with the extension to the α-fair resource allocation problem with introduced uncertainties caused by the varying available capacities of the network links that are constantly evaluated in real-time by the SDN controller. In this case, it is assumed that the capacity of each link of the network can take values within a discrete set of possible numbers following a general probability distribution. We therefore introduce a model based on chance-constraints programming that permits to operate the desired trade-off between strict feasibility (robustness) and efficiency (fairness) in this situation. We observe that the chanced-constrained exten-

sion of our problem relies on finding an optimal set of constraints (in fact, an optimal scenario among all possible values that the link capacities can take) to activate among the set of all possible constraints, that finds a desired balance between feasibility preservation and fairness maximization. This problem can be cast as a minimization problem with convex objective but in all generality non-convex constraints and yields generally NP-hard problems. This is why we define a heuristic based on a sensitivity analysis of the ground problem defined in Chapter 2 that iteratively summons FD-ADMM to choose the best constraints to activate.

1.5 Symbols and notations

All thorough this dissertation, we denote by $\mathbf{R}$ the set of real numbers, $\mathbf{R}_+$ the set of non-negative real numbers, and $\mathbf{R}_{++}$ the set of positive real numbers.

The letter $\mathbf{P}$ designs a probability measure on the adequate measured space, and the letter $\mathbf{p}$ can either design a compactly supported probability measure on $\mathbf{R}$, or the associated density function. Otherwise, boldface letters are always reserved for matrices and vectors. For more clarity though, only the vector $\boldsymbol{\varepsilon}$ is written as $\underline{\varepsilon}$.

For a matrix $\mathbf{A} \in \mathbf{R}^{m\times n}$, we denote by A_{jp} its entry at line j and column p, where $1 \leqslant j \leqslant m$ and $1 \leqslant p \leqslant n$. The symbol $\mathbf{I}$ stands for the identity matrix of the appropriate dimension. The p-th entry of a vector $\mathbf{x} \in \mathbf{R}^n$ is written x_p. The vector of the appropriate dimension with all entries equal to 0 (respectively 1) is written $\mathbf{0}$ (respectively $\mathbf{1}$).

Sometimes, we consider elements of product spaces such as $S = S_1 \times \ldots \times S_k$, where typically, $S_i = \mathbf{R}^{n_i}$. A general vector $\boldsymbol{z} \in S$ can be sliced into k sub-vectors, each belonging to one of the S_i. Then, we write $\boldsymbol{z} = (\boldsymbol{z}_i)_{i=1\ldots k}$ with the implicit property that $\boldsymbol{z}_i \in S_i$, $i = 1, \ldots k$. Vectors of $\mathbf{R}^n$ are sometimes regarded as applications from $[\![1, n]\!]$ to $\mathbf{R}$ for more convenience. Therefore, for $V \subset [\![1, n]\!]$, $\boldsymbol{z} : [\![1, n]\!] \to \mathbf{R}$ and $\mathbf{x} : V \to \mathbf{R}$, the sum of $\mathbf{x}$ and $\boldsymbol{z}$ is defined as $\mathbf{x} + \boldsymbol{z} : V \to \mathbf{R}, i \mapsto x_i + z_i$. This permits to avoid too much heaviness in our formulas. Moreover, the proposition $\mathbf{x} \leqslant \mathbf{y}$ for two vectors is always interpreted component-wise.

We use the symbol $\coloneqq$ to define objects.

Chapter 2

Fair resource allocation: Main concepts and structure

THE CONCEPT OF fair resource allocation has been a central topic in networking. It was notably introduced and built as a way to formalize and address congestion control in fixed end-to-end connections through the Internet, but can also describe numerous use-cases such as power allocation in wireless networks [25, 26, 27], or be of a more theoretical interest in bargaining theory [28]. The question of how to allocate bandwidth to flows over a network *fairly* and *efficiently* is fundamental. The efficiency of a resource sharing policy is measured by the aggregate throughput of the network flows. The aggregate throughput of a network is simply the sum of the allocated bandwidth over all flows, and overall evaluates how well the network is serving its flows altogether. Fairness, on the contrary, is not necessarily a clearly defined notion. One is tempted to say that the share of a finite amount c of resource between n agents is "fair" if all the agents get an equal part of the resource. But this does not answer to the question (although obvious) of what should be preferred between the uniform allocations c/n and $c/2n$. For instance, imposing Pareto optimality of the allocation solves the question and characterizes the uniform allocation c/n as the most fair and efficient single resource sharing policy for multiple agents. Therefore, it is not enough to impose an equal allocation to all agents; one must also impose that the allocation should be efficient. Thereupon, the concept of fairness in networking cannot bypass the one of efficiency of the sharing policy. A popular resource sharing policy that brings, as much as possible, an equal share of bandwidth over a network of resources while enforcing Pareto optimality is *max-min fairness*. This notion will be defined later on and interpreted as the "ultimately fair" sharing policy. It is however known that in a large class of resource sharing problems[1], max-min fairness has the tendency to deteriorate considerably the overall throughput, and consequently, throughput and fairness can be

[1]This, however, is not always true although largely stated as fact in the literature, as shown in the simple counter example in [29].

interpreted as two conflicting concepts that a "fair" allocation policy should trade-off.

The fairness concept that is central to this thesis is α-fairness, and more generally, weighted $(\boldsymbol{w}, \alpha)$-fairness. Intuitively, it represents an allocation policy that encompasses both the ability of being "fair", and the one of being "efficient". As from now, let us avoid all ambiguities in the interpretation of the word "fair". We define by the word *equity* the ability to provide (as much as possible) the same amount of resources to all agents and in that sense, the concept accepts all the allocations of the form κ/n, for $\kappa \leqslant c$ in the above example. The word "fair" will be reserved for general $(\boldsymbol{w}, \alpha)$-fairness.

The α-fair resource allocation problem has received remarkable attention and has been studied in numerous application fields. This chapter is dedicated to introducing its concept in resource sharing systems. Fairness has been a key notion in the problem of resource sharing and has been conceptualized rigorously and studied extensively for the past decades in order to build answers to the question of equitable and efficient resource sharing. Different interpretations of what should "fair" mean brought a series of intuitive (yet sometimes contestable) axioms that have helped to build families of fairness notions, and in turn families of resource sharing policies. Certainly, the most popular axiomatization of a fairness notion is embodied in the *Nash Bargaining Solution* (NBS) to the *Bargaining Problem* [30]. The bargaining problem (historically introduced by Nash [31] for two players or bargainers) introduces the question of how two players should share a set of finite resources. Mathematically, a compact subset of $\mathbf{R}^2$ is given, and the question is how to find a point within this subset that would *simultaneously satisfy* (in the sense, for example, of maximizing their corresponding utility function U_i) both players as much as possible. Nash brought the answer, known as the NBS, that the desired point should maximize the product of the utility functions. Later, this solution was characterized as the only solution that satisfies the axiom of the midpoint domination, and the independence of irrelevant alternatives, bringing a more intuitive and humanly understandable definition of the policy, instead of only a mathematical formula. The NBS can also be defined and characterized in many other manners, that is with different sets of axioms (see, for instance, [32]).

More generally, a unified axiomatization framework was brought in by Lan et al. [33] and introduces a series of five axioms[2] that permit one to define numerous fairness measures, including Atkinson's index or Jain's index. Particularly, it turns out that α-fairness is a particular form of this family of fairness measures combining one *efficiency* factor and one *equity* factor which helps quantify the actual meaning of the trade-off between the two notions.

Although axiomatizations and general purpose algorithms have been

[2]Those five axioms are 1) the axiom of continuity, 2) the axiom of homogeneity, 3) the axiom of asymptotic saturation, 4) the axiom of irrelevance of partition and 5) the axiom of monotonicity. For a detailed presentation of the axioms and the fairness family they characterize, we refer the interested reader to the cited article.

studied and proposed for α-fairness, a few works have been done on its practical structure. Motivated by this, we provide a contribution on its structural properties and in particular give a novel lower bound on the $(\boldsymbol{w}, \alpha)$-fair share for a set of users competing for multiple resources. The lower bound gives a minimal fair-share that a user should get on a network of multiple resources for multiple users, and refines our understanding of its behavior and sensitivity. Our statement holds for $(\boldsymbol{w}, \alpha)$-fairness because it can be cast as an optimization problem with separable objective function and linear packing constraints. Particularly, we try to derive formulas that do not deteriorate with the size of the optimization problem. This means that we prefer formulas that do not contain global parameters such as total number of constraints (resources), of variables (users), maximum or minimum objective coefficients (weights)/right-hand-side (resource capacity), *etc*. On this type of problems, we introduce a localization property that permits one to better exploit local structures. Remarkably, for the case of *proportional fairness* ($\alpha = 1$), we give a local version of the midpoint domination axiom that represents a building block of the axiomatization of the NBS (equivalent to proportional fairness). We also propose a localized formula for our lower bound for values of α within $[0, 1]$. The formulas for general $(\alpha > 1)$-fair policies suffer from one global dependency to an instance's asymmetries, but contain no global parameter related to the size of the problem and still represent a considerable improvement with regards to existing works on α-fair lower bounds. Finally, we conjecture that an extended formula of the midpoint domination axiom to general $\alpha > 1$ provides a tighter lower bound on the α-fair resource allocation solution.

IN A NUTSHELL

In this chapter, we introduce the notion of fairness in networking. The notion of α-fairness as a resource allocation policy is presented, and the α-fair resource allocation problem is formalized as an optimization problem. We study its structure and specifically give a lower bound on the optimal allocation of each of the flows. Our lower bounds seek to improve existing formulas that could deteriorate too much with the size or complexity of the problem instance and is based on localization properties that permit one to eliminate the dependencies on global quantities.

2.1 Fairness concepts in resource sharing systems

We first aim at introducing a construction by the example in order to give the reader an intuitive understanding of the motivations underneath the definition of α-fairness. We illustrate the notion of α-fairness in a simple example, on which fairness embodies a continuous trade-off between the notion of equity and the one of efficiency. The first building block of our introduction method lies beneath the fundamental difference between the amount

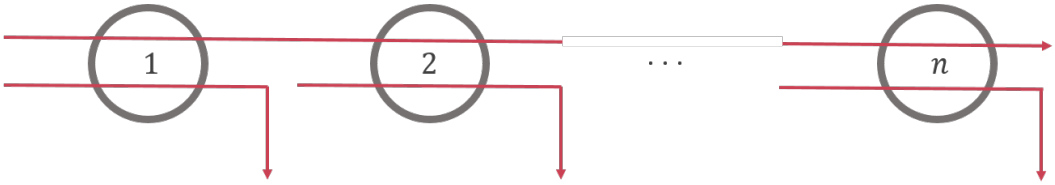


Figure 2.1 | The n-linear network: each of the n links serves an individual route, while all the links serve simultaneously one route.

of reserved physical resources and the actual amount of service it provides. Remarkably in network rate control topics, it is clear that an amount of service (that is, a bandwidth reservation along a path of network links) of x requires a quantity of resources equal to lx, where l is the length of the path (that is, the number of links it contains) through which we are allocating bandwidth. This is why very often, path establishments prefer shortest path algorithms, of course because other requirements such as maximum tolerable delay constraints should be satisfied and naturally favor the use of short paths, but also because longer paths have the obvious tendency to require too much resources over a network. Therefore, the length of paths can cause efficiency degradation and this is why a valid question should be how to simultaneously provide accurate service to all paths *and* avoid efficiency degradation. We will see that $(\boldsymbol{w}, \alpha)$-fairness naturally brings an answer to this question.

The weighted $(\boldsymbol{w}, \alpha)$-fair resource allocation problem, introduced by Mo and Walrand [34], is to find a vector $\boldsymbol{x}^* \in \mathbf{R}_+^n$ such that:

1) the utility

$$f^\alpha(\boldsymbol{w}, \boldsymbol{x}) = \begin{cases} \sum_{i=1}^n w_i \frac{x_i^{1-\alpha}}{1-\alpha}, & \alpha \neq 1, \\ \sum_{i=1}^n w_i \log(x_i), & \alpha = 1 \end{cases}$$

is maximized at $\boldsymbol{x} = \boldsymbol{x}^*$, and

2) $\boldsymbol{x}^*$ lies in a feasible set defined by linear constraints of the form $\boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{c}$ where $\boldsymbol{c} \in \mathbf{R}_+^p$ is a capacity vector for a number p of resources and $\boldsymbol{A}$ is the binary user-resource incidence (p, n)-constraint matrix, for a number n of users, weighted by a positive vector $\boldsymbol{w} \in \mathbf{R}_+^n$. Particular fairness concepts can be derived from $(\boldsymbol{w}, \alpha)$-fair metrics with a particular value of α, namely: *max-throughput* ($\alpha = 0$), *proportional fairness*, or NBS ($\alpha = 1$), *minimum potential delay* ($\alpha = 2$) and arbitrarily close approximations of *max-min fairness* ($\alpha \to \infty$).

In particular, the concept of *max-min fairness* illustrates the idea that fairness should mean that improving the wealth of the richer automatically deteriorates the one of the poorer: a resource allocation strategy is said to be max-min fair if no allocation can be increased without penalizing another allocation that was smaller or equal in the beginning. This has been the classic resource sharing principle [35] and has been studied extensively. We illustrate a simple example where the reader can get a proper intuition of

the fundamental conflict between equity and efficiency[3].

EXAMPLE
Let us consider a network of n resources (called links) indexed by $1, \dots, n$. The network serves $n+1$ users, indexed by $0, \dots, n$, in the following way. User 0 has a path that goes through all the links, and each user $i = 1, \dots, n$ has a path made of the sole link i. We call this network the n-linear network. An illustration of the sharing scenario is carried in Figure 2.1.

For simplicity, let us assume all the links have the same capacity normalized at 1. On the one hand, if one wants to allocate resources to all the users in order to maximize the overall throughput, one can see that the best solution is to allocate 1 to all users but user 0 who would get 0, which is ultimately unfair, because as some are getting full capacity and therefore full service, one of the users is getting no service at all; if one is not convinced by the qualification *unfair*, one just needs to imagine that user 0 is also paying, let's say, as much as the other users, to get service from this network. On the other hand, allocating resources following max-min fairness would give $1/2$ to all users, which is intuitively ultimately fair.

Let us however adopt the throughput point of view. The maximum achievable throughput is n, as n users are receiving an allocation of 1 unit of resource. The overall throughput of the max-min fair allocation is $(n+1)/2$, because $n+1$ users are receiving an allocation of $1/2$ unit of resource. So, we see that the ultimate fairness offered by max-min fairness comes with the price of throughput degradation with a factor of nearly 2.

Let us imagine now that the sharing policy should ensure that all users *have the same amount of reserved resource*. For example, in the max-min fair allocation, user 0 is getting a resource allocation of $1/2$ but has actually $1/2$ unit of resource reserved on *each link*, whereas the other users have a resource allocation *and* a reservation of $1/2$ unit of resource all in all. If we say it is on the amount of reserved resources that fairness should apply, we get a new allocation: user 0 gets $\frac{1}{n+1}$, while all other users get $\frac{n}{n+1}$. Although user 0 still gets less than the other users, we can consider that this new allocation is *more fair* than the allocation given while maximizing throughput, and in the meantime, the overall throughput of this new allocation sums up to $\frac{n^2+1}{n+1}$ which is greater than $\frac{n+1}{2}$ (the max-min fair throughput). To sum up, this new allocation seems to trade-off better fairness and network efficiency. It turns out this last allocation strategy is exactly the NBS (or proportional fairness allocation). □

The concept of proportional fairness and its weighted variants were introduced by Kelly, Maulloo, and Tan [36] before the works of Mo and Walrand [34] in which it is presented as the $(\alpha = 1)$-fair allocation. Thus, the definition α-fairness permitted to formalize the idea of a continuous fairness-efficiency equilibrium parameterized by the real number $\alpha \geqslant 0$. To sum up, there was now a spectrum of functions including all the particular aforementioned fairness notions unified through a single formula,

[3]Of course, when the conflict occurs.

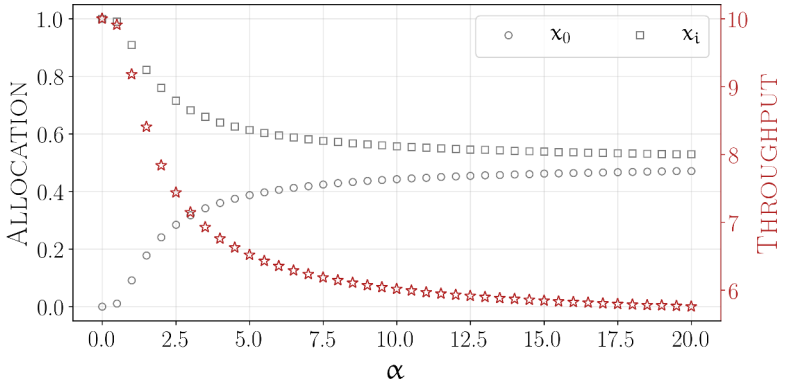


Figure 2.2 | The $(\mathbf{1}, \alpha)$-fair allocation and corresponding throughput in the 10-linear network.

and approaching arbitrarily the two extremes (maximum throughput and max-min fairness), with only the parameter α.

Now that we have more intuition on α-fairness, we can delve into its formalization as a mathematical problem. In the next section, we formalize, once and for all, the α-fair resource allocation problem in a network of multiple links. As the reader may have guessed, we define and adopt the terminology of rate control in fixed communication networks for the rest of this thesis.

2.2 The α-fair resource allocation: formalization

Let us formalize the weighted α-fair resource allocation problem.

Let $\mathcal{J}$ be the set of network links, each link j having a capacity $c_j \in \mathbf{R}_+$. Let $\mathcal{R}$ be the set of paths. Each path r is a predefined route that identifies with a subset $\mathcal{J}_r \subset \mathcal{J}$ of links of the network. In turn, for each link $j \in \mathcal{J}$, $\mathcal{R}_j \coloneqq \{r \in \mathcal{R}; j \in \mathcal{J}_r\}$ is the set of all paths having a route that contains the link j. We define the link-route incidence $|\mathcal{J}| \times |\mathcal{R}|$-matrix $\mathbf{A}$ as:

$$A_{jr} = \begin{cases} 1 & \text{if } j \in \mathcal{J}_r \\ 0 & \text{otherwise} \end{cases}$$

For each path r, x_r denotes the bandwidth allocated to r along its route $\mathcal{J}_r$. We say that an allocation $\boldsymbol{x} = (x_r)_{r\in\mathcal{R}}$ belongs to the feasibility set $\mathcal{C}$ (or is *feasible*) if it satisfies the capacity constraint (2.1) below:

$$\boldsymbol{x} \in \mathcal{C} \Leftrightarrow \mathbf{A}\boldsymbol{x} \leqslant \mathbf{c},\ \boldsymbol{x} \geqslant \mathbf{0} \tag{2.1}$$

where $\mathbf{c} = (c_j)_{j\in\mathcal{J}}$, and the inequality has to be understood component-wise. In this thesis, we define fairness in an asymmetric manner, in all generality. The asymmetry is enforced by an individual weight $w_r \in \mathbf{R}_+$ for each path r. In practice, the weight vector $\boldsymbol{w} = (w_r)_{r\in\mathcal{R}}$ accounts for a degree of relative importance between paths that can be defined at the discretion of the net-

work. For instance, it can account for a price paid by a customer purchasing bandwidth (hence the higher the price, the higher the incentive to reserve bandwidth for the corresponding route), or simply the number of connections requesting the path in question (hence, the larger the number of connections, the higher the incentive to reserve bandwidth to the corresponding path). The latter situation is illustrated for instance in [37]. Weighted $(\boldsymbol{w}, \alpha)$-fairness is formalized as in Definition 1 below.

DEFINITION 1 ($(\boldsymbol{w}, \alpha)$-fairness)
Let $\mathcal{C} \subset \mathbf{R}_+^{|\mathcal{R}|}$ be a feasibility set defined as in (2.1), being a strict superset of $\{0\}$. Let $\boldsymbol{w} \in \mathbf{R}_+^{|\mathcal{R}|}$ and $\boldsymbol{x}^* \in \mathcal{C}$. We say that $\boldsymbol{x}^*$ is $(\boldsymbol{w}, \alpha)$-*fair* (or simply α-fair when there is no confusion on $\boldsymbol{w}$) if the following holds:

$$\forall r \in \mathcal{R}, \quad x_r^* > 0 \quad \text{and} \quad \forall \boldsymbol{x} \in \mathcal{C}, \quad \sum_{r \in \mathcal{R}} w_r \frac{x_r - x_r^*}{x_r^{*\alpha}} \leqslant 0. \tag{2.2}$$

Equivalently, $\boldsymbol{x}^*$ is $(\boldsymbol{w}, \alpha)$-fair if, and only if $\boldsymbol{x}^*$ maximizes the α-fair utility function f^α defined[a] over $\mathcal{C} \cap \mathbf{R}_{++}^{|\mathcal{R}|}$:

$$\max_{\boldsymbol{x} \in \mathcal{C}} f^\alpha(\boldsymbol{w}, \boldsymbol{x}) = \sum_{r \in \mathcal{R}} f_r^\alpha(w_r, x_r), \tag{2.3}$$

where

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

[a] We introduce the redundant subscript r to the individual functions f_r^α. In the next chapter, the argument w_r will be omitted to avoid heaviness but remembered through this notation.

The equivalence between the two definitions is straightforward: saying that f^α is maximal at some $\boldsymbol{x}^*$ over $\mathcal{C}$ is exactly saying that $\nabla_{\boldsymbol{x}} f^\alpha(\boldsymbol{x}^*)^T(\boldsymbol{x} - \boldsymbol{x}^*) \leqslant 0$ for all $\boldsymbol{x} \in \mathcal{C}$, which is the compact way of writing Equation (2.2).

Therefore, the $(\boldsymbol{w}, \alpha)$-fair resource allocation problem can be cast as the following optimization problem:

$$\begin{aligned} \max \ & f^\alpha(\boldsymbol{w}, \boldsymbol{x}) = \sum_{r \in \mathcal{R}} f_r^\alpha(w_r, x_r) \\ \text{s.t. } & \boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{c}, \\ & \boldsymbol{x} \geqslant \boldsymbol{0}. \end{aligned} \tag{$\mathcal{P}_\alpha$}$$

As an illustration, Figure 2.2 shows the optimal allocations of the user 0 and users $i = 1, \ldots, 10$ in the 10-linear network introduced in our example above, for different values of α, as well as the corresponding throughput

of the system. The reader can appreciate the throughput deterioration that occurs for this particular topology when α-fairness approaches the notion of equity.

We are facing a concave maximization problem under linear constraints. Equivalently, it is a convex minimization problem under linear constraints. The methods that can be employed to solve this problem will be presented in the next chapter, focused on *how to solve this problem distributively by modern distributed SDN controllers*, rather than only *how to solve this problem*. But first, we present some results on the structure of the optimal solution. In fact, one can remark that the fairness objective f is unbounded from below near (any of) the x_r-axis[4]. This means that the allocation vectors that lie *near the axis* are sub-optimal. The question we answer to, in the next section, is how to prune the feasible set away from the axis. This manipulation is quite intuitive, yet very useful. Indeed we end up with a bounded problem with a differentiable objective function whose gradient is *then* also bounded. In order to do this, one could simply choose ε sufficiently small and prune the feasible set according to the value of ε. However, though easier to carry, this does not give any information on the fair allocation or the problem structure. The properties of a lower bound on an optimal solution is important and gives it a more rigid structure (especially in the case of unbounded variations near the axis) that can be used to optimize the behavior of algorithms that compute its value. This is the topic of Chapter 4, where the reader will see how we use the structural properties of the present chapter to tune our algorithms. For now, we present the lower bound on the fair resource allocation problem.

2.3 A lower bound on α-fairness

In this section, we derive an explicit lower bound on the general $(\boldsymbol{w}, \alpha)$-fair resource allocation problem. Our lower bound depends on the weight vector $\boldsymbol{w}$, the capacity vector $\mathbf{c}$ and the link-route incidence matrix $\mathbf{A}$. Moreover, the bound exploits the local structure of the problem, which prevents it from deteriorating systematically with the problem size. We compare it to the bound that one can formulate as follows:

[4]Formally, we mean that for $r \in \mathcal{R}$, for all $(x_s)_{s \neq r} > 0$, if $x_r > 0$ and $x_r \to 0$, then $f(\boldsymbol{x}) \to -\infty$.

THEOREM 1 (Marasevic, Stein, and Zussman [38])
Let the vector $\mathbf{x}^*$ be the optimal solution to the α-fair resource allocation problem. Then, for all $r \in \mathcal{R}$:

- if $0 < \alpha \leqslant 1$, $\quad x_r^* \geqslant m_r(\alpha) \coloneqq \left(\frac{w_r}{w_{max} M} \min_{j \in \mathcal{J}_r} \frac{c_j}{|\mathcal{R}_j|}\right)^{1/\alpha} c_{max}^{1-1/\alpha}$
- if $\alpha > 1$, $\quad x_r^* \geqslant m_r(\alpha) \coloneqq \left(\frac{w_r}{w_{max} M}\right)^{1/\alpha} \min_{j \in \mathcal{J}_r} \frac{c_j}{|\mathcal{R}_j|} \left(\frac{c_{min}}{c_{max}}\right)^{1-1/\alpha}$

where $w_{max} = \max w_r$, $M = \min\{|\mathcal{R}|, |\mathcal{J}|\}$, $c_{min} = \min c_j$ and $c_{max} = \max c_j$.

We seek to derive *user-centric* formulas in the sense that their value for a specific user would depend only on the resources within a localized problem (and not on the global topology) and only on the users *that compete over the same resources*. We then evaluate the formulas under different instance regimes and compare them to the literature in order to appreciate the improvements they provide.

A well known lower bound of the proportionally fair ($\alpha = 1$) resource allocation was brought in as a building block of the axiomatization of the Nash Bargaining Solution and is commonly referred to as the *midpoint domination* axiom [32]. It states that each path r is given at least a fraction $\frac{w_r}{\sum_{s \in \mathcal{R}} w_s}$ of their *dictatorial allocation*, that is, the resource they would receive if the other users accept to receive 0. We refer to the bound given by the midpoint domination axiom as the *midpoint allocation*. One can imagine that the midpoint allocation becomes arbitrarily poor as the total number of users becomes large, and its utility as a first estimation of the optimum allocation, negligible. Indeed, the formula includes the weights of the whole set of users and is independent of the problem's local structure. Similarly, the general lower bound found in [38] may suffer from these dependencies.

Concerning proportional fairness ($\alpha = 1$), we give a more precise midpoint domination axiom, and provide a lower bound that we call *local midpoint*. Our lower bound on the proportionally fair allocation can be interpreted as a particular case of the midpoint domination axiom to locality – now, each path r is proportionally fairly attributable at least a fraction $\frac{w_r}{\sum_{s \in \mathcal{R}^r} w_r}$ of their dictatorial allocation, where $\mathcal{R}^r$ is not the total set of paths, but *the set of paths in competition with the user* r *for some resource*. Few works attempted at providing lower bounds for the general $(\boldsymbol{w}, \alpha)$-fair resource allocation. In fact, the most recent available bound is shown by [38] (see Theorem 1), and used by the authors for an initialization of their α-fair heuristic. To the best of our knowledge, this is the best bound that could be found in the literature for the $(\boldsymbol{w}, \alpha)$-fair resource allocation problem and we refer to it as the State-of-the-Art (SoA). We seek to improve the above bound by removing the global dependencies on w_{max}, n_j, and M, c_{min} and c_{max}, those

parameters being the major degradation factor when the size or congestion of the problems increase.

For each path $r \in \mathcal{R}$, let $u_r \coloneqq \min_{j \in \mathcal{J}_r} c_j$. The so-called *utopia point* $\mathbf{u} \coloneqq (u_r)_{r \in \mathcal{R}}$ is the (infeasible when the problem is non trivial) allocation representing the value each path would receive if they were alone in the network, that is, its dictatorial allocation. Our bound for the $(\boldsymbol{w}, \alpha)$-fair allocation only depends on the utopia point (hence on the capacity vector $\mathbf{c}$), the matrix $\mathbf{A}$ and on the weight vector $\boldsymbol{w}$. For $r \in \mathcal{R}$, let $\mathcal{R}^r \coloneqq \{s \in \mathcal{R}; \mathcal{J}_r \cap \mathcal{J}_s \neq \varnothing\}$, i.e., the set of paths sharing at least one resource with r and $\bar{\mathcal{R}}^r \coloneqq \mathcal{R} - \mathcal{R}^r$.

First of all, we use the separability of the objective function of Problem $(\mathcal{P}_\alpha)$ to better estimate our lower bound on a restricted problem. Specifically, we prove a *restriction lemma* (see Lemma 1) that permits one to avoid unnecessary dependencies between paths that do not share resources together. Then, we prove our general lower bound on the corresponding restricted problems. Thanks to Lemma 1, the bound remains unchanged in the original problem.

2.3.1 A restriction lemma

In this paragraph, we show that instead of evaluating our bound on Problem $(\mathcal{P}_\alpha)$, one can use a smaller path-centric problem. Specifically, let $\boldsymbol{x}^*$ denote the optimal solution of Problem $(\mathcal{P}_\alpha)$ and let $r_0 \in \mathcal{R}$ be *an arbitrary path*. We define the *restricted problem* at r_0, as the following:

$$
\begin{aligned}
\max \quad & \sum_{r \in \mathcal{R}^{r_0}} f_r^\alpha(w_r, x_r) \\
\text{s.t.} \quad & \sum_{r \in \mathcal{R}_j \cap \mathcal{R}^{r_0}} x_r \leqslant \tilde{c}_j \coloneqq c_j \quad \forall j \in \mathcal{J}_{r_0}, \\
& \sum_{r \in \mathcal{R}_j \cap \mathcal{R}^{r_0}} x_r \leqslant \tilde{c}_j \coloneqq c_j - \sum_{r \in \mathcal{R}_j \cap \overline{\mathcal{R}^{r_0}}} x_r^* \quad \forall j \in \mathcal{J} - \mathcal{J}_{r_0}.
\end{aligned}
\tag{$\mathcal{P}_\alpha^{r_0}$}
$$

Intuitively, Problem $(\mathcal{P}_\alpha^{r_0})$ arises when the allocations of all the paths that do not share any link with r_0 are fixed to their optimal α-fair value (that is, following the vector $\mathbf{x}^*$), and one needs to compute the α-fair allocation of the remaining paths, that is, the paths within $\mathcal{R}^{r_0}$ that share at least one resource with r_0. The capacity constraints are thus updated taking into account the amounts of resources that are already allocated, as shows the second line of the constraints. We remark that all the links in $\mathcal{J} - \mathcal{J}_{r_0}$ that do not serve any of the paths within $\mathcal{R}^{r_0}$ form trivial constraints in $(\mathcal{P}_\alpha^{r_0})$ and can hence be removed without any loss. Also, the restriction to Problem $(\mathcal{P}_\alpha^{r_0})$ comes with an *a priori* new value of the utopia point that we denote $\tilde{\mathbf{u}}$. We thus

remind the definition[5]:

$$\forall r \in \mathcal{R}^{r_0}, \quad \tilde{u}_r \coloneqq \min_{j \in \mathcal{J}_r} \tilde{c}_j \tag{2.4}$$

We then have the following result:

LEMMA 1
The restriction to $(\mathcal{P}_\alpha^{r_0})$ does not change the optimal allocation of the remaining paths: if $\mathbf{x}$ is the optimal solution of the Problem $(\mathcal{P}_\alpha^{r_0})$, then, $x_s = x_s^*$, for $s \in \mathcal{R}^{r_0}$.

Proof. Consider the problem:

$$\begin{aligned} \max & \sum_{r \in \mathcal{R}} f_r^\alpha(w_r, x_r) \\ \text{s.t. } & \boldsymbol{A}\boldsymbol{x} \leqslant \mathbf{c}, \\ & x_r \geqslant x_r^* \quad \forall r \in \bar{\mathcal{R}^{r_0}}. \end{aligned} \tag{2.5}$$

It suffices to show that the problems (2.5) and $(\mathcal{P}_\alpha^{r_0})$ are equivalent. Then, the unicity of the solutions permits one to conclude.

We know that the problem (2.5) is feasible, as $\mathbf{x}^*$ is a feasible point. Denote its optimal solution by $\tilde{\mathbf{x}}$. We remark that both $\mathbf{x}^*$ and $\tilde{\mathbf{x}}$ are feasible for both problems $(\mathcal{P}_\alpha)$ and (2.5). Hence, by optimality, we necessarily have $f^\alpha(\boldsymbol{w}, \tilde{\mathbf{x}}) = f^\alpha(\boldsymbol{w}, \mathbf{x}^*)$. Moreover, for instance, problem $(\mathcal{P}_\alpha)$ has a unique optimal solution. Thus,

$$\mathbf{x}^* = \tilde{\mathbf{x}}.$$

Particularly for $r \in \bar{\mathcal{R}^{r_0}}, x_r^* = \tilde{x}_r$. Thus, we can fix the values $x_r = x_r^*$ for $r \in \bar{\mathcal{R}^{r_0}}$ without changing the optimum. Thus, solving Problem (2.5) is equivalent to solving the restricted problem $(\mathcal{P}_\alpha^{r_0})$, in the sense that the two allocations $\mathbf{x}$ and $\tilde{\mathbf{x}}$ they give are equal. hus, $x_s^* = \tilde{x_s} = x_s$, for all $s \in \mathcal{R}_{r_0}$. □

Thanks to Lemma 1, we are now ready to present our lower bound on the α-fair allocation based on the structure of the restricted problems.

2.3.2 Lower bound

We now show the main result of the discussion. We define the *local midpoint* $\mathbf{p}$ as the following:

$$\forall r \in R \quad p_r \coloneqq \frac{w_r}{\sum_{s \in \mathcal{R}^r} w_s} u_r.$$

[5]Here, we omit the subscript reference to r_0 to avoid heaviness. Each time the notation $\tilde{u}$ will be used, the restriction will be clearly specified.

THEOREM 2
Let x^* denote the optimal solution of problem $(\mathcal{P}_\alpha)$. Let $r_0 \coloneqq \operatorname{argmin}_{s\in\mathcal{R}} p_s$. Then, x^* can be lower bounded as follows:

- if $\alpha \geqslant 1$, $\quad \forall r \in \mathcal{R} \quad x_r^* \geqslant d_r(\alpha) \coloneqq p_{r_0}^{1-1/\alpha} p_r^{1/\alpha}$
- if $0 < \alpha \leqslant 1$, $\quad \forall r \in \mathcal{R} \quad x_r^* \geqslant d_r(\alpha) \coloneqq \left(\dfrac{w_r u_r}{\sum_{s\in\mathcal{R}^r} w_s u_s^{1-\alpha}} \right)^{1/\alpha}$.

Proof. We first prove the proposition for $\alpha \geqslant 1$. Let us define the path $r_{\min}$ as the path with the least optimal allocation: $r_{\min} = \operatorname{argmin}_{s\in\mathcal{R}} x_s^*$. By definition of r_0, we have:

$$p_{r_{\min}} \geqslant p_{r_0} \tag{2.6}$$

Let $r \in \mathcal{R}$. By Lemma 1, it suffices to show the inequality in the restricted problem $(\mathcal{P}_\alpha^r)$ associated to r. Let $\mathcal{C}^r$ denote its feasible set. Thus, for all $(x_s)_{s\in\mathcal{R}^r} \in \mathcal{C}^r$ we have:

$$\sum_{s\in\mathcal{R}^r} w_s \frac{x_s - x_s^*}{x_s^{*\alpha}} \leqslant 0,$$

This inequality holds for all feasible $(x_s)_{s\in\mathcal{R}^r} \in \mathcal{C}^r$. Thus, we evaluate it at the dictatorial allocation of r, that is, at the point x defined as $x_r = \tilde{u}_r$ and $x_s = 0$ for all $s \in \mathcal{R}^r - \{r\}$. In light of (2.4), we have: $\tilde{u}_r = \min_{j\in\mathcal{J}_r} \tilde{c}_j = \min_{j\in\mathcal{J}_r} c_j = u_r$. Thus,

$$w_r \tilde{u}_r = w_r u_r \leqslant x_r^{*\alpha} \sum_{s\in\mathcal{R}^r} w_s x_s^{*1-\alpha} \leqslant \left(\sum_{s\in\mathcal{R}^r} w_s \right) x_{r_{\min}}^{*1-\alpha} x_r^{*\alpha}, \tag{2.7}$$

where we remind that $r_{\min} = \operatorname{argmin}_{s\in\mathcal{R}} x_s^*$ and $1 - \alpha \leqslant 0$. Rearranging the terms, one gets:

$$\frac{w_r u_r}{\sum_{s\in\mathcal{R}^r} w_s} x_{r_{\min}}^{*\alpha-1} \leqslant x_r^{*\alpha},$$

which yields:

$$p_r^{1/\alpha} x_{r_{\min}}^{*1-1/\alpha} \leqslant x_r^* \tag{2.8}$$

In particular, applying equation (2.8) to $r = r_{\min}$ and using (2.6), we get:

$$x_{r_{\min}}^* \geqslant p_{r_{\min}} \geqslant p_{r_0} \tag{2.9}$$

Finally, we plug (2.9) in (2.8) to obtain the desired lower bound on x_r^* (because $1 - 1/\alpha \geqslant 0$).

Path i	0	1	2	3	4	5
w_i	0.51	0.54	0.72	0.73	1.48	1.08
c_i	x	1.05	0.66	1.25	1.11	1.08

Table 2.1 | The samples of $\mathbf{c}$ and $\boldsymbol{w}$ used for the comparisons of $\mathbf{d}$ and $\mathfrak{m}$ in the 5-linear network.

Next, we show the bound for $0 < \alpha < 1$. In the same fashion, we look at the restricted problem. Let $r \in \mathcal{R}$ and consider its restricted problem. Then, one has:

$$\frac{w_r u_r}{x_r^{*\alpha}} \leqslant \sum_{s\in\mathcal{R}^r} w_s x_s^{*1-\alpha} \leqslant \sum_{s\in\mathcal{R}^r} w_s \tilde{u}_s^{1-\alpha} \leqslant \sum_{s\in\mathcal{R}^r} w_s u_s^{1-\alpha}. \tag{2.10}$$

Rearranging the terms finally provides the desired bound. For any value of α, one can remark that the bound $(d_r(\alpha))_{r\in R}$ only depends on the capacity vector $\mathbf{c}$, the weight vector $\boldsymbol{w}$, and the link-route incidence structure given by $\boldsymbol{A}$. □

2.4 Comparison of $\mathbf{d}$ and $\mathfrak{m}$

In this section, we compare the bound derived in Theorem 2 to the one presented in Theorem 1 on several examples and settings. Broadly, our bound $\mathbf{d}$ numerically outperforms $\mathfrak{m}$ for $\alpha = 1, 2$ and greater values to a certain extent. In all generality, the analytical expression of $\frac{\mathbf{d}}{\mathfrak{m}}$ suggests that the relative values of the bounds should depend on the system settings, namely the variance in the weights $\boldsymbol{w}$ and in the capacities $\mathbf{c}$. This means that the two bounds may behave differently to a fluctuating asymmetry of the weight vector $\boldsymbol{w}$ or the capacity vector $\mathbf{c}$, namely, a variation of the two parameters $\delta_w := \frac{\min w_r}{\max w_r}$ and $\delta_c := \frac{c_{min}}{c_{max}}$. This is why, on an illustration on the 5-linear network (see the illustration in Figure 2.1), we compare the bounds in the four possible scenarios, *scenario 0)* $\boldsymbol{w}$ and $\mathbf{c}$ generated following Table 2.1, *scenario 1)* $\mathbf{c}$ generated following Table 2.1 and $\boldsymbol{w}$ equal to $\mathbf{1}$, *scenario 2)* $\boldsymbol{w}$ generated following Table 2.1 and $\mathbf{c}$ equal to $\mathbf{1}$ *scenario 3)* $\boldsymbol{w}$ and $\mathbf{c}$ equal to $\mathbf{1}$. Figures 2.3–2.6 show the values of $\mathbf{d}$, $\mathfrak{m}$ and the optimum $\mathbf{x}^*$ on the 5-linear network.

Next, we compare on synthetic instances how the two bounds relate to each other when α varies.

The two bounds were compared on instances with 1000 requests over a same graph of type *barabasi(100,4)* (see [39]). The routes were generated at random by taking the shortest path between pairs of sources and destinations drawn uniformly at random. The weights (resp. link capacities) were also drawn uniformly at random within intervals I satisfying $\inf I / \sup I = \delta_w$ (resp. δ_c). For each instance, and each α, we define the *score* of $\mathbf{d}$ as the number $|\{r : d_r(\alpha) > m_r(\alpha)\}|/|\mathcal{R}|$. The score represents the proportion of requests for which our bound $\mathbf{d}(\alpha)$ beats the SoA bound $\mathfrak{m}(\alpha)$ for a particular

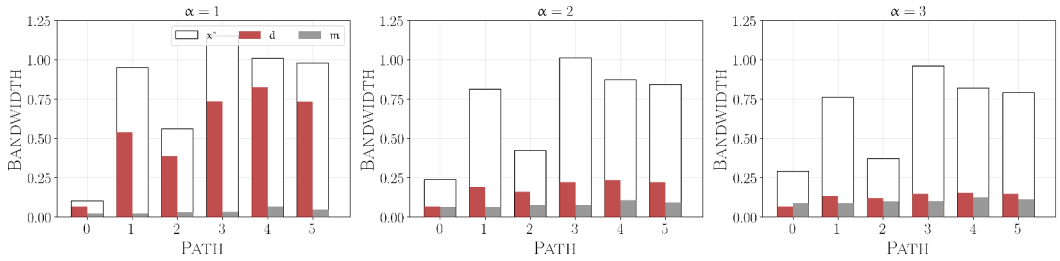


Figure 2.3 | Optimal allocation $\mathbf{x}^*$ and the bounds $\mathbf{d}$ and $\mathbf{m}$ on the 5-linear network with scenario 0.

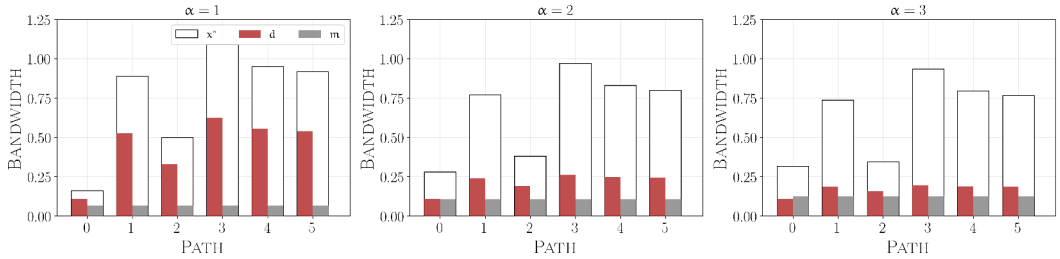


Figure 2.4 | Optimal allocation $\mathbf{x}^*$ and the bounds $\mathbf{d}$ and $\mathbf{m}$ on the 5-linear network with scenario 1.

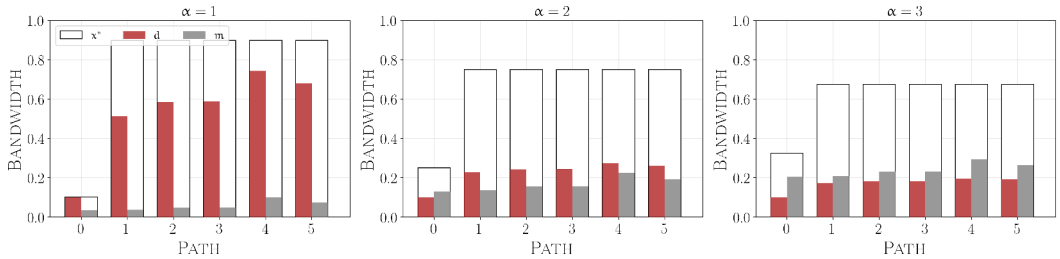


Figure 2.5 | Optimal allocation $\mathbf{x}^*$ and the bounds $\mathbf{d}$ and $\mathbf{m}$ on the 5-linear network with scenario 2.

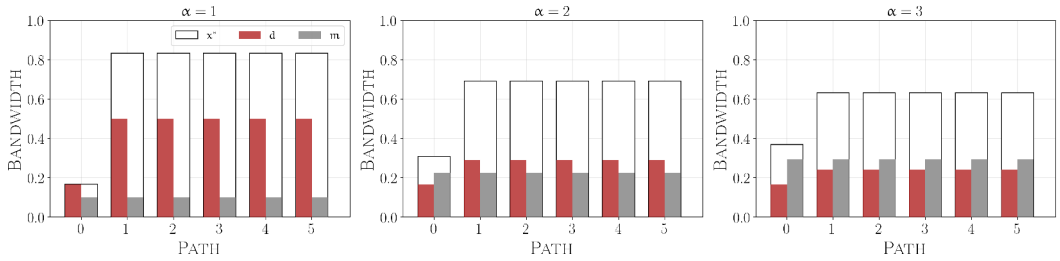


Figure 2.6 | Optimal allocation $\mathbf{x}^*$ and the bounds $\mathbf{d}$ and $\mathbf{m}$ on the 5-linear network with scenario 3.

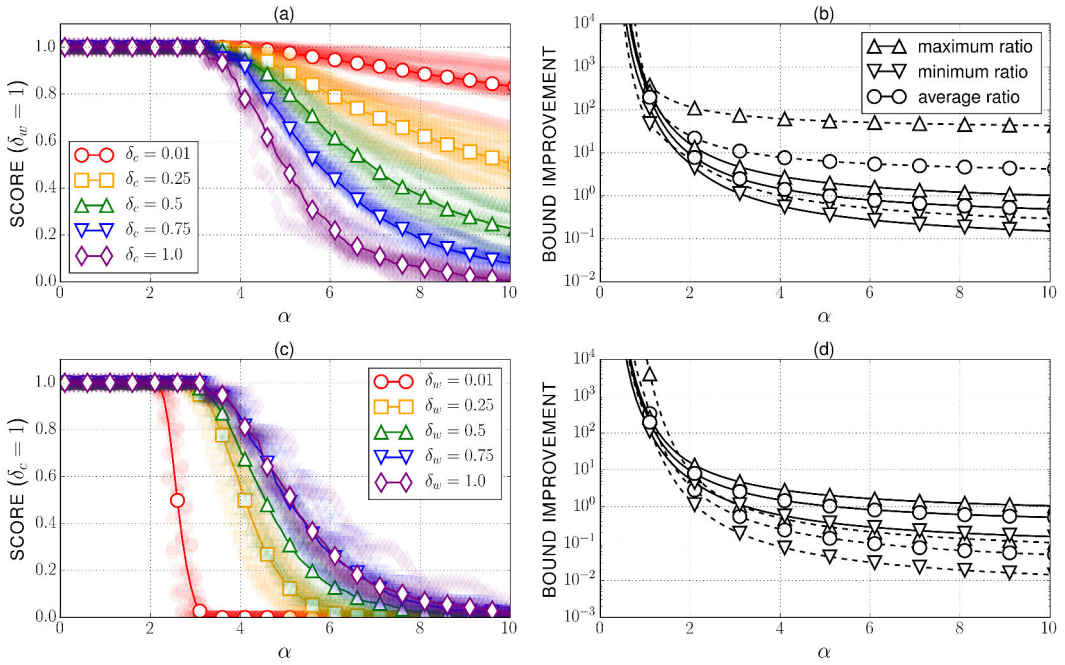


Figure 2.7 | A comparison of the two bounds. The scores, and the minimum, average and maximum bound improvements are illustrated in the cases of (a)-(b) a constant δ_w for different values of δ_c, and of (c)-(d) a constant δ_c for different values of δ_w. Figures (b) and (d) show the bound improvements in the two extreme situations δ_c (resp. δ_w) $= 0.01$ (resp. 1) in dashed lines (resp. solid lines).

α. In Figure 2.7(a), the parameter δ_w was fixed to 1 (which namely means $\boldsymbol{w} = \mathbf{1}$) and we plotted the score of $\mathbf{d}$ versus α for different values of δ_c. Figure 2.7(c) shows the score in the other extreme situation $\delta_c = 1$ (which means all the link capacities are equal) for different values of δ_w.

In order to appreciate the quality of the bound improvement, if any, we plotted, in Figures 2.7(b) and 2.7(d), the corresponding bound improvements, measured with the values of the ratios $d_r(\alpha)/m_r(\alpha)$. To preserve readability of the plots, we represented only the extreme situations corresponding to the values $\delta_c = 0.01$ (dashed lines) and $\delta_c = 1$ (solid lines) for Figure 2.7(b) and to the values $\delta_w = 0.01$ (dashed lines) and $\delta_w = 1$ (solid lines) for Figure 2.7(d). Figures 2.7(b) and 2.7(d) show the best, worst, and average improvements encountered in the same problem instance. All the points represented in Figure 2.7 correspond to an average over 10 instances generated under identical conditions. In Figures 2.7(a) and 2.7(c), we also included the specific points as translucent scattered markers.

According to Figure 2.7, our bound is an absolute improvement for values of α in the interval [0,2] (thus including the max-throughput, proportional fairness, and min-delay popular concepts) in all situations. Particularly for proportional fairness, the simulations show that we improved the bound $\mathbf{m}$ by two orders of magnitude in all situations. For min-delay fairness, the bound is generally improved on average by a multiplicative factor

between 1 and several tens. For greater values of α, it is interesting to see that either $\mathbf{d}$ or $\mathbf{m}$ is more adapted to certain problem structures. For instance, $\mathbf{d}$ will be of greater interest when the network link capacities are more heterogeneous, $\delta_c \ll 1$ (which may correspond to situations where the network is asymmetrically congested), whereas $\mathbf{m}$ is more adapted to asymmetrically weighted problems, $\delta_w \ll 1$. One can thus conclude that the two available bounds complement each other for general $\alpha \geqslant 1$.

2.5 Reflexions on a possible improvement

In this section, we seek to discuss on a possible improvement on the case $\alpha \geqslant 1$. As one can see, the bound given in Theorem 2 are completely localized (in the sense that they only depend on parameters within the restricted problem) for $0 \leqslant \alpha \leqslant 1$, and the improvement they offer compared to Theorem 1 are definitive. However, for greater values of α, the influence of the global term p_{r_0} is clearly destructive to the individual values of the bounds, as now, a *global* (in the sense that it could intuitively have limited incidence given the form of the restricted problem) imbalance in the weights $\boldsymbol{w}$ and/or the capacity vector $\mathbf{c}$ has a substantial influence on the degradation of the bounds (see Figure 2.7). Therefore, we wish to derive, in the same fashion as in the cases $0 \leqslant \alpha \leqslant 1$, the same type of *user-centric* formulas. In order to do this, we examine more closely the methodology of the previous section.

The proof in the previous section happens to suggest an iterative technique to derive a new bound based on an already known bound. Indeed, in the inequalities within Equation (2.10), one can see that when $\alpha \geqslant 1$, the term $\sum_{s\in\mathcal{R}^r} w_s x_s^{*1-\alpha}$ can be dominated by a term $\sum_{s\in\mathcal{R}^r} w_s \beta_s^{1-\alpha}$, where $\boldsymbol{\beta}$ is a *known* lower bound on the fair allocation $\mathbf{x}^*$. Particularly, this observation can be summarized in the following lemma:

LEMMA 2
Let $\alpha \geqslant 1$. Let $\mathbf{b}$ be a known lower bound on the optimal solution. Then, we have:

$$\forall r \in \mathcal{R}, \quad x_r^* \geqslant \left(\frac{w_r u_r}{\sum_{s\in\mathcal{R}^r} w_s b_s^{1-\alpha}} \right)^{1/\alpha} \tag{2.11}$$

Proof. Fix $r \in \mathcal{R}$. In the same fashion as in 2.3.2, we look at the restricted problem around r. We remark that the allocation $x_r = u_r$ and $x_{s\in\mathcal{R}^r-\{r\}} = 0$ is feasible in the restricted problem. Therefore, the optimality condition:

$$\frac{w_r u_r}{x_r^{*\alpha}} \leqslant \sum_{s\in\mathcal{R}^r} w_s x_s^{*1-\alpha} \tag{2.12}$$

holds. But $1-\alpha \leqslant 0$. Hence, the right-hand side of the inequality above is less than the bound:

$$\sum_{s\in\mathcal{R}} w_s b_s^{1-\alpha}.$$

Rearranging the terms, we get:

$$x_r^{*\alpha} \geqslant \frac{w_r u_r}{\sum_{s\in\mathcal{R}^r} w_s b_s^{1-\alpha}}, \tag{2.13}$$

which concludes the proof. □

Therefore, given a lower bound $\mathbf{b}$ on the fair allocation $\mathbf{x}^*$, Lemma 2 tells us that another lower bound can be derived. More precisely, we have the following immediate corollary:

COROLLARY 1
Let $\alpha \geqslant 1$. Let $\mathbf{b}(0) \coloneqq \mathbf{d}(\alpha)$ and the sequence $\mathbf{b}(n)$ be defined by the formula:

$$\forall r \in \mathcal{R} \quad b_r(n+1) \coloneqq \left(\frac{w_r u_r}{\sum_{s\in\mathcal{R}^r} w_s b_s(n)^{1-\alpha}} \right)^{1/\alpha}. \tag{2.14}$$

Then, $(\mathbf{b}(n))_n$ is a sequence of lower bounds for the α-fair resource allocation problem for $\alpha \geqslant 1$.

The behavior of the sequence $(\mathbf{b}(n))_{n\in\mathbf{N}}$ is of course of interest. Let us first illustrate it on a simple example of the 2-linear network introduced in 2.1. In this case, we have the following:

$$b_0(n+1) = \frac{w_0 c_0}{w_0 b_0(n)^{1-\alpha} + w_1 b_1(n)^{1-\alpha} + w_2 b_2(n)^{1-\alpha}} \tag{2.15}$$

and

$$b_i(n+1) = \frac{w_i c_i}{w_0 b_0(n)^{1-\alpha} + w_i b_i(n)^{1-\alpha}}, \quad i = 1, 2. \tag{2.16}$$

In Figure 2.8, we plot the iterates $\mathbf{b}(n)$ for a value of α set to $1, 2$ and 4. The trajectories are plotted for 100 different initializations of $\mathbf{b}(0)$ (drawn uniformly at random in the interval $[0, 1]$) and we demonstrate its empirical convergence in this situation. Figure 2.8 is shown to illustrate the convergence of $\mathbf{b}(n)$. In fact, we remark in the simulations that $\mathbf{b}(n)$ converges to the same point, no matter its initialization. This behavior suggests us that a limit point could be derived in a closed form that does not contain any influence from initialization. We seek to get insights on this form, and its potential expression. In order to do that, we relax a little bit the formulas

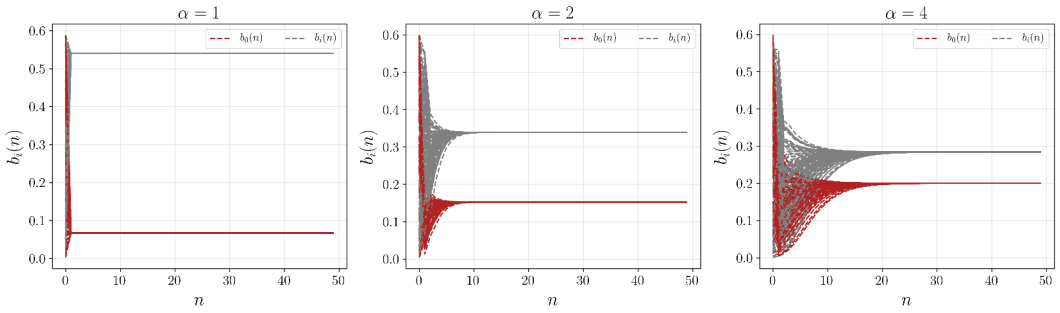


Figure 2.8 | The behaviour of $\mathbf{b}(n)$ on the 5-linear network on scenario 0

by not restricting the problem. Lemma 2 and Corollary 1 can also be stated in the global problem, that is, by replacing $\mathcal{R}^r$ by simply $\mathcal{R}$ in the denominator. Of course, this will necessarily deteriorate the value of the bounds. Nonetheless, it is more simple to work with, and get the following result:

PROPOSITION 1
Let $\alpha \geqslant 1$. Let $\mathbf{x}^*$ denote the optimal solution of problem $(\mathcal{P}_\alpha)$. Then, $\mathbf{x}^*$ can be lower bounded as follows:

$$\forall r \in \mathcal{R}, \quad x_r^* \geqslant \frac{w_r^{1/\alpha} u_r^{1/\alpha}}{\sum_{s \in \mathcal{R}} w_s^{1/\alpha} u_s^{1/\alpha - 1}}. \tag{2.17}$$

Proof. Let n be a positive index. We show by recursion on $1 \leqslant k \leqslant n$ that $\mathbf{b}(n)$ takes the following form:

$$b_r(n) = w_r^{1/\alpha} u_r^{1/\alpha} K_{n-k}^{q_{n-k}^n} \left(\sum_{s \in \mathcal{R}} w_s^{1/\alpha} u_s^{1/\alpha - 1} \right)^{h_{n-k}^n}, \tag{2.18}$$

where $K_{n-k} := \left(\frac{f^\alpha(\boldsymbol{w}, \mathbf{b}(n-k))}{1-\alpha} \right) = \sum_{s \in \mathcal{R}} w_s b_s(n-k)^{1-\alpha}$, q_{n-k}^n and h_{n-k}^n are real numbers satisfying a recursion rule that will be deduced.

We know that for $k = 1$, this is Equation (2.18) with $q_{n-1}^n = -1/\alpha$ and $h_{n-1}^n = 0$.

Further, one has:

$$K_{n-k}^{q_{n-k}^n} = \left(\sum_{s \in \mathcal{R}} w_s \left(\frac{w_s u_s}{K_{n-(k+1)}} \right)^{1/\alpha - 1} \right)^{q_{n-k}^n} \tag{2.19}$$

$$= \left(\sum_{s \in \mathcal{R}} w_s^{1+1/\alpha-1} u_s^{1/\alpha-1} \right)^{q_{n-k}^n} K_{n-(k+1)}^{-(1/\alpha-1)q_{n-k}^n} \tag{2.20}$$

$$= \left(\sum_{s \in \mathcal{R}} w_s^{1/\alpha} u_s^{1/\alpha-1} \right)^{q_{n-k}^n} K_{n-(k+1)}^{(1-1/\alpha)q_{n-k}^n}. \tag{2.21}$$

Therefore, by plugging in the expression of $K_{n-k}^{q_{n-k}^n}$ in Equation (2.18), one gets:

$$b_r(n) = w_r^{1/\alpha} u_r^{1/\alpha} K_{n-(k+1)}^{q_{n-(k+1)}^n} \left(\sum_{s \in \mathcal{R}} w_s^{1/\alpha} u_s^{1/\alpha-1} \right)^{h_{n-(k+1)}^n}, \tag{2.22}$$

with the recursion formulas:

$$\begin{cases} q_{n-(k+1)}^n = (1 - \frac{1}{\alpha}) q_{n-k}^n, \\ h_{n-k+1}^n = h_{n-k}^n + q_{n-k}^n \end{cases}. \tag{2.23}$$

Solving Equation (2.23) is straightforward, as $(q_{n-k}^n)_{k=1,\dots,n}$ is a geometric sequence with factor $(1 - 1/\alpha)$, and the expression of $(h_{n-k}^n)_{k=1,\dots,n}$ follows, one gets:

$$\begin{cases} q_{n-k}^n = -\frac{1}{\alpha}(1 - \frac{1}{\alpha})^{k-1} \\ h_{n-k}^n = (1 - \frac{1}{\alpha})^k - 1 \end{cases}. \tag{2.24}$$

Therefore, Equation (2.18) is proved, and, evaluating it for $k = n$, we have:

$$b_r(n) = w_r^{1/\alpha} u_r^{1/\alpha} K_0^{\frac{1}{\alpha}(1-\frac{1}{\alpha})^{n-1}} \left(\sum_{s \in \mathcal{R}} w_s^{1/\alpha} u_s^{1/\alpha-1} \right)^{(1-\frac{1}{\alpha})^n - 1}. \tag{2.25}$$

The formula (2.25) being valid for all n, as $q_0^n \to 0$ and $h_0^n \to -1$, we see that the sequence $b_r(n)$ is convergent[6] to the value:

$$b_r^\star = \frac{w_r^{1/\alpha} u_r^{1/\alpha}}{\sum_{s \in \mathcal{R}} w_s^{1/\alpha} u_s^{1/\alpha-1}}. \tag{2.26}$$

Finally, for each n, $b_r(n) \leqslant x_r^*$ by Corollary 1. □

The formula (2.26) gives a quite elegant expression of a lower bound for

[6] And we remark that the limit does not depend on $\mathbf{b}(0)$, because $q_0^n \to 0$.

the fair allocation for $\alpha > 1$. This form is seen in numerous works around resource sharing problems when only one resource is shared and can be derived by classic water-filling arguments or as an immediate consequence of KKT conditions [40]. Here we have generalized this formula to the case of the multi-resource sharing problem, and proved that it corresponds to a lower bound on the solution. However, as such, it does not improve the formulas derived in the last section. Indeed, we are now stuck with a summation over $\mathcal{R}$ in the denominator, which naturally deteriorates unnecessarily the value of the bound when the problem gets arbitrarily large. In particular, for $\alpha = 1$, the two formulas would coincide if the summation in the denominator of (2.25) was $\mathcal{R}^r$, and not $\mathcal{R}$. We would therefore prefer a localized formula in the fashion of Theorem 2. Unfortunately, the calculus in the proof of Proposition 1 does not hold in the restricted problem. Indeed, the quantity $K_k = K_k(r) = \sum_{s\in\mathcal{R}^r} w_s b_s(k)^{1-\alpha}$ now depends on r and cannot be taken out of the sum from line (2.19) to (2.20).

One way to circumvent this technical question is to see if, as in Lemma 1, it is possible to restrict the problem around $\mathcal{R}^r$ and derive the corresponding formulas. The difficulty is in that even though the result in Lemma 1 holds, it gives a bound that depends on the restricted utopia points $\tilde{\mathbf{u}}$. Unfortunately, we do not have in all generality $\mathbf{u} = \tilde{\mathbf{u}}$. We end this chapter by proposing a possible localization of (2.25) that generalizes the result obtained for $\alpha = 1$ in Theorem 2 to arbitrary values of α.

2.5.1 Suspected restricted formula

Our suspected restricted formula of a localized lower bound for the α-fair resource allocation problem can be stated as the (unproven) proposition below:

CONJECTURE 1 (Proven by Theorem 2 if $\alpha = 1$.)
Let $\alpha \geqslant 1$. Let $\mathbf{x}^*$ denote the optimal solution of problem $(\mathcal{P}_\alpha)$. Then, $\mathbf{x}^*$ can be lower bounded as follows:

$$\forall r \in \mathcal{R}, \quad x_r^* \geqslant \frac{w_r^{1/\alpha} u_r^{1/\alpha}}{\sum_{s\in\mathcal{R}^r} w_s^{1/\alpha} u_s^{1/\alpha-1}}. \tag{2.27}$$

Although this proposition is not proven for $1 < \alpha \leqslant \infty$, we illustrate in Figures 2.9–2.12 the effect of this possible improvement and plot on the same four scenarios as in Figures 2.3–2.6, its value (that we refer to as $\mathbf{D}$), against $\mathbf{m}$ and the optimum $\mathbf{x}^*$.

Because of the improved quality of the bound it yields, in the sequel, we always mean by the term "lower bound", the bound $\mathbf{D}$ given by the conjecture. Of course, we always verify numerically that the formula is indeed a lower bound before we use it, keeping in mind that we always have the

(less tight) lower bound $\mathbf{d}$ as backup. Of course, we could have used instead either $\mathbf{d}$, or the limit point of the series $(\mathbf{b}(n))_n$ initialized with $\mathbf{b}(0) = \mathbf{d}$, following the result of Corollary 1. However, interestingly, we have never found any counter example where $D_r > x^*_r$ for some r, for any value of $\alpha > 1$, in any of our instances. Therefore, because of the practicality of the evaluation of $\mathbf{D}$, and its empirical correctness and improvement with regards to $\mathbf{d}$ and $\mathbf{m}$, we used $\mathbf{D}$ instead.

2.6 Concluding remarks

In this chapter, the α-fair resource allocation problem was formalized, and presented as a convex optimization problem. We studied its structure and derived a lower bound on the solution. The first lower bound that we derive in Theorem 2 relies on a localization of the problem onto which we simply apply the definition or α-fairness. We discover that this lower bound permits to improve the one of Theorem 1 for values of $\alpha \leqslant 2$ in all the considered settings (Figure 2.7) but not for greater values of α. Motivated by the form of the individual formulas we obtain in Proposition 1, and by the restricted particular case for $\alpha = 1$ (Conjecture 1 and Theorem 2), we postulate that it is possible to show the same restricted versions of the formulas in Conjecture 1 for general $\alpha > 1$. However, this question is still problematic for us. We nevertheless illustrated (Figures 2.9–2.12) the potential improvement the bound of Conjecture 1 gives, compared to our established bounds. Other than bounding the optimal α-fair allocation from below, which itself is an interesting exercise and helps improve our understanding of α-fair allocations, the lower bound will be very helpful to tune the algorithms that we derive in order to compute them, the design of which (only) is the topic of the next chapter.

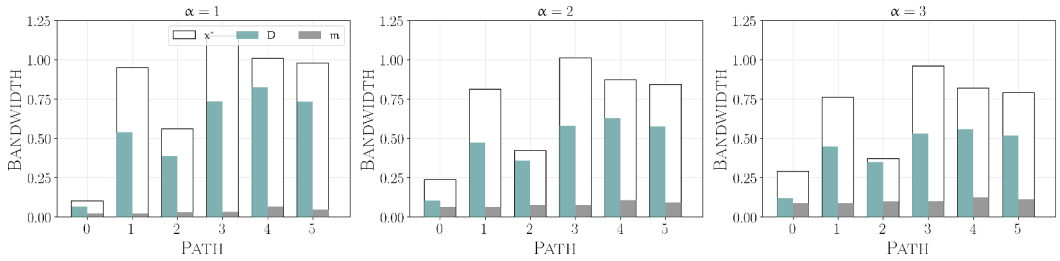


Figure 2.9 | Optimal allocation $\mathbf{x}^*$ and the bounds $\mathbf{m}$ and $\mathbf{D}$ on the 5-linear network with scenario 0.

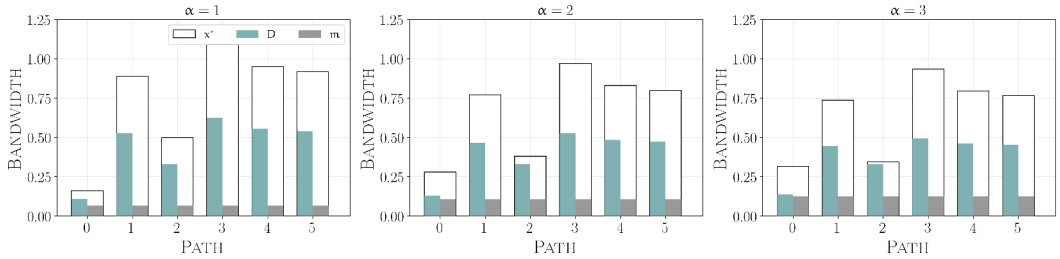


Figure 2.10 | Optimal allocation $\mathbf{x}^*$ and the bounds $\mathbf{m}$ and $\mathbf{D}$ on the 5-linear network with scenario 1.

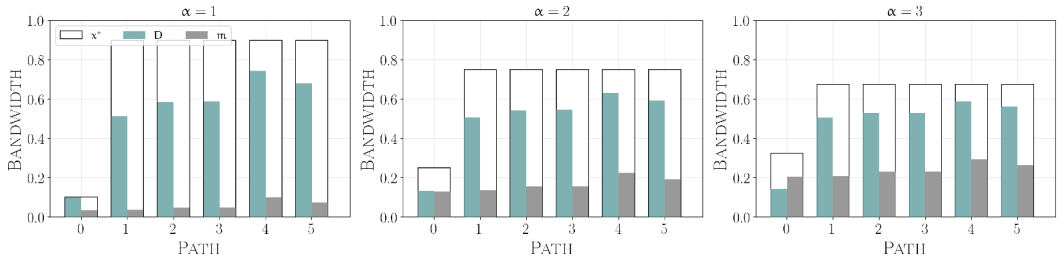


Figure 2.11 | Optimal allocation $\mathbf{x}^*$ and the bounds $\mathbf{m}$ and $\mathbf{D}$ on the 5-linear network with scenario 2.

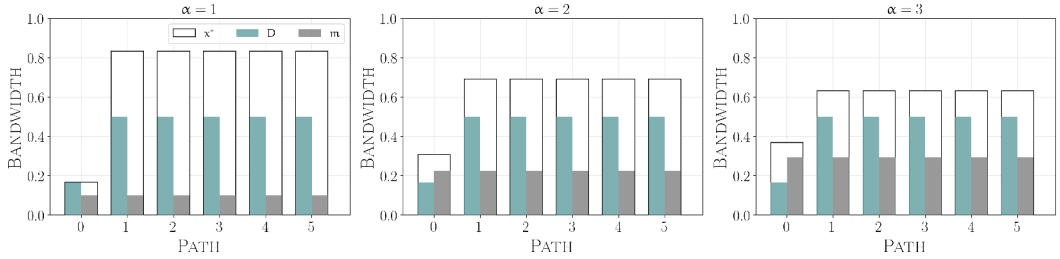


Figure 2.12 | Optimal allocation $\mathbf{x}^*$ and the bounds $\mathbf{m}$ and $\mathbf{D}$ on the 5-linear network with scenario 3.

Chapter 3

Fair resource allocation: Distributed algorithms

A FORMALIZATION OF the fair resource allocation problem was done and its structure was studied in the previous chapter. Particularly, we presented a minimal value for the allocation of bandwidth to network paths in form of a lower bound (Theorem 2). At present state, the interest of this lower bound appears in that the search space for the optimal solution can be reduced to the upper right region defined by this point, giving more structure to the problem that is now bounded (away from zero). Another important interest of this bound will be introduced in the next chapter. But for now, the natural next step is to ask ourselves how the problem can be solved efficiently in distributed systems. We remind that the SDN controller is physically distributed. This means that the global view of the network is logically available, but in fact achieved through information sharing between a group of controllers each in charge of a (partially or entirely) exclusive network domain. The distribution of the SDN (global) controller is a scenario that was described in Chapter 1, where we identified three main requirements for our α-fair resource allocation algorithm:

1. *Real-time:* converging to a "good" solution in a small number of iterations,
2. *Feasibility:* producing *feasible* solutions at all iterations,
3. *Distributivity:* being massively parallelized.

Thus, the SDN domains should enforce goals 1–3 cooperatively by solving *local* sub-problems (with as sole information the internal knowledge of their own domain parameters, plus the information that they can possibly gather from their peers - or a central synchronization master - through inter-domain communications), and broadcasting their partial results such that global iterates (good quality feasible iterates) or solutions can be rebuilt easily by all the controllers. The cooperation of the different controllers is

unavoidable in a distributed setting when the problem to solve at hand is not trivially separated. Of course, if all the paths that exist in the network do not cross domains, the α-fair resource allocation problem becomes separated into P independent sub-problems (where P is the number of controllers/domains). This is why in the sequel, we always assume each domain serves at least one path that crosses it but is not contained in it.

To sum up, we look for algorithms that can decompose adequately into sub-problems fitted to the domain structures, typically in terms of topology. Also, the iterative algorithms should demonstrate good rate of convergence so that a minimal number of communication rounds (that come at non trivial costs in terms of delay) are needed in order to achieve good quality solutions. By good quality solutions, we mean feasible solutions with acceptable accuracy in terms of optimality gap. At last, the algorithms should have efficient and simple update rules.

The present chapter is organized as follows. In Section 3.1, we review the work around algorithms designed to tackle the general α-fair resource allocation problem or one of its particulars. Specifically, we present an algorithm based on the classic dual decomposition method introduced by Voice [41] and Kelly, Maulloo, and Tan [36] that could be considered as the State-of-the-Art for this problem. Further, we dedicate Section 3.2 to an introduction on the ADMM, the tool that we use to design our distributed algorithms. Let us say in passing that the curious reader may also refer to Boyd et al. [23] for a very complete theory. Finally, we present our algorithm, FD-ADMM, as the tool to answer all our challenges, and we illustrate its performance by comparing it against the State-of-the-Art dual decomposition based algorithm. From now on, we drop some heavy notations and write simply $f_r(x_r) \coloneqq f_r^\alpha(w_r, x_r)$. The value of α will always be general or specified, and the weight w_r remembered through the subscript r.

IN A NUTSHELL

In this chapter, we review existing algorithms that compute α-fair resource allocations. Based on the identified goals that algorithms designed for distributed SDN should tackle in real-time allocation problems, we develop an ADMM-based method (FD-ADMM, Algorithm 3) for the 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 in each domain. This yields a convergence rate that does not depend on the partitioning of the network into domains. FD-ADMM is a distributed version of a generic centralized algorithm offered by the ADMM framework and present the general tricks to perform such a distribution.
We will show 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 2), a property that standard primal-dual decomposition methods generally lack. This permits to adjust within up to a few milliseconds the bandwidth of flows that evolve quickly and need immediate response. Finally, we compare FD-ADMM with the State-of-the-Art (SoA) approach for computing α-fair resource allocations in distributed scenarios in [22], which is based on the Lagrangian dual splitting method and that as from now we refer to as LAGR. We show that our algorithm outperforms LAGR in terms of convergence rate, feasibility preservation and hence, overall, real-time responsiveness.

3.1 Related works

Works around resource allocation and algorithms to compute them have been arising for the past decades in various application domains.

The probably most important (and known) notion of fairness would be max-min fairness. As we remarked in the last chapter, its definition brings a more natural and acceptable notion of what one can call “fair". As such, we have seen in the last chapter that computing the max-min fair allocation is not quite difficult: a finite-time algorithm based on the water-filling principle is available. Unfortunately, its design makes it naturally a centralized algorithm, and as such in traditional networks, its implementation was problematic. This is why some early notable works on max-min fairness were focussed on proposing algorithms that could work distributively and asynchronously to adapt to traditional networks, where centralized management was not yet available. For instance, Charny, Jain, and Clark [42] propose an asynchronous distributed algorithm that communi-

cates explicitly with the sources and pays some overhead (between the network switches) in exchange for more robustness and faster convergence. Later in [43], a distributed algorithm is defined for the weighted variant of max-min fair resource allocation problem in MPLS networks, based on the well-known bottleneck property, that we recalled and used in Chapter 2, stating that an allocation is max-min fair if and only if each *Label-Switched Path* (LSP) either admits a bottleneck link among its used links or meets its maximal bandwidth requirement. With the apparition of SDN and its ability to tackle these problems with a global viewpoint, the problem of Network Utility Maximization (NUM) was more and more relevant to the network resource allocation challenges. The idea was that instead of implementing local algorithms that would globally perform (or converge to an a-like behavior with guarantees to ensure) a certain fairness objective, it was possible to compute directly the optimal solution, because the vision was now global, on the links, the capacities, the traffic matrices. Therefore, the NUM problem 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 [41], then McCormick et al. [22], tackle the α-fair resource allocation problem with a gradient descent applied to the dual of the problem. The derived algorithm is today known and commonly referred to in the literature as Kelly's algorithm.

Dual gradient descent based algorithm

In the last chapter, the problem of α-fair resource allocation was formulated in terms of a concave maximization (or equivalently, convex minimization) problem under linear inequality constraints. In [22], the Lagrangian dual of the problem was formed:

$$\mathcal{L}(\mathbf{x}, \mathbf{u}) = \sum_{r \in \mathcal{R}} f_r(x_r) + \mathbf{u}^T(\mathbf{c} - \mathbf{A}\mathbf{x}). \tag{3.1}$$

Then, the standard dual algorithm to solve the NUM (as surveyed by Palomar and Chiang [44]) problem was applied and gave the following update rules:

$$\begin{aligned} \text{(primal) } x_r &\leftarrow \underset{x>0}{\operatorname{argmax}}\, f_r(x) - \left(\sum_{j \in \mathcal{J}_r} u_j\right) x \quad \forall r \in \mathcal{R} \\ \text{(dual) } u_j &\leftarrow \left[u_j - \lambda_j\left(c_j - \sum_{r \in \mathcal{R}_j} x_r\right)\right]_+ \quad \forall j \in \mathcal{J}, \end{aligned} \tag{3.2}$$

where $\lambda_j > 0$ is a sufficiently small step-size (that can depend on j in all

generality). In [22], the step-size λ_j is chosen as:

$$\lambda_j = \frac{u_j}{2c_j}, \tag{3.3}$$

following [41].

The dual algorithm thus designed seems extremely simple to carry: each update is fully separable, either with respect to $\mathcal{R}$, or with respect to $\mathcal{J}$, and can work very efficiently on massively parallel hardware. It was indeed tested on FPGA hardware and demonstrated time performances measurable at the scale of the millisecond, instead of seconds or minutes. The accuracy of a trick to condition the multi-path version was also demonstrated to reach a 1% root-mean-square error within the millisecond.

Certainly, the major contribution of the works around dual decomposition method is that the algorithm is very simple to implement, has update rules with very limited computation cost and massively parallelized, and therefore, can propose extremely fast and highly accurate solution. Notwithstanding, a fast and accurate solution should not necessarily be the only criteria that justifies its adaptability to real-time scenarios. We pointed out the fact that feasibility was a non negligible property in distributed scenarios. Indeed, a central projection sub-routine being not trivially available (unless, of course, it is implemented as a distributed iterative algorithm in the same way as the fair share computation itself), the availability of a solution that respects the link capacity constraints is highly desirable. Not forgetting in real-time scenarios where the problem parameters are changing continuously, sometimes abruptly, a whole process to compute *the* optimal solution of a fixed problem may be irrelevent, because at the time the optimal solution is finally available, the problem parameters have changed and the solution is no longer optimal: the long term performance of the system might be better if feasible close to optimal solution were implemented on a shorter time scale. This preference can justified in fields of machine learning, where optimality to high accuracy does not considerably/necessarily improve interpolations compared to modestly accurate solutions. It happens that penalizing the constraints through the classic Lagrangian relaxation as occurs in the dual decomposition method does not permit to avoid feasibility preservation. In fact, the iterates generically approach the optimal solution while being super-optimal, that is, from the outside of the feasible set, providing upper-bounds on the optimal value of the problem. Of course, the projected gradient algorithm permits to avoid this phenomenon, but again, it requires a central projection algorithm, which is not available in our case.

In the works 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. In fact, the computation time to optimum can be a valid argument to swipe away this question and justify its adaptability to real-time scenarios. Nonetheless, an implementation on a distributed system makes not only the computation time

important, but also the number of required iterations to reach convergence. As one iteration equals (at least) one communication round between the different agents, which comes at a non trivial cost, the time-to-value itself of the optimal point may be a burden if the transient solution is not feasible and optimality has to be reached so that feasibility is at least approached. For these reasons, real-time feasibility should somehow, in distributed real-time scenarios, be more precious than extremely accurate optimality. Regarding this topic on max-min fairness, the authors of [45] 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 [46] provide a feasibility preserving version of Kelly's methodology from [36]. 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 works contribute to this problem by proposing an efficient *real-time* version of the slave process, for any topology, preserving feasibility at each (slave) iteration. Among approximative approaches, one can quote the works [38] 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. In the same vein, the algorithm we derive solves the problem optimally and reaches feasibility as from the first iteration from any starting point.

3.2 Presentation of ADMM

In this preliminary section, we present the main definitions and assumptions required to design our algorithm based on ADMM.

3.2.1 General principles and challenges

The Alternating Method of Multipliers (ADMM) is an optimization framework dedicated to convex programs. Its main feature is that it adapts elegantly to a convex problem's separable structure, and permits to break down optimization stages. The framework is based on augmenting the classical dual Lagrangian by a penalization of the constraint violations by means of its squared 2-norm.

The work around ADMM is currently flourishing. The $O(\frac{1}{n})$ best known convergence rate of ADMM [47] failed to explain its empirical fast convergence until very recently, for instance in [48], 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[1] λ in the augmented Lagrangian formulation (see Section 3.2.2 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 [48], 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 derive a near-optimal tuning of ADMM 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 [49] and resource allocation in wireless virtual networks [50] but do not address this fundamental detail.

To the best of our knowledge, we are the first to show how ADMM can help designing real-time distributed algorithms for computing α-fair resource allocations in distributed settings. We are also the first to exhibit a near-optimal convergence rate of ADMM in this situation with our reciprocal penalty parameter adaptation scheme.

3.2.2 ADMM algorithm

In this paragraph, we summarize the principles of ADMM and show the unaware reader how the algorithm is designed. We also show how one can intentionally breakdown sub-problems in a coupled problem by introducing artificial separability. Of course, separability always come with a price when it is not natural, and we will qualify the tradeoffs later on.

Optimization model

We present the general optimization model that ADMM naturally addresses and the general definitions assumptions we need. Let n, m, p be positive integers.

[1] In the literature, what we usually call *penalty parameter* is the number $\rho = \frac{1}{\lambda}$. The number λ is therefore the reciprocal penalty parameter. Here, we arbitrarily adopt the notation λ and abusively still call it penalty parameter to avoid too much heaviness.

DEFINITION 2
Let $f : \mathbf{R}^n \to \overline{\mathbf{R}}$ be an extended real-valued function.
We say that f is *convex*, if for $x_1, x_2 \in \mathbf{R}^n$, and $t \in [0, 1]$,

$$f(tx_1 + (1-t)x_2) \leqslant tf(x_1) + (1-t)f(x_2).$$

We say that f is *lower semi-continuous*, or *closed*, if for all $\xi \in \mathbf{R}$, the set

$$\{x : f(x) \leqslant \xi\}$$

is closed.
We denote $\mathrm{dom}(f)$, and we call *domain of* f, the set $\{x \in \mathbf{R}^n : f(x) < \infty\}$.
We say that f is *proper* if $\mathrm{dom}(f) \neq \varnothing$.

Now, we assume $g : \mathbf{R}^n \to \overline{\mathbf{R}}$ and $h : \mathbf{R}^m \to \overline{\mathbf{R}}$ are two functions satisfying the assumption:

Assumption 1. The functions g and h are convex, closed, and proper.

Let $\mathbf{A}$ and $\mathbf{B}$ be two matrices of size (p, n) and (p, m), respectively, and $b \in \mathbf{R}^p$. The ADMM applies to the following generic optimization model:

$$\begin{aligned} &\min_{x,y} g(x) + h(y) \\ &\text{s.t. } \mathbf{A}x + \mathbf{B}y = b \end{aligned} \tag{$\mathcal{A}_0$}$$

In this thesis, the models in question are re-formulated in the particular case where $n = m = p$, and $\mathbf{A} = -\mathbf{B} = \mathbf{I}$ (the identity matrix of the appropriate dimension) and $b = 0$. For this reason, we avoid too much heaviness in the discussion and place ourselves directly in this particular case. For a complete presentation, the reader is of course referred to the reference in this topic [23].

Assumption 1 defines the minimal conditions under which the following definition is well-posed:

DEFINITION 3 (Proximity Operator)
Let $f : \mathbf{R}^n \to \overline{\mathbf{R}}$ be a function satisfying Assumption 1. Then, for all $\mathbf{u} \in \mathbf{R}^n$, there exists an unique minimizer $\mathbf{x}_\mathbf{u}^f \in \mathbf{R}^n$ to the problem:

$$\operatorname{argmin} f(\mathbf{x}) + \frac{1}{2}\|\mathbf{x} - \mathbf{u}\|^2. \tag{3.4}$$

Therefore, the function $\mathbf{x} \mapsto \mathbf{x}_\mathbf{u}^f$ is well-defined and we denote it $\mathbf{x}_\mathbf{u}^f =: \operatorname{prox}_f(\mathbf{u})$. As f is proper, prox_f takes finite values at all point. Therefore, $\operatorname{dom}(\operatorname{prox}_f) = \mathbf{R}^n$. Remarkably, if $\mathcal{C}$ is a convex non-empty set, and f is its indicator function:

$$f(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x} \in \mathcal{C} \\ \infty & \text{otherwise} \end{cases}, \tag{3.5}$$

Then $\operatorname{prox}_f = P_\mathcal{C}$, the euclidean projection onto the set $\mathcal{C}$.

The general algorithm

The ADMM algorithm aims at finding a minimizer of the *augmented Lagrangian*:

$$\mathcal{L}_\lambda(\mathbf{x}, \mathbf{y}, \mathbf{u}) = g(\mathbf{x}) + h(\mathbf{y}) + \frac{1}{2\lambda}\|\mathbf{x} - \mathbf{y}\|^2 + \mathbf{u}^T(\mathbf{x} - \mathbf{y}), \tag{3.6}$$

where λ is a positive parameter. The vector $\mathbf{u}$ is a dual variable associated to the constraints $\mathbf{x} - \mathbf{y} = 0$. For convenience we can re-write $\mathcal{L}_\lambda$ in the form:

$$\mathcal{L}_\lambda(\mathbf{x}, \mathbf{y}, \mathbf{u}) = g(\mathbf{x}) + h(\mathbf{y}) + \frac{1}{2\lambda}\|\mathbf{x} - \mathbf{y} + \lambda\mathbf{u}\|^2 - \frac{\lambda}{2}\|\mathbf{u}\|^2. \tag{3.7}$$

Then, the following update rules are to be repeated till a suitable stopping criterion is satisfied, and one can check that they correspond to computing a proximity operator of Definition 3:

$$\mathbf{x}^{k+1} \leftarrow \operatorname*{argmin}_\mathbf{x} \mathcal{L}_\lambda(\mathbf{x}, \mathbf{y}^k, \mathbf{u}^k) = \operatorname{prox}_{\lambda g}(\mathbf{y}^k - \lambda\mathbf{u}^k) \tag{3.8}$$

$$\mathbf{y}^{k+1} \leftarrow \operatorname*{argmin}_\mathbf{y} \mathcal{L}_\lambda(\mathbf{x}^{k+1}, \mathbf{y}, \mathbf{u}^k) = \operatorname{prox}_{\lambda h}(\mathbf{x}^{k+1} + \lambda\mathbf{u}^k) \tag{3.9}$$

$$\mathbf{u}^{k+1} \leftarrow \mathbf{u}^k + \frac{1}{\lambda}\left(\mathbf{x}^{k+1} - \mathbf{y}^{k+1}\right) \tag{3.10}$$

In the above equations, it is important to notice that the update rules (3.8) and (3.9) actually replace the update rule in the classic method of multipliers that consists in minimizing $\mathcal{L}_\lambda$ with respect to $\mathbf{x}$ and $\mathbf{y}$ jointly. By doing two minimization stages for one variable each, the two separate update rules can be considerably more simple to solve than the joint update

rule of the method of multipliers. Also, the alternating directions variant adapts naturally to the separability properties of g and h, as a block, but also *individually*.

Separation patterns

Now, let us assume that g has a specific separable structure. Let $\Omega = (\Omega_p)_{p\in P}$ be a partition of $[1, n]$. This means that the Ω_p are disjoint and their union sums up to $[1, n]$. We say that g is *separable with respect to* Ω if g can be written in the form:

$$g(x) = \sum_{p\in P} g_p(x_p), \quad g_p : x_p = (x_\omega)_{\omega\in\Omega_p} \in \mathbf{R}^{\Omega_p} \to g(x_p) \in \mathbf{R}. \tag{3.11}$$

As a direct consequence, the proximity operator of g becomes, for $u = (u_p)_{p\in P}$:

$$\operatorname{prox}_g(u) = (\operatorname{prox}_{g_p}(u_p))_{p\in P}. \tag{3.12}$$

In other words, computing the proximity operator of g boils down to evaluating, separately and independently, the ones of each g_p and then concatenating them together. This interesting property forms the first building block of accurate distribution in ADMM.

Correctness

The convergence of ADMM has motivated numerous works and is still an area of active research today. In the present chapter, we do not focus on the speed of convergence of ADMM (see Chapter 4). However, one can establish (non-trivial, see [23], Appendix A) that the update rules (3.8)-(3.10) give iterates x^k, y^k, and dual iterates u^k that verify:

1. $x^k - y^k \to 0$
2. $g(x^k) + h(y^k) \to p^*$ (the optimal value of the problem)
3. u^k converges to an optimal dual point,

as $k \to \infty$, under the additional (mild) assumption that the classical Lagrangian function $\mathcal{L}_{\lambda=\infty}(x, y, u) := g(x) + h(y) + u^T(x - y)$ admits a saddle point. On top of this result, it is possible to guarantee that the primal variables x and y converge also to an optimal point, but with additional assumptions.

Stopping criteria

Let us consider the general primal-dual iterate (x^k, y^k, u^k), where k is the iteration number. The optimality conditions give the necessary and sufficient conditions for a primal-dual solution (x^*, y^*, u^*):

1. Primal feasibility: $Ax^* + By^* = b$, and

$\mathcal{R}$	set of paths
$\mathcal{J}$	the set of links
c_j	the capacity of the link $j \in \mathcal{J}$
w_r	the weight associated to path r
x_r	the resource allocation variable at path $r \in \mathcal{R}$
$\mathcal{J}_r$	the set of links the path r is crossing
$\mathcal{R}_j$	the set of paths that are crossing the link j
$\mathbf{A}$	link route incidence binary matrix: $a_{jr} = 1$ *iff* $j \in \mathcal{J}_r$

Table 3.1 | Terminology.

2. Dual feasibility: $0 \in \partial g(\mathbf{x}^*) + \mathbf{A}^T\mathbf{u}^*$ and $0 \in \partial g(\mathbf{y}^* + \mathbf{B}^T\mathbf{u}^*)$.

We define the primal and dual residuals, respectively, by:

$$r^k := \mathbf{A}\mathbf{x}^k + \mathbf{B}\mathbf{y}^k - \mathbf{b}, \qquad \text{and} \qquad s^k = \frac{1}{\lambda}\mathbf{A}^T\mathbf{B}(\mathbf{y}^k - \mathbf{y}^{k-1}). \tag{3.13}$$

Following Boyd et al. [23], these residuals permit to bound the optimality gap, up to a certain constant factor depending on the problem parameters, and converge to 0 as k grows, and therefore yield the practical stopping criteria:

$$\|r^k\| + \|s^k\| \leqslant \varepsilon. \tag{3.14}$$

Other than time limits, this is always the stopping criteria that will be used in the simulations.

3.3 Model Formulation

In this section, we reformulate the α-fair resource allocation problem as a convex optimization problem that matches the structure of $(\mathcal{A}_0)$. We use the tools described in the last section to formulate our problem in the canonical form of ADMM and design a first algorithm, C-ADMM, that solves our problem in a centralized fashion and that will be helpful to design our distributed algorithm. We use the same terminology as in Chapter 2 that we remind in Table 3.1.
We recall that our aim is to compute an α-fair allocation $\mathbf{x}$:

$$\max_{\mathbf{x}\geqslant 0, \mathbf{A}\mathbf{x}\leqslant \mathbf{c}} f(\mathbf{x}) \tag{3.15}$$

where the α-fair utility function f is defined as in Definition 1 (Chapter 2) and that we remind below.

$$f(\mathbf{x}) = \sum_{r=1}^{n} f_r(x_r),$$

where

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

We observe that the α-fair utility functions are non-decreasing, strictly concave, non-identically equal to $-\infty$, and upper semi-continuous. It is well-known that under these conditions, the function f admits a unique maximizer over any convex closed non-empty set.

From now on, we adopt the convex optimization terminology. Define for each $r \in \mathcal{R}$ the convex cost function $g_r : x_r \mapsto g_r(x_r) := -f_r(x_r)$. Then, $g := \bigoplus_r g_r = -f$ is a convex closed proper[2] function over $\mathbf{R}_+^{|\mathcal{R}|}$.

Let us introduce ι as the indicator function of the convex closed set $\{\boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{c}, x \geqslant \boldsymbol{0}\}$:

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

Then the α-fair allocation problem can equivalently be formulated as the following convex program in the form of $(\mathcal{A}_0)$:

$$\min_{x,z} \sum_{r \in R} g_r(x_r) + \iota(z), \tag{3.16}$$

$$\text{s.t.} \quad \boldsymbol{x} - \boldsymbol{z} = \boldsymbol{0}. \tag{3.17}$$

A centralized algorithm

The above formulation (3.16) naturally yields a centralized algorithm based on ADMM that can be expressed in the proximal form below, which we refer to as *Centralized* ADMM (C-ADMM) and in which the stopping criterion (3.14) is integrated.

Algorithm 1 Centralized ADMM (C-ADMM)

Input: Initial values $\boldsymbol{z}$, $\mathbf{u}$

1: **while** the stopping condition (3.14) is not met **do**
2: $\quad \boldsymbol{x} \leftarrow \text{prox}_{\lambda g}(z - \lambda \mathbf{u})$
3: $\quad \boldsymbol{z} \leftarrow P(\boldsymbol{x} + \lambda \mathbf{u})$
4: $\quad \mathbf{u} \leftarrow \mathbf{u} + \frac{1}{\lambda}(\boldsymbol{x} - \boldsymbol{z})$
5: **end while**

In Algorithm 1, $P = \text{prox}_{\lambda \iota}$ is the projection on $\{\boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{c}, \boldsymbol{x} \geqslant \boldsymbol{0}\}$, and $\mathbf{u}$ the dual variable. Now, the first step of Algorithm 1 (line 2) can be separated thanks to the separability property of the objective function. As g is fully separable, that is, $g(\boldsymbol{x}) = \sum g_r(x_r)$, the proximal update of line 2 takes the trivially parallel form:

$$\forall r \quad x_r^{k+1} = \text{prox}_{\lambda g_r}(z_r^k - u_r^k) \tag{3.18}$$

[2] See the definitions in the Assumptions of the previous section.

such that each local variable x_r can be computed separately.

Through expression (3.18), we are thus able to provide an efficient update rule for $\mathbf{x}$, provided that the separate proximal computations are inexpensive. However, two main issues arise.

a) First, an update of the variable $\boldsymbol{z}$ 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 a *fundamental 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.

b) Moreover, although the convergence of C-ADMM may be fast (see Section 3.5.1, and Chapter 4 for further details), computation time may be a burden for its application due the successive summoning of a projection subroutine that would not scale with respect to the problem size. One must keep issue *a)* in mind that makes a global projection subroutine impossible to compute unless another decentralized scheme is triggered by the set of controllers to iteratively compute its solution in inner communication rounds. More specifically, to compute a global projection in order to deliver the solution of line 3, the distributed controller engages in an inner loop of computations and communications, to end up with the *sole update of the variable* $\boldsymbol{z}$. This therefore gives rise to a double loop algorithm where each iteration requires the convergence of an inner process that can be time-consuming (remember that communications between controllers can only be done at a non-trivial cost). Finally, let us remark that 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 [51].

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

3.4 Toward a distribution that respects the domain structure

A physically distributed SDN controller separates the physical network's topology into domains and assigns each domain to a corresponding domain controller. We thus assume that the set of links $\mathcal{J}$ is split into subsets $J_p, p = 1 \dots P$ such that $(J_p)_p$ forms a partition of the set of links J. Let R_p be the set of paths traversing the domain J_p via some link $j \in J_p$. More formally, $R_p = \{r \in \mathcal{R} : \exists j \in J_p \text{ s.t. } j \in \mathcal{J}_r\}$. Hence, $(R_p)_p$ forms a covering of $\mathcal{R}$. Let ι_j denote the indicator function for the capacity contraint of link $j \in \mathcal{J}$, i.e.,

$$\iota_j(\mathbf{x}) = \begin{cases} 0 & \text{if } \sum_{r \in \mathcal{R}_j} x_r \leqslant c_j, \mathbf{x} \geqslant 0, \\ \infty & \text{otherwise.} \end{cases} \tag{3.19}$$

Thus, we can define the indicator function for the capacity constraints of

the links within a domain p as:

$$\iota_p := \sum_{j \in J_p} \iota_j. \tag{3.20}$$

Specifically, an allocation vector x satisfies the capacity constraints within domain p if, and only if, $\iota_p(x) = 0$, which is in turn equivalent to $\iota_j(x) = 0, \forall j \in J_p$.

Also, let us define $S_p := \mathrm{dom}(\iota_p)$. Thus, for each $p \in P$, S_p is the (convex, closed) capacity set of the associated to domain p. Finally, for $p \in P$ and $z \in \mathbf{R}^{|\mathcal{R}|}$, $\mathrm{PRO}(p, z)$ denotes the Euclidean projection of z onto S_p.

3.4.1 Consensus form

We can now reformulate our objective to a fully separable form. We write $I_r = \{q \in [1, P] \quad r \in R_q\}$ to design the set of domain indices which r traverses. We say that the path r traverses a domain p when there exists a link $j \in J_p$ such that $r \in \mathcal{R}_j$. In the same fashion as in Section 3.3, we plug the feasibility constraints into the objective by means of the indicator functions ι_p. Problem (3.16), (3.17) becomes equivalent to:

$$\min \sum_{r \in R} g_r(w_r, x_r) + \sum_{p \in P} \iota_p(x). \tag{3.21}$$

In order to fully benefit from the canonical structure from Section 3.2.2, we first artificially give a separable structure that is adapted to the domain distribution. Thus, we create a copy of the variable x, let us say z, for the term $\sum_{p \in P} \iota_p(x)$ and impose the equality constraint $x - z = 0$. We go a little bit further by creating a special variable $z^p = (z_r^p)_{r \in R_p}$ for each domain p and by adding the equality constraints

$$z_r^p - x_r = 0, \forall r \in \mathcal{R}, \forall p \in I_r. \tag{3.22}$$

Each variable z^p will be handled exclusively by its corresponding domain p.

Under the condition (3.22), we have:

$$\sum_{r \in R} g_r(x_r) = \sum_{p \in P} \sum_{r \in R_p} \frac{1}{|I_r|} g_r(z_r^p). \tag{3.23}$$

We can thus write the objective function as follows, where the variable z is the concatenation of all the z^p:

$$G(z) = \sum_{r\in R} g_r(x_r) + \sum_{p\in P} \iota_p(z^p) \tag{3.24}$$

$$= \sum_{p\in P} \left\{ \sum_{r\in R_p} \frac{1}{|I_r|} g_r(z_r^p) + \iota_p(z^p) \right\} \tag{3.25}$$

$$=: \sum_{p\in P} g_p(z_p). \tag{3.26}$$

To sum up, *we have artificially separated the objective function by creating a minimal number of copies of the primal variable* $\mathbf{x}$. Now, instead of a global resource allocation variable, several copies of the variable account for how its value is perceived by each domain. To enforce an 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 a domain p has a cost g_p. As we separated the global objective on purpose, the separability property of the proximal operator thus gives the following:

$$\text{prox}_{\lambda G}(u) = \left((\text{prox}_{\lambda g_p}(u_r^p))_r, (\text{prox}_{\lambda \iota_p}(\mathbf{u}^p)) \right).$$

These definitions permit next to write the distributed consensus model where each agent only has access to local information. Now that the objective is separated according to the network domains, we mimic the form of $(\mathcal{A}_0)$ by posing the following optimization problem:

$$\begin{aligned} \min & \sum_{p=1}^{P} g_p(z^p) + \chi(z'^p) \\ \text{s.t. } & z^p = z'^p \quad \forall p = 1 \dots P, \\ & z^p = (z_r^p)_{r\in R_p}, \\ & z'^p = (z'^p_r)_{r\in R_p}, \end{aligned} \tag{3.27}$$

where χ is the indicator function of the *consensus set*

$$\mathcal{X} := \{z' = (z'^p)_{p=1\dots P} : \forall r \in \mathcal{R}, \forall p, q \in I_r, z'^p_r = z'^q_r\},$$

the Euclidean projection[3] on which we denote $P_{\mathcal{X}}$. Finally, the construction of last section applies to this formulation, and one gets the following update

[3]Remark: In fact, the operator maps the vector z'^p to a vector $\mathbf{x}$ of the same size, that is, $\mathbf{x} = (\mathbf{x}^p)_{p=1\dots P}$ with $x^p \in \mathbf{R}^{R_p}$ that verifies the consensus constraints. We therefore do not lose any information by keeping one coordinate only for each path r and thus post-restricting $P_{\mathcal{X}} : \bigoplus_p \mathbf{R}^{R_p} \to \mathbf{R}^{\mathcal{R}}$, instead of having a larger vector with identical entries $(x_r^p)_{p\in I_r}$.

rules, where $\mathbf{u}$ is the dual variable associated to the constraints $z = z'$:

Algorithm 2 Distributed ADMM (D-ADMM)

Input: Initial values z, $\mathbf{u}$

1: **while** the stopping condition (3.14) is not met **do**
2: $\quad \forall p \in P \quad z^p \leftarrow \text{prox}_{\lambda g_p}(\overline{z}^p - \lambda \mathbf{u}^p)$
3: $\quad \overline{z} \leftarrow P_{\mathcal{X}}(z + \lambda \mathbf{u})$
4: $\quad \mathbf{u} \leftarrow \mathbf{u} + \frac{1}{\lambda}(z - \overline{z})$
5: **end while**

Of course, Algorithms 1 (C-ADMM) and 2 (D-ADMM) have the same form. The main difference is that in D-ADMM, the "central projection" is in fact the Euclidean projection $P_{\mathcal{X}}$ onto the consensus set $\mathcal{X}$, which is simply the average point of all coordinates:

$$\forall z = (z^p)_{p=1\ldots P} \text{ such that } z^p \in \mathbf{R}^{R_p}, \quad P_{\mathcal{X}}(z)_r = \frac{1}{|I_r|} \sum_{q \in I_r} z_r^q. \tag{3.28}$$

This difference is *fundamental* in our design. The complicating (coupling) constraints $\mathbf{A}x \leqslant \mathbf{c}$, that makes C-ADMM fundamentally centralized, have been broken down and integrated to the objective function while respecting the domain structure. Now, each domain will handle the constraints corresponding to the link belonging to them separately. Also, and this is a major improvement, the communication round between the different domains boils down into computing an average, whereas before, it would require the summoning of an iterative method based on numerous communication rounds (see *b)*). Thus, one can see that D-ADMM, has a much simpler multi-agent structure, as the only work that requires communication happens to be the less complicated work to do. Of course, we are assuming that each domain p has an efficient way of evaluating the operator prox_{g_p}. This means that each controller solves the sub-problem:

$$\begin{aligned} \min \sum_{r \in R_p} \frac{w_r}{|I_r|} g_r(z_r^p) + \frac{1}{2\lambda} \|z^p - (\overline{z}^p - \mathbf{u}^p)\|^2 \\ \text{s.t.} \sum_{r \in \mathcal{R}_j} z_r^p \leqslant c_j \quad \forall j \in J_p. \end{aligned} \tag{3.29}$$

In other words, each domain solves a regularized (with the presence of the squared 2-norm) version of the original problem. Although smaller, the problem itself was not simplified. This is why we further decompose the problem in order to get explicit update rules.

3.4.2 Fast Distributed ADMM

This time, we push the distribution to the link level. For each link $j \in \mathcal{J}$, let $z_j \in \mathbf{R}^{\mathcal{R}_j}$ be a copy of the variables $(x_r)_{r\in\mathcal{R}_j}$. Then , we can equivalently express our problem as follows:

$$\begin{aligned} \min H(\boldsymbol{x}, \boldsymbol{z}) &\coloneqq \sum_{r\in\mathcal{R}} g_r(x_r) + \sum_{j\in\mathcal{J}} \iota_j(z_j) \\ z_{jr} &= x_r \quad \forall r \in \mathcal{R}_j \quad \forall j \in \mathcal{J} \end{aligned} \tag{3.30}$$

We apply ADMM again to this formulation. Now, the objective that replaces the one in the update 2 is the whole objective function in (3.30). Indeed, we have separated sufficiently the indicator functions to obtain a fully separable proximity operator:

$$\mathrm{prox}_{\lambda H}(\boldsymbol{v}, \boldsymbol{u}) = \Big((\mathrm{prox}_{\lambda g_r}(v_r)_{r\in\mathcal{R}}, (\mathrm{PRO}(j, \boldsymbol{u}^j))) \Big) \tag{3.31}$$

Just like for the design of D-ADMM, we plug in the consensus constraints into the objective via the introduction of copies of the variables $(\boldsymbol{x}, \boldsymbol{z})$, and end up with the canonical form of ADMM that follows:

$$\begin{aligned} &\min\ H(\boldsymbol{x}, \boldsymbol{z}) + \chi(\boldsymbol{x}', \boldsymbol{z}') \\ &\text{s.t. } \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{z} \end{pmatrix} - \begin{pmatrix} \boldsymbol{x}' \\ \boldsymbol{z}' \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \end{aligned} \tag{3.32}$$

where χ is again the indicator function of the set $\mathcal{X} \coloneqq \{(\boldsymbol{x}, \boldsymbol{z}) : z_{jr} = x_r \quad \forall r \in \mathcal{R}_j \quad \forall j \in \mathcal{J}\}$. Again, we define the operator $\mathbf{P}_{\mathcal{X}}$ as the following. For the generic variable $(\boldsymbol{x}, \boldsymbol{z}) \in \mathbf{R}^{\mathcal{R}} \times \times_{j\in\mathcal{J}} \mathbf{R}^{\mathcal{R}_j}$, let $(\overline{\boldsymbol{x}}, \overline{\boldsymbol{z}})$ be its projection onto the consensus constraint set of (3.30). Then,

$$\tilde{\boldsymbol{z}} \coloneqq P_{\mathcal{X}}(\boldsymbol{x}, \boldsymbol{z}) \in \mathbf{R}^{\mathcal{R}} \text{ is the vector defined by } \tilde{z}_r = \overline{x}_r = \overline{z}_r^j \quad \forall j \in \mathcal{J}_r, \forall r \in \mathcal{R}. \tag{3.33}$$

Then, one has simply:

$$P_{\mathcal{X}}(\boldsymbol{x}, \boldsymbol{z}) = \left(\frac{x_r + \sum_{j\in\mathcal{J}_r} z_r^j}{|\mathcal{J}_r| + 1} \right)_{r\in\mathcal{R}}. \tag{3.34}$$

Using the distributive properties, we obtain Algorithm 3 (Fast Distributed (FD)-ADMM).

This yields the simple update rules at lines 3 and 12[4].

[4]These updates rules are also simplified using the straightforward fact that the sum $\sum_{l\in r} u_{lr}$ is constant. It can thus be fixed to 0 by initialization.

Algorithm 3 Fast Distributed ADMM (FD-ADMM)

1: **procedure OF** DOMAINp
2: RECEIVE $z_{qr} \quad \forall q \in I_r \quad \forall r \in R_p$
3: $\tilde{z}_r \leftarrow \frac{1}{|\mathcal{J}_r|+1}\left(\sum_{q\in I_r} z_{qr} + x_r\right) \quad \forall r \in R_p$
4: **for** $j \in J_p$ **do**
5: $u_r^j \leftarrow u_r^j + \frac{1}{\lambda}(z_r^j - \tilde{z}_r) \quad \forall r \in R_j$
6: $z^j \leftarrow$ LINK_PROJ$(j, \tilde{z} - \lambda u^j)$
7: **end for**
8: **for** $r \in R_p$ **do**
9: $v_r \leftarrow v_r + \frac{1}{\lambda}(x_r - \tilde{z}_r)$
10: $x_r \leftarrow \text{prox}_{\lambda g_r}(\tilde{z}_r - \lambda v_r)$
11: **end for**
12: SEND $z_{pr} = \sum_{j\in\mathcal{J}_r\cap J_p} z_{jr}$ and $z_{*pr} = \min_{j\in J_p} z_{jr}$ to domains $q \in I_r \quad \forall r \in R_p$
13: **end procedure**

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 at lines 2 and 12. In these steps, the domains gather from and broadcast to adjacent domains the sole information related to paths that they share in common. In particular, domains are blind to paths that do not traverse them, and can keep their internal paths secret from others. In details, after each iteration of the algorithm, each domain p receives the minimal information from other domains such that it is still to compute a local value z_{pr} that is destined to be broadcasted again in order to build the global consensus value $\tilde{z}_r$. The domain p has to send the value z_r^p back to neighboring domains within I_r only.

Communication overhead: 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. 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 10 in Algorithm 3) locally. 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 [22] and stated in Section 3.1, 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.

Feasibility preservation

As we just saw, a potential drawback of the distributed approach is the potential feasibility violation by the iterate $\tilde{z}^k$. After all, this is a major appre-

ciable feature of C-ADMM and the fact that global anytime feasibility, had to be traded for distribution through D-ADMM and FD-ADMM is a point that needs to be accounted for. However, we have the following positive result.

PROPOSITION 2
D-ADMM and FD-ADMM provide sequences of feasible points that converges to the optimum.

Proof. The arguments are the same for both algorithms. We only expand the proof for FD-ADMM. For any link j, we have by line 6 of Algorithm 3 that z^j is feasible in link j. That is, $\sum_{r\in j} z_{jr} \leqslant c_j$. Define $z_{*r} = \min_{j\in J_r} z_{jr}$. Then, for each link j:

$$\sum_{r\in j} z_{*r} \leqslant \sum_{r\in\mathcal{R}_j} z_{jr} \leqslant c_j. \tag{3.35}$$

Thus, no capacity is violated by the allocation z_{*r}. At the optimum, the consensus is reached. This means that $\lim z_{*r} = \lim x_r = \lim z_r^j, \forall j \in \mathcal{J}_r$. Thus z_{*r} forms a feasible sequence that converges to the optimum. □

Thus, in a certain way, for sufficiently loaded and communicating domains (i.e. the $|R_p \cap R_q|$ are large enough) we might sacrifice some overhead (counted on a per iteration basis) compared to standard dual methods. However, the algorithm supports anytime feasibility, a major feature that dual methods do not generically provide (unless a global projection sub-routine is implemented, which we assumed is not the case).

3.4.3 Update rules: some precisions

In this paragraph, we specify the update rules of FD-ADMM. This permits us to show precisely the costs of the update rules sub-routine, and at the same time, we justify the benefits of the distribution: further than being flexible enough to adapt to the domain structure, the distribution that appeard in the design of ADMM permits one to end up with drastically simpler computations.

Proximity operators

Let $\alpha > 0$ and $r \in \mathcal{R}$ Let $u \in \mathbf{R}$. We want to evaluate the proximity operator with penalty λ of g_r at the point u. We have:

$$\text{prox}_{\lambda g_r}(u) = \underset{x\in\mathcal{R}}{\text{argmin}}\, g_r^(x) + \frac{1}{2\lambda}||x-u||^2. \tag{3.36}$$

Fact 3 ensures that the minimizer $x_{\lambda r}$ of the function exists and is unique. Moreover, by differentiability, it is a critical point of the function. Therefore,

$$\nabla g_r(x_{\lambda r}) + \frac{1}{\lambda}(x_{\lambda r} - u) = 0, \tag{3.37}$$

which gives:

$$x_{\lambda r}^{\alpha+1} - u x_{\lambda r}^{\alpha} + \lambda w_r = 0. \tag{3.38}$$

In other words, the evaluation of the proximity operator of g_r boils down to finding the root of a univariate function over $\mathbf{R}_+$. This can be done very efficiently with basic root finding algorithms. Particularly, (3.38) is a second order polynomial equation for $\alpha = 1$, a cubic equation for $\alpha = 2$ and a quartic for $\alpha = 3$. For all values of α, a straightforward function study shows the equation admits a unique positive real solution for which a closed form is derivable. In general, using integer values of α such as 1, 2 and 3 as an approximation of max-min fairness[5] is common and acceptable, and when a closed form is not available (in general for greater integer values of α) polynomial root finding algorithms can be even more efficient[6] than generic methods that work for all convex functions.

Link projections

The particular interest that comes with the distribution of the problem to the level of the links is that we end up with individual constraints taken separately. In FD-ADMM, instead multiple constraints defined polyhedron, we now have to project points onto single constrained (plus positivity constraints) polyhedron. These actually define (the positive regions of) simplices with a radius equal to the right hand side of the constraint. In [52], the authors give an algorithm to compute the exact projection onto a simplex of dimension q with a complexity equivalent to the one of sorting a list, that is, in average $\mathcal{O}(q \log q)$. We recall the projection algorithm here. A correctness proof and performance evaluation are available in [52].

Algorithm 4 Algorithm for LINK_PROJ.

Input: x^j **Output:** LINK_PROJ(x^j), the projection of x^j onto the feasible set of constraint j.

1. Sort the components of x^j as $\xi_1 \geqslant \ldots \geqslant \xi_N$ where N is the length of x^j
2. Set $k := \max\{\ell \text{ s.t. } \frac{\sum_{s=1}^{\ell} \xi_s - c_j}{\ell} < \xi_\ell\}$
3. Set $\tau := \frac{\sum_{s=1}^{k} \xi_s - c_j}{k}$
4. For all p set $x_p^j = [x_p^j - \tau]_+$.

[5] In this thesis, we consider only those three metrics.

[6] For instance, algorithms that compute the root of a polynomial can compute the eigenvalues of the companion matrix of the polynomial.

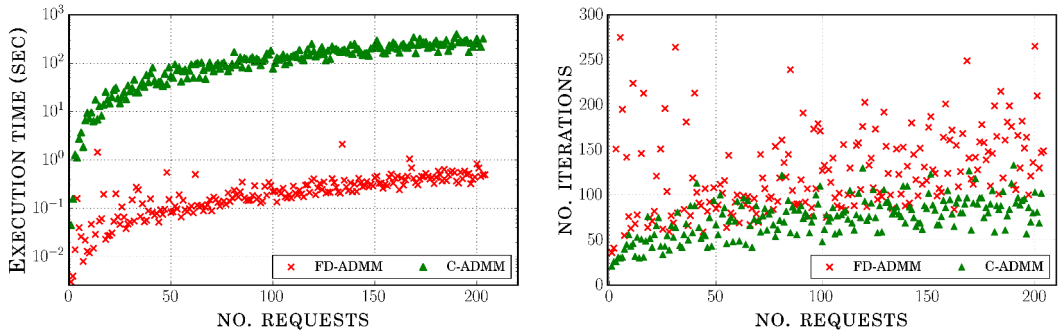


Figure 3.1 | CPU time and number of iterations required to reach the same level of residual tolerance for C-ADMM and FD-ADMM.

3.4.4 What level of distribution should we chose?

With the design of D-ADMM and FD-ADMM, one can see that any level of distribution is possible. One can separate the global problem into P independent sub-problems (where dP is the number of domains) and solve a consensus problem with P agents. Then, each agent has to solve independently the problem (3.29). The first globally feasible point is thus available after the agents solve this problem and communicate for the first time. Such a level of distribution is perfectly acceptable for the domain independence constraints in distributed SDN. However, it still requires a black box that solves the sub-problem (near-)optimally. For instance, this black box can be an implementation of a localized C-ADMM, or FD-ADMM.

3.5 Numerical results

In this section, we compare FD-ADMM with C-ADMM and LAGR, and evaluate, the respective convergence speed in the former case, and the responsiveness in terms of transient feasibility preservation in the latter. We are not focused on absolute performance in terms of time, but rather on relative improvements all our algorithms are implemented in Python.

Our algorithms are evaluated either on synthetic networks generated following the Barabasi-Albert model [39], or on the real world British Telecom (BT) topology that was used by McCormick et al. [22].

3.5.1 The cost of distribution

A first thing one wants to check is the cost, in terms of convergence, of the distribution from C-ADMM to FD-ADMM. Although the question of the convergence rates will be posed in the next chapter, we can already say here, as we will see, that the convergence of C-ADMM is linear with a rate that can be upper-bounded with the problem parameters. The question is, what do we lose in terms of convergence rate or speed by distributing the algorithm link-wise? One should expect that the convergence of FD-ADMM is

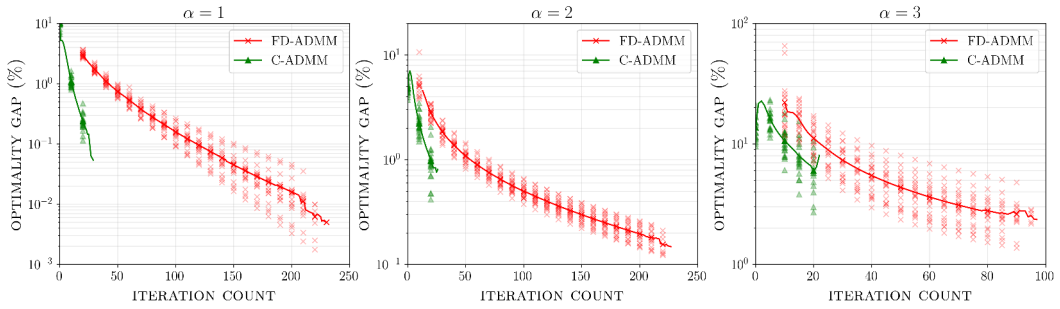


Figure 3.2 | Gaps versus iteration number for FD-ADMM and C-ADMM.

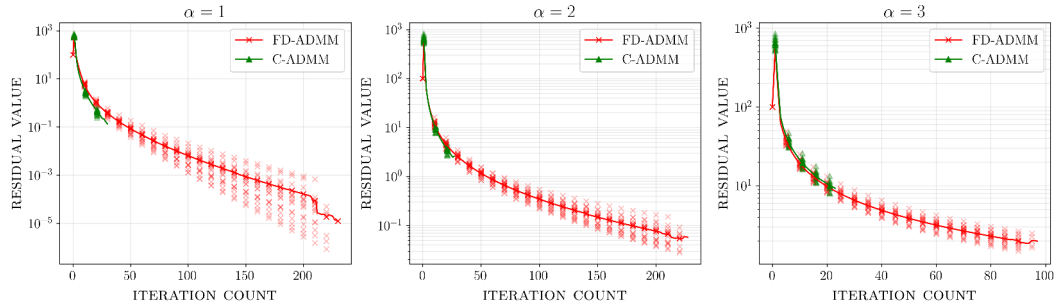


Figure 3.3 | Residual value versus iteration number for FD-ADMM and C-ADMM.

slower, but iterations can be done quicker. We demonstrate this fact by running simulations on the synthetic networks. They are small networks made of 10 nodes with attachment parameter 5. On these small networks, we generate 500 paths by choosing source-destination pairs uniformly at random as well as a corresponding weight uniformly in the interval $1\pm20\%$. The reason why we use such a setting is to generate highly congested networks with only active constraints so that the consensus is the most difficult to obtain in FD-ADMM. The link capacities are drawn uniformly at random within the interval $10\pm20\%$.

First, we look at the gap and residuals descent with time as the algorithms run. The algorithms are allowed to run for five seconds, at the end of which they output their last feasible solution (which is also the best). As is shown in Figures 3.2 and 3.3, the distributed version FD-ADMM requires more iterations to reach the same level of precision as C-ADMM in terms of optimality gap. However, within 5 seconds, FD-ADMM is able to perform drastically more iterations than C-ADMM. In 3.4 and 3.5, the central projection in C-ADMM is done by CPLEX; yet, the projection step time sums up to 80% of the overall computation time, which explains the slow performance of C-ADMM. Next, we plot, for a varying number of paths, the computation time and required iteration steps, to reach the same level of residual tolerance for both algorithms. The results are shown in Figure 3.1.

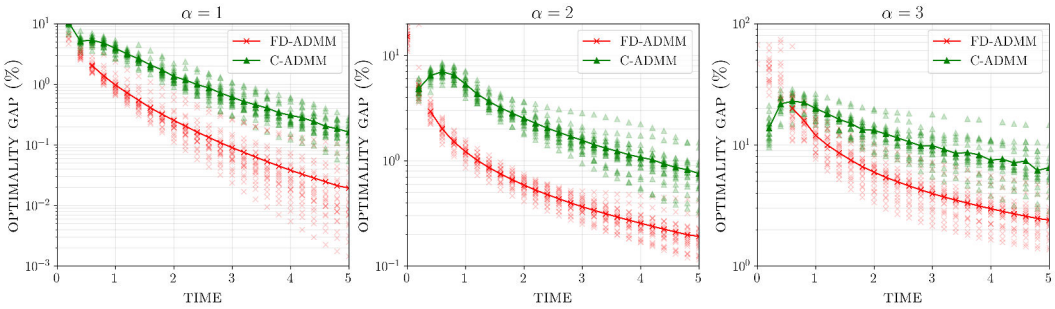


Figure 3.4 | Gaps versus time (in seconds) for FD-ADMM and C-ADMM.

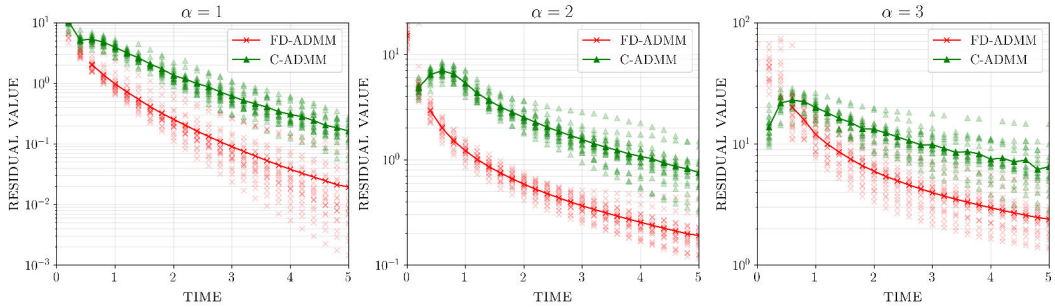


Figure 3.5 | Residual value versus time (in seconds) for FD-ADMM and C-ADMM.

3.5.2 Comparison against LAGR

We now compare the proposed FD-ADMM algorithm against LAGR (see Section 3.1), for all three values of α. 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 $(w_r^t)_{r\in R, t\in 0\ldots T}$ 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 3.6 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 whereas LAGR constantly violates the constraints as weights w_r change in real-time. Figure 3.7 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

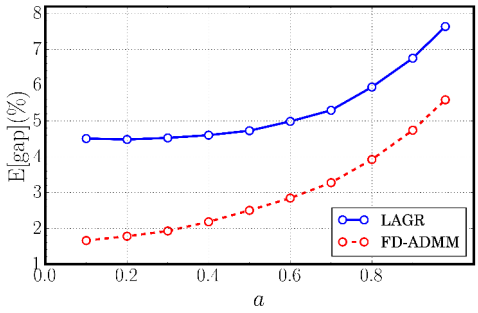


Figure 3.6 | Average optimality gap E[gap] vs. the variation amplitude a.

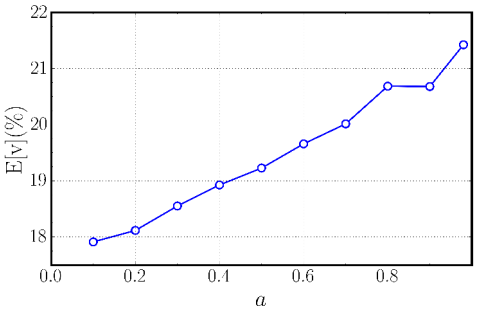


Figure 3.7 | Average percentage of violated constraints E[v] by LAGR vs. the variation amplitude a.

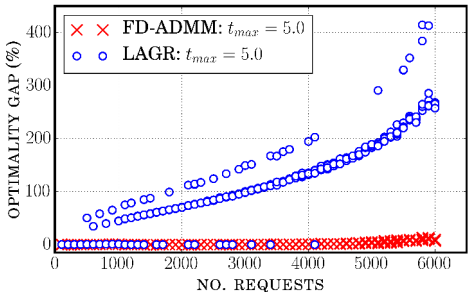


Figure 3.8 | Optimality gap of the best feasible point found after 5 seconds runtime.

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 3.8 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.

3.6 Concluding remarks

The distributed architecture of a logically centralized SDN controllers was brought in as a solution to many concerns in control plane design including scalability, resiliency, fault tolerance, *etc*. As a consequence to the development of such specific architectures, algorithms deployed in the control plane

have to be specifically designed to match best with the implied performance requirements. In this chapter, we identified two important goals that a real-time bandwidth allocation application should fulfill. First, the algorithm itself should be suitable for a multi-domain implementation that obeys the distributed architecture of the SDN controller. This means that it must be possible for each domain to handle the portion of the global problem that includes the local information that it detains exclusively, independent from the other domains. Also, this portion is embodied as a sub-problem that solving requires the summoning of a sub-routine that should be as simple and efficiently solvable as possible, so that its computation does not represent an overwhelming burden for the whole system's efficiency. Lastly, in a real-time scenario, anytime feasible solutions should be preferred to asymptotically optimal but transiently infeasible solutions, so that an intermediate but halfway good solution is always available at hand.

To respond to those conflicting requirements, we proposed FD-ADMM, a massively distributed algorithm, whose behavior does not depend on the domain distribution, that provides feasible solutions converging to the optimum at any iterations, and that demonstrates fast convergence rate.

In the next chapter, we will show how versatile FD-ADMM can be in adapting to feature additional complicating structural constraints. Moreover, it is important to mention that the algorithm is inspired from its centralized version that is not implementable in practice, but that is very useful to its design. Indeed, concerning the speed of convergence, we swept under the carpet a major technical detail that is the penalty parameter λ. It happens that its choice conditions highly the convergence speed of ADMM-based algorithms; a suitable design and implementation of an ADMM -based algorithm cannot neglect this fundamental detail. This is why the next chapter is also dedicated to address the technical considerations that lie beneath the penalty parameter selection in the runs of FD-ADMM. The reader will then see how C-ADMM and the results of Chapter 2 become even more useful.

Chapter 4

Extensions and refinements

We designed in the previous chapter FD-ADMM, a distributed algorithm that computes optimally the fair resource allocation problem on general capacitated networks and generally weighted flows. The algorithm is based on the ADMM that makes it efficiently adaptable to the distribution model of SDN and the separated structure of the fairness metrics. In the previous chapter, we observed a likely linear convergence rate of C-ADMM and comparable convergence speed of FD-ADMM (in terms of iteration count). Concerning a practical question on the simulations, all the tests were represented for a particular value of the penalty parameter λ that was not specified until now. Undoubtedly, the convergence of ADMM is highly impacted by the choice of the penalty parameter, and can be very bad if it is poorly conditioned (initialized, and/or updated). In this chapter, we investigate on these issues and propose a near-optimal choice of the penalty parameter for FD-ADMM. The choice is based on a linear convergence rate result for C-ADMM, which gives a rate of convergence that can be optimized with respect to λ.

Also, we propose to extend the α-fair resource allocation problem, that was presented and adressed with FD-ADMM in its most simple setting, to several practical use cases. The first obvious extension is the one to the multi-path setting. We can assume that a connection request[1] r can have more than one available paths between its source and destination, that are pre-computed and can carry the traffic. With the hypothesis that traffic is infinitely many times splittable, the problem can again be formulated as a convex optimization problem. As we will see, this extension does not change the objective function, but rather the feasible set of the problem.

The objective also can be adapted to account for different kinds of system requirements. We show how FD-ADMM can feature these requirements and take the particular example of sparse solution structure preferences.

[1] In this chapter, we change slightly the terminology and call $\mathcal{R}$ the set of *connection requests* (or shortly, requests) between a source and a destination, whereas the word *path* will be reserved for the actual paths that are available to the requests from their source to their destination.

IN A NUTSHELL

The convergence of ADMM is well-known to depend highly on the choice and update of the penalty parameter λ of the augmented Lagrangian form. In this chapter, we quantify this dependence for our centralized version of the resource allocation problem and adapt the results to provide a fine tuning of FD-ADMM that helps observing a near optimal convergence speed. Also, we show the versatility of the model to specific solution structures and give the corresponding algorithms to two possible examples of extensions: the multi-path setting and the sparse structure preferences on the solutions.

4.1 Convergence of ADMM

4.1.1 Background

The work around the convergence of ADMM has been flourishing during the past years. There is an extensive literature on the ADMM itself and its possible applications, in various distributed settings in machine learning, resource allocation, and classification (see for instance, [53], [54], [55]), but still quite few concerning its theoretical convergence. Goldfarb, Ma, and Scheinberg [56] show a general $\mathcal{O}(1/\varepsilon)$ convergence rate and an accelerated algorithm that converges in $\mathcal{O}(1/\sqrt{\varepsilon})$ to an ε-optimal solution, particularly in the case where only one of the two functions is smooth with Lipschitz continuous gradient. An extension of these results to different scenarios can be found in [57], where the authors consider various regularity assumptions and show the corresponding achievable performances. The different assumptions include strong convexity, Lipschitz continuity of the gradient of one, both or none of the functions, on top of the basic convexity assumption. Also, the full rankness (of the rows) of the constraint matrices $\mathbf{A}$ and $\mathbf{B}$ are additional assumptions that play a key role in the type of convergence. The authors show that R-linear convergence is achieved when one of the two functions is strongly convex and one of the two functions has a Lipschitz continuous gradient and the corresponding constraint matrix has full row rank.

Basically, there are two main approaches to tune the penalty parameter of ADMM, depending on the structure of the problem:

1. **Optimal penalty derivation:** Certain problems are particularly well structured and admit a (worst-case) convergence rate that can be expressed as a function of λ. In this way, the worst case guarantee can be optimized with respect to λ, which gives an optimal parameter λ^*. Of course, this optimal penalty guaranties the *worst case* convergence rate, and there is no guarantee that it is the value that gives the best possible convergence in practice. Nevertheless, this technique generally gives

very satisfactory results in practical implementations of ADMM-based algorithms. We will demonstrate this by simulations on our problem.

2. **Adaptive penalty:** In other problems, it is not possible, or too complicated, to derive a clear penalty value. To circumvent this problem, numerous works have been proposing adaptive penalty tuning schemes in order to accelerate the convergence. The idea is to initialize the penalty at a certain value λ_0 and then, adapt its value on-the-fly as the algorithm runs. The adaptive strategy has to be set and it is possible to still guarantee the convergence of the algorithm to an optimal point. Adaptive strategies for penalty parameter tuning are known to make the performance of ADMM less dependent of the initialization in practical implementations which justifies its use in many applications.

In the next paragraph, we review the principal results on the two aforementioned strategies.

Optimal penalty parameter

For a short survey of existing works on the penalty tuning of ADMM, we return to the general form of the 2-block formulation of ADMM that we remind here. Let $F : \mathbf{R}^n \to \mathbf{R}$ and $G : \mathbf{R}^m \to \mathbf{R}$ be two convex functions and $\mathbf{A} \in \mathbf{R}^{p\times n}$ and $\mathbf{B} \in \mathbf{R}^{p\times m}$ two matrices. Let $b \in \mathbf{R}^p$. We look at the problem:

$$\begin{aligned} &\min\ F(\mathbf{x}) + G(\mathbf{y}) \\ &\text{s.t. } \mathbf{M}\mathbf{x} + \mathbf{P}\mathbf{y} = \mathbf{b} \end{aligned} \tag{4.1}$$

The update rules of ADMM to solve (4.1) always give a convergent series to an optimal point for *any* fixed parameter λ. However, the speed of convergence will depend of the choice of this parameter. A way of addressing this question is to look for an actual expression of the convergence rates: a convergence rate depending on λ, if derivable, can be optimized, and the optimal value of λ can be deduced. The following result motivates this observation:

THEOREM 3 (Deng and Yin [48])
Assume that the problem (4.1) has a saddle point. Assume that $\mathbf{M}$ 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[a]

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

and λ is the penalty parameter in the augmented Lagrangian form.

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

And the corollary that directly follows:

COROLLARY 2
The optimal penalty parameter[a] is

$$\lambda_* = \sqrt{\frac{||\mathbf{M}||^2\lambda_{\min}(\mathbf{M}^T\mathbf{M})}{\sigma L}}. \tag{4.2}$$

[a] That minimizes the worst case convergence rate in the assumptions of Theorem 3.

Adaptive penalty tuning schemes

In numerous practical use cases, the conditions of Theorem 3 are not fulfilled. Many problem formulations do not fulfill the strong convexity assumption for instance, or even the full row-rank assumption that can be rare in applications. Except for linear programs where linear convergence of ADMM is still guaranteed, it is not possible to get a theoretically better convergence than the classic $\mathcal{O}(1/\varepsilon)$ worst case ε-approximation in general, even less an insight of what fixed parameter to use. This is why it makes sense to try an adapt the penalty parameter in real-time while the algorithm is running. The most popular adaptive penalty strategy is undeniably proposed by He, Yang, and Wang [58] and commonly referred to as *residual balancing*. We have seen in Chapter 3 how a valid stopping criteria can be used to detect the convergence of the algorithm. We need the primal and dual residual, r^k and s^k respectively (where k is the iteration count), to be both small. A typical event when the penalty parameter is badly chosen, is that one of the residual is substantially greater than the other. On the other hand, we know that the value of the penalty parameter gives more or less weight to the objective function $F + G$ compared to the norm of the con-

sensus constraints violation $\|\mathbf{M}\mathbf{x} + \mathbf{P}\mathbf{y} - \mathbf{b}\|^2$ in the augmented Lagrangian, which creates the imbalance between the primal and residual values. Thus, it makes sense to use the relative imbalance of the residuals and adapt the penalty value accordingly. The general scheme reads:

$$\lambda_{k+1} = \begin{cases} \beta\lambda_k & \text{if } r^k > \mu s^k \\ \lambda_k/\beta & \text{if } s^k > \mu r^k \\ \lambda_k & \text{otherwise ,} \end{cases} \tag{4.3}$$

where $\beta, \mu > 1$.

In many applications, the common practice is to set $\beta = 2$ and $\mu = 10$. We keep this setting in all our numerical experiments when residual balancing is used.

4.1.2 Penalty tuning in FD-ADMM

Problem structures

For appropriate problems, it is possible to use a result shown in [48] to compute an optimal penalty parameter. It will first help us in tuning C-ADMM to optimize its convergence. Let us first remind the definitions of strong convexity and Lipschitz continuity.

DEFINITION 4
We recall that a differentiable function $f : \mathbf{R}^n \to \bar{\mathbf{R}}$ is *strongly convex* with modulus σ if:

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

Moreover, the function f is *Lipschitz* with modulus L if:

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

In order to apply Corollary 2, we express the coefficients of interest σ, L_d.

Now, let $w > 0, \alpha > 0$, and let $g : x \in \mathbf{R}_{++} \to g(x) = -w\frac{x^{1-\alpha}}{1-\alpha}$ if $\alpha \neq 1, -w\log(x)$ if $\alpha = 1$. In all cases, we have:

$$\nabla g(x) = -\frac{w}{x^\alpha}.$$

Let us assume $\alpha > 1$. Therefore, for $0 < y < x \leqslant t$:

$$\begin{aligned}
(\nabla g(x) - \nabla g(y))(x-y) &= w(\frac{1}{y^\alpha} - \frac{1}{x^\alpha})(x-y) \\
&= \frac{w}{x^\alpha y^\alpha}(x^\alpha - y^\alpha)(x-y) \\
&= \frac{w}{x^\alpha y^\alpha}\alpha c^{\alpha-1}(x-y)^2 \\
&\qquad (y < c < x) \\
&= \frac{\alpha w}{x^\alpha y}\left(\frac{c}{y}\right)^{\alpha-1}|x-y|^2 \\
&\geqslant \frac{\alpha w}{x^\alpha y}|x-y|^2 \\
&\geqslant \frac{\alpha w}{t^{\alpha+1}}|x-y|^2,
\end{aligned} \tag{4.4}$$

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 case $\alpha = 1$ is straightforward.

Similarly, take $0 < s < y < x$, we have:

$$\begin{aligned}
|\nabla g(x) - \nabla g(y)| &= w|\frac{1}{y^\alpha} - \frac{1}{x^\alpha}| \\
&= \frac{w}{x^\alpha y^\alpha}|x^\alpha - y^\alpha| \\
&= \frac{w}{x^\alpha y^\alpha}\alpha c^{\alpha-1}|x-y| \\
&\qquad (y < c < x) \\
&\leqslant \frac{\alpha w}{s^{\alpha+1}}|x-y|,
\end{aligned} \tag{4.5}$$

where the last line is obtained in the same fashion as in (4.4), for each case $\alpha < 1$, $\alpha > 1$. The case $\alpha = 1$ is straightforward. We can therefore state the following.

FACT 1
Recall the expression of $g_r(x) = -w_r \frac{x^{1-\alpha}}{1-\alpha}$ for $\alpha \neq 1$ or $-w_r \log(x)$ for $\alpha = 1$, the weighted α-fair (differentiable) convex function of one variable of domain within $[s, t]$, where $0 < s < t$. Then, g_r is σ_r-strongly convex, and ∇g_r is L_r-Lipschitz, with $\sigma_r = \frac{w\alpha}{t^{\alpha+1}}$, and $L_r = \frac{w\alpha}{s^{\alpha+1}}$.

Corollary: The α-fair (convex) function g is σ-strongly convex and has L_d-Lipschitz gradient with:

$$\sigma = \alpha \min_r \frac{w_r}{u_r^{\alpha+1}}, \quad L_d = \alpha \max_r \frac{w_r}{d_r^{\alpha+1}} \tag{4.6}$$

on any compact subset of $\mathrm{dom}(g)$ of the form $K_d = \{\mathbf{x} \geqslant \mathbf{d}, \mathbf{A}\mathbf{x} \leqslant \mathbf{c}\}$, for $\mathbf{d} > 0$ (component-wise), and $u_r = \min_{j \in r} c_j$ (the utopia allocation introduced in Chapter 2).

Optimal penalty for C-ADMM

Inspired by Theorem 3, we can now derive our optimal penalty parameter for C-ADMM. Let us recall again the centralized formulation:

$$\begin{aligned} \min \ & g(\mathbf{x}) + \iota(\mathbf{y}) \\ \text{s.t. } & \mathbf{x} - \mathbf{y} = 0, \end{aligned} \tag{4.7}$$

where ι is the indicator function of the feasible (convex) set, and is thus convex.

We just showed in Fact 1 that g is σ-strongly convex and has an L_d-Lipschitz gradient given by formulas (4.6) on a compact set of the form K_d. The point $\mathbf{d} > 0$ has to be chosen so that K_d is non-empty, and contains the optimal solution $\mathbf{x}^*$. Therefore, to have a global Lipschitz factor for g, one must reduce the feasible set to K_d where $\mathbf{d}$ is a *lower bound* on the fair allocation. Let us redefine ι to be the indicator function of the set K_d. Then, we know that the modified problem (4.7) responds to the conditions of Theorem 3. All in all, we have the:

THEOREM 4
With a chosen penalty parameter $\lambda > 0$, the iterates of C-ADMM converge linearly to the optimal solution at rate $(1+\delta)^{-1}$, where:

$$\delta = 2\left(\frac{1}{\lambda\sigma} + L_d\lambda\right)^{-1}. \tag{4.8}$$

We thus have the optimal penalty parameter:

COROLLARY 3
The optimal penalty parameter for C-ADMM is:

$$\lambda_* = \sqrt{\frac{1}{\sigma L}} = \frac{1}{\alpha}\sqrt{\frac{1}{\min_r \frac{w_r}{u_r^{\alpha+1}} \max_r \frac{w_r}{d_r^{\alpha+1}}}}, \tag{4.9}$$

where $\mathbf{u}$ is the utopia point of the problem, and $\mathbf{d}$ is a lower bound on the fair allocation problem, both introduced or derived in Chapter 2.

Adaptive penalty for FD-ADMM

We now want to propose a penalty parameter for our distributed algorithm FD-ADMM. We were able to circumvent the difficulty of the unboundedness of the α-fair functions near the axis, by pruning the feasible set away from them. In this way, we equipped the fair functions with a finite global Lipschitz factor and were able to verify the assumptions of Theorem 3 which gave us a parameter with theoretical best worst cast linear convergence rate. Here, the structure of FD-ADMM is different. In the design of the algorithm, we plugged the separated link capacity constraints into the objective (with their indicator function) and landed back on a general form of ADMM (4.1) with $\mathbf{M} = -\mathbf{P} = \mathbf{I}$ by integrating again the consensus constraints of the type $z_{jr} = x_r$ into the objective. Specifically, in $(\mathcal{A}_0)$, the structure of FD-ADMM is as follows:

- $F(\mathbf{x}, \mathbf{z}) = g(\mathbf{x}) + \sum \iota_j(\mathbf{z}_j)$,
- $G(\mathbf{x}', \mathbf{z}')$ is the indicator function of the consensus set defined by $z_{jr} = x_r$ for all r and all $j \in \mathcal{J}_r$,
- $\mathbf{M} = -\mathbf{P} = \mathbf{I}$, and $\mathbf{b} = \mathbf{0}$.

Thus, in FD-ADMM, the functions F and G are both convex, and the matrix $\mathbf{M}$ is invertible. However, it is clear that the function F is no longer strongly convex with respect to the joint variable $(\mathbf{x}, \mathbf{z})$, due to the concatenation of the indicator functions ι_j. Thus, the results of Theorem 3 no longer apply. In fact, theoretically, we no longer have a worst case convergence rate guarantee of FD-ADMM and as a consequence cannot provide a correct penalty tuning. We consider two solutions for this:

1. **Fixed parameter tuning:** use the optimal parameter from C-ADMM for FD-ADMM with the same lower bound.
2. **Adaptive parameter tuning:** use the adaptive scheme presented in Section 4.1.1.

4.2 Illustration

4.2.1 Objective

We now illustrate the results obtained in the previous sections. We wish to evaluate in practice, the efficiency of the two possible penalty parameter tuning strategies for FD-ADMM, namely, the adaptive penalty with residual balancing on-the-fly, and the fixed parameter that is given by a combination of Theorem 3 and our results in Chapter 2. The residual balancing strategy is certainly the most tempting when one does not know how to tune the penalty. Indeed, it is well-known, and we will observe this in the simulations, that residual balancing makes the performance of the algorithm less dependent of the initialization of the parameter value. The guarantees of convergence for residual balancing ADMM are set when the parameter is fixed *ultimately*. However, it is not clear when to stop the residual balancing and to which value to fix ultimately the penalty parameter. An acceptable criterion is to assume that the residual balancing strategy should stop when the residual values are small enough – a small residual value is a certificate for a small optimality gap; therefore, we can stop the residual balancing when we are close enough to the optimum and continue with a fixed penalty thereafter. On the other hand, it is interesting to see, instead of balancing the residuals, what would have happened if we had a fixed parameter from the beginning. In other worlds, we evaluate the gains of a fixed conditioning of λ against the ones of a generic adaptation scheme like residual balancing. We therefore compare the achieved gaps when the residual falls below 10^{-3}, for both fixed and residual balancing strategies. We plot the number of achieved iterations to reach such a precision, and the achieved optimality gap at the end of the experiment. We run both algorithms for different values of the initial penalty parameter λ_0, to account for the sensitivity of the performance with respect to initialization. We take as a reference λ^*, the penalty value obtained with Theorem 3 and Proposition 1 (Chapter 2). Of course, each time the results of Proposition 1 were used, we verified whether on the particular instance, the lower bound was correct. Among all our instances, we found no counter example.

4.2.2 Simulation setting

The aim of this experiment is to analyze the behavior of FD-ADMM under different tuning strategies. To observe the structure of the effect of the penalty initialization/update, it is not crucial to run FD-ADMM on large instances. This is why we limit ourselves to small synthetic instances. All our instances are generated following the Barabasi-Albert model. We thus used networks consisting of 10 nodes built following the model with a general connection factor of 3. This gave networks with approximately 30–40 links on which we generated 50 paths formed as the shortest path between randomly chosen couple of source/destination nodes. All the capacities c_j

and the weights w_r were fixed to a value within the interval $1 \pm 20\%$ drawn uniformly at random. All in all, we generated 20 such instances.

We run FD-ADMM separately on those instances, with a fixed parameter, and using residual balancing. In both cases, we analyze numerically the effect of the initialization. Therefore, with λ^* as a reference, we plot the results (iterations to achieve a residual tolerance of 10^{-3}, and achieved optimality gap) against the initial factor $\rho = \lambda_0/\lambda^*$, so that the initial penalty is $\rho\lambda^*$. The runs of FD-ADMM stop whenever the desired tolerance level is achieved, or when a time limit of 400 iterations is reached.

4.2.3 Results

The results of our experiment for FD-ADMM are shown in Figure 4.1.

Fixed parameter

From the simulation results, we observe that the performance of FD-ADMM is indeed very dependent of the penalty parameter when it is initialized to a fixed value. In terms of iteration count, we reach the desired level of tolerance the fastest when the initial factor λ_0/λ_* is close to 1, for $\alpha = 1, 2$, and 3, in average. For the specifics, the "optimal" parameter is within the same order of magnitude as λ_*. Otherwise, when the initial factor is too large or too small, we observe a performance deterioration, as the number of required iterations to reach the same level of tolerance explodes quickly (below 0.1 and over 10, the iteration count is already more than twice the one for the best observed performance). Although for all values of α, there is clearly an optimal penalty parameter that provides a same guarantee of the residual tolerance, we can observe that the case of proportional fairness is more permissive of small errors in the initialization. As α grows, it seems that a fixed penalty should be chosen more accurately to avoid performance degradation.

In terms of optimality gap, we can interestingly observe that the actual quality of the solution (which, in passing, cannot be evaluated in real scenarios other than via an estimator based on the residual value – see the discussion in Chapter 3) is also at its best when the initial parameter is chosen around λ^*. The best optimality gaps are, in all three values of α, reached in average and in the specific instances for initial factors around but smaller than 1. On the contrary, we observe that the fewest iterations are achieved for values of the initial factor close to but *greater* than 1 in most specific cases. Interestingly, an initial factor of 1 seems like a good trade-off between iteration count and accuracy.

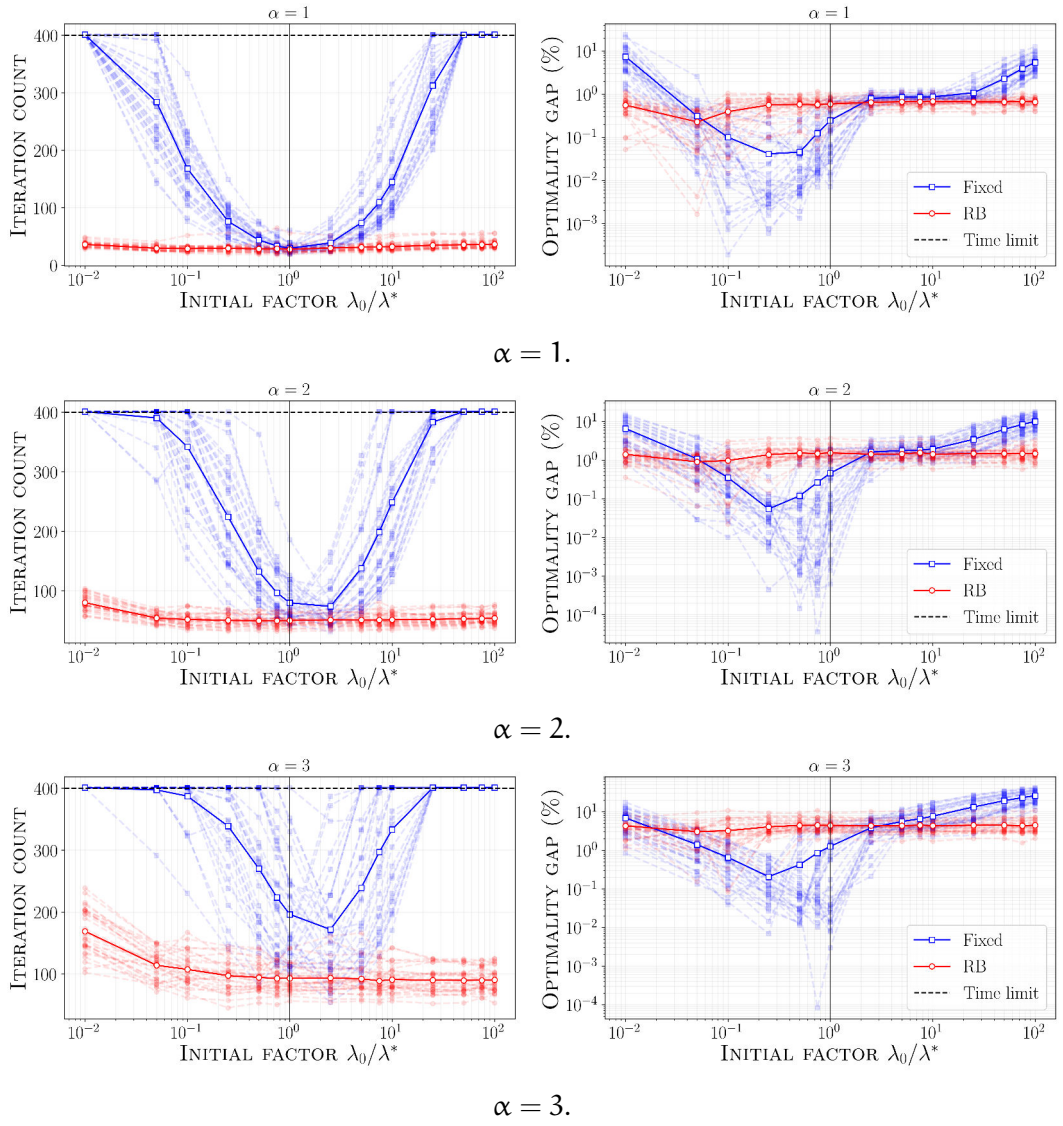


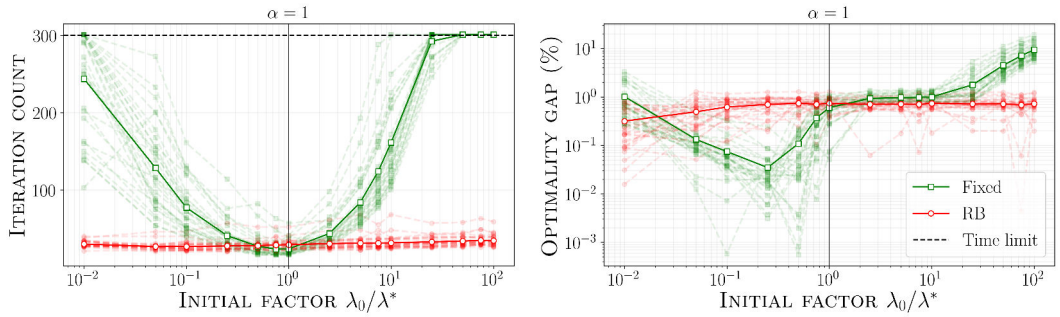
Figure 4.1 | The number of iterations required for FD-ADMM to reach a residual tolerance of 10^{-3} (left panel) and the achieved optimality gap (right panel), versus the initial factor λ_0/λ^* for $\alpha = 1, 2, 3$.

Residual balancing

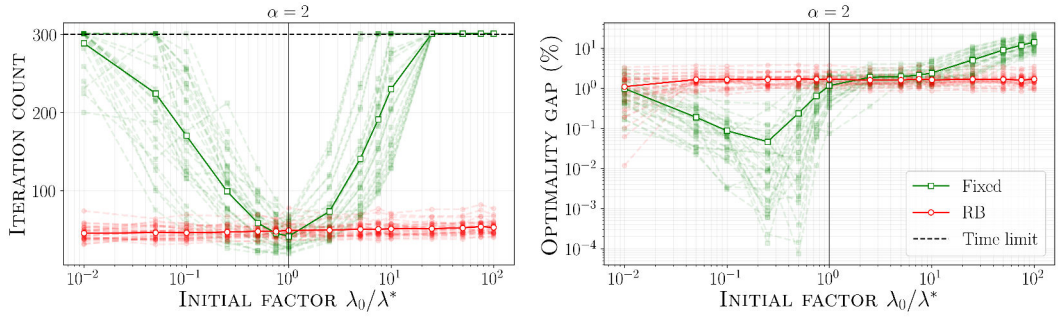
The results show without any doubt that the residual balancing strategy permits to make FD-ADMM essentially independent from the initial factor. Although we did not plot the points, we observe that for an initial factor below 10^{-2}, the iteration count also explodes, which means that the independence occurs to a certain extent. All in all, the curves for RB clearly show that to reach the same level of residual tolerance (which, again, gives a practical guarantee on the optimality gap, up to a certain factor), it is enough to chose an initial penalty parameter that is "large enough". For instance, an acceptable lower bound on the initial penalty parameter is clearly λ^*.

As for the optimality gap, we observe the same independence property.

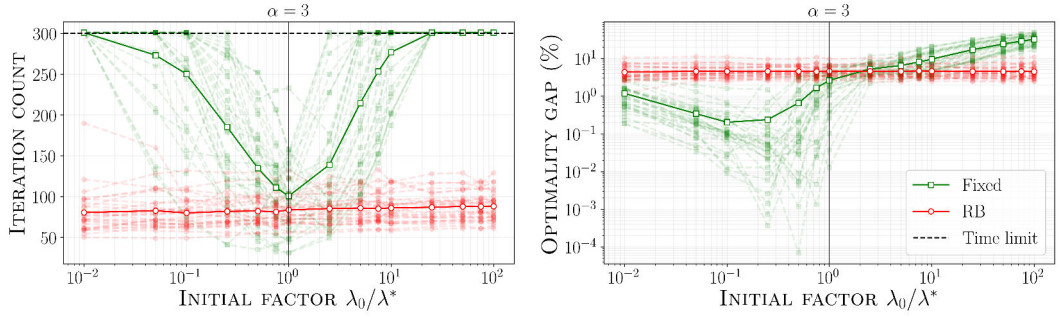
Although the independence of the optimality gap with regard to the penalty parameter initialization is very desirable, we cannot ommit the fact that the achieved optimality gap, although acceptable, is not as good as the best performance for the fixed parameter run. This illustrates the fact that a residual balancing scheme needs (theoretically) to be followed by a fixed penalty run to converge. Notwithstanding, we can observe that the average, best case, and worst case optimality gaps are always better for the fixed run with penalty λ^*.



$\alpha = 1$.



$\alpha = 2$.



$\alpha = 3$.

Figure 4.2 | The number of iterations required for C-ADMM to reach a residual tolerance of 10^{-3} (left panel) and the achieved optimality gap (right panel), versus the initial factor λ_0/λ^* for $\alpha = 1, 2, 3$.

Validation for C-ADMM

Finally, we conducted the same experiment for C-ADMM and observed essentially the same properties as for FD-ADMM. The aim of this last exper-

iment was to validate our approach of combining Theorem 3 and Proposition 1 (Chapter 2) to derive the penalty parameter λ^*. Surprisingly, the improved lower bound conjectured in Chapter 2 permits to derive a numerically optimal parameter in average, as the residuals seem to converge the fastest to 0 at this setting. Also, in terms of optimality gap, we observe that the best gaps are obtained for fixed values of λ^0/λ^* between 0.01 and 1. In all cases, these gaps are considerably (especially for the case $\alpha = 3$) better than the ones for RB, although when the initial factor is set to 1, both schemes have a comparable performance.

To summarize the first part of this chapter, we have shown that the iterates of C-ADMM converge linearly to the unique optimal point of the problem, and we derived a theoretical value of the penalty parameter to use when running the centralized algorithm. This value depends on a lower and an upper bound on the optimally fair allocation. Although it is easy to derive an upper bound, as the utopia point is one, deriving a lower bound is not trivial and was the purpose of the discussion in Chapter 2. We used the proposed lower bound in Proposition 1 (after checking that it was correct on each used instance) to derive what we called the *optimal penalty parameter*. This parameter was proposed as initialization of the parameter to run our algorithm FD-ADMM, that does not a priori detain the same convergence speed and the same dependednce on λ. However, we tested its performance and showed by simulation that the optimal penalty parameter was a very good candidate. To validate this property, we compared the performance of FD-ADMM with fixed parameter against the classic residual balancing strategy and showed that it was possible do obtain comparable performance.

In our distributed setting, the major advantage of having a fixed penalty parameter is that an update requires to communicate residual values between the controllers in order to decide whether the penalty parameter has to change or not. This adds more communication overhead whereas with a fixed penalty parameter, only the consensus point $\tilde{z}$ needs to be built (and the feasible point $z^\dagger$ when the algorithm decides to stop at the end of the run).

We therefore argue that knowing that there is a closed form it might be more profitable to use ADMM with a fixed penalty parameter. This is definitely the case for FD-ADMM. However, in other applications, it is of course always better to use residual balancing when no optimal penalty parameter can be derived.

4.3 Practical extensions of the model

In this section, we present some extensions of the fair resource allocation model. We show how they can be easily adapted in FD-ADMM and present the corresponding modified update rules.

4.3.1 Multi-path extension

The first natural feature that one can extend the α-fair model with is the multi-path setting. In this situation, each connection request $r \in \mathcal{R}$ has a set of candidate paths P_r. In all generality, no assumption is made on the link-disjointness of the paths for the same request, but this requirement appears as an important constraint to empower the model against link/node failures. As in the rest of this thesis, we assume all the paths are pre-computed and fixed once and for all. Thus, we have the following terminology:

1. The *path-wise bandwidth allocation* for a request r over a path $p \in P_r$ is denoted x_p;
2. The *aggregate bandwidth allocation* of a request r over the totality of its paths is denoted y_r.

Notations

The multi-path extension requires the extensions of the definitions and notations established in the last chapter. The set $\mathcal{J}$ still denotes the set of the network links. The set $\mathcal{R}$ now denotes the set of *connection requests*, each linking a source node to a destination node through *one or many* pre-established paths. For each request r, let P_r denote the set of its paths. We denote $\mathcal{P} \coloneqq \sqcup_{r\in\mathcal{R}} P_r$ the disjoint union of the P_r. Therefore, for each path p, there is an unique $r = r(p)$ such that $p \in P_r$. For each path p, $\mathcal{J}_p$ is the set of links forming it.

Therefore, the capacity constraint matrix $\mathbf{A}$ is now with shape $|\mathcal{J}| \times |\mathcal{P}|$ and has the definition:

$$A_{jp} = \begin{cases} 1 & \text{if } j \in \mathcal{J}_p \\ 0 & \text{otherwise,} \end{cases} \tag{4.10}$$

For an SDN domain m, the set of links that belong to the domain is denoted $\mathcal{J}(m)$, and the set of paths p it contains is $\mathcal{P}(m) = \{p \in \mathcal{P}; \exists j \in J_m, j \in \mathcal{J}_p\}$. Note that a domain may contain only some but not all of the paths of a certain request. Conversely, I_p is the set of domains that the path p traverses, *i.e.*, $p \in \mathcal{P}(m) \iff m \in I_p$. Moreover, we assume we have a partition of the set $\mathcal{R}$ into M subsets $(\mathcal{R}(m))_{m=1\ldots M}$, where each request is assigned to a unique domain. For instance, $\mathcal{R}(m)$ can be the set of requests r that take their source or destination within the domain m.

The centralized algorithm

The allocation fairness is measured over the vector $\mathbf{y}$: the resource allocation policy should be fair with respect to the aggregate bandwidth allocation. Therefore, we introduce the *request-path* incidence matrix $\mathbf{B}$ as follows:

$$B_{r,p} \coloneqq \begin{cases} 1 & \text{if } p \in P_r \\ 0 & \text{otherwise,} \end{cases} \tag{4.11}$$

and the flow conservation constraint:

$$\mathbf{B}\boldsymbol{x} = \boldsymbol{y}. \tag{4.12}$$

Consequently, the multi-path weighted $(\boldsymbol{w}, \alpha)$ fair resource allocation problem can be cast as the following modified optimization problem:

$$\begin{aligned} \min\ & g(\boldsymbol{y}) \\ \text{s.t. } & \boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{c}, \\ & \mathbf{B}\boldsymbol{x} = \boldsymbol{y}. \end{aligned} \tag{4.13}$$

Introducing again the link capacity constraints into the objective via their indicator function, we obtain the modified version of the centralized formulation for C-ADMM:

$$\begin{aligned} \min\ & g(\boldsymbol{y}) + \iota(\boldsymbol{x}) \\ \text{s.t. } & \boldsymbol{y} - \mathbf{B}\boldsymbol{x} = \mathbf{0}. \end{aligned} \tag{4.14}$$

Now, solving the problem formulated by (4.14) with ADMM gives essentially the same update rules than C-ADMM, except that the central projection in C-ADMM (see Algorithm 1 line 3, page 42) is slightly modified due to the presence of the matrix $\mathbf{B}$ in the equality constraints. Nonetheless, the two algorithms have a similar structure.

FACT 2
The modified version of C-ADMM adapted to the multi-path formulation (4.14) has the same convergence guarantees as C-ADMM.

Proof. Indeed, we did not step out of the conditions of the convergence theorem 3. The function g is strongly convex with respect to the variable $\boldsymbol{y}$, and a lower bound on the optimal aggregate bandwidth allocation $\boldsymbol{y}^*$ can be obtained with the same techniques as in Chapter 2, which gives a bounded problem on a reduced set and a global Lipschitz constant of ∇g. Moreover, the constraint matrix applied to the variable $\boldsymbol{y}$ is the identity matrix – hence, the full row rankness condition holds. □

We need to remark immediately that the lower bound is no longer pathwise. Indeed, the fairness is measured on the aggregate bandwidth allocations $\boldsymbol{y}$. Thus, a lower bound for the multi-path fair allocation problem is a vector $\mathbf{b} > \mathbf{0}$ such that there exists $\mathbf{d} \geqslant \mathbf{0}$ verifying the three conditions $\mathbf{B}\mathbf{d} = \mathbf{b}$, $\boldsymbol{A}\mathbf{d} \leqslant \boldsymbol{c}$, and $\mathbf{b} \leqslant \boldsymbol{y}^*$. In other words, we call lower bound, an aggregate bandwidth allocation corresponding to some feasible path-wise

bandwidth allocation and that is smaller (component-wise) than the optimal aggregate bandwidth allocation.

Also, another important remark is that, although for strict (and in fact, strong) convexity reasons, the optimal aggregate bandwidth allocation $\mathbf{y}^*$ is always unique, it is no longer the case for the path-wise bandwidth allocation, which is thus not uniquely defined[2]. It is possible to modify the problem in order to make the optimal path-wise allocation $\mathbf{x}^*$ unique; instead of constraining $\mathbf{B}\mathbf{x} = \mathbf{y}$, one can introduce, for $q \in (0,1)$, the modified convex constraint $\mathbf{B}\mathbf{x}^q = \mathbf{y}^q$ (where the exponentiation is defined component-wise as $\mathbf{x}^q \coloneqq (x_p^q)_{p\in\mathcal{P}}$, and likewise for $\mathbf{y}$), which renders the function $\mathbf{x} \to g((B\mathbf{x}^q)^{1/q})$ strongly convex in $\mathbf{x}$ and therefore eliminates the variable $\mathbf{y}$ while still preserving the structure assumption of Theorem 3. This trick can be seen in [22]. But in our case, we remarked that this made the proximal operator of the function $g \circ \mathbf{B}$ more difficult to evaluate. Therefore, we adapted FD-ADMM to the multi-path problem while keeping the path-wise bandwidth allocation non-unique.

FD-ADMM for multi-path resource allocation

The decomposition procedure takes a similar form than the one in Chapter 3. The multi-path distributed formulation reads:

$$\begin{aligned} \min\ & H(\mathbf{x},\mathbf{z}) + \chi(\mathbf{x}',\mathbf{z}') \\ \text{s.t.}\ & \begin{pmatrix}\mathbf{x}\\ \mathbf{z}\end{pmatrix} - \begin{pmatrix}\mathbf{x}'\\ \mathbf{z}'\end{pmatrix} = \begin{pmatrix}\mathbf{0}\\ \mathbf{0}\end{pmatrix}, \end{aligned} \tag{4.15}$$

with the new definitions:

$$\begin{aligned} H(\mathbf{x},\mathbf{z}) &= \sum_{r\in\mathcal{R}} g_r\Big(\sum_{p\in\mathcal{P}_r} x_p\Big) + \sum_{j\in\mathcal{J}} \iota_j(z_j) \\ \chi(\mathbf{x}',\mathbf{z}') &= \begin{cases} 1 & \text{if } x_p = z_{j,p} \forall p \in \mathcal{P}, \forall j \in \mathcal{J}_p \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{4.16}$$

This formulation yields the version of FD-ADMM made for multi-path α-fair resource allocation that is summarized through the update rules of Algorithm 5.

In the following discussion, we display the extended results of Chapter 2 on the lower bounds for the multi-path setting of the fair resource allocation.

Aggregate bandwidth lower bounds

We now wish to extend the results of Chapter 2 to the multi-path fair resource allocation extension. As we remarked earlier, the extension of the

[2]Simply, when the number of paths is greater than the number of requests, the matrix $\mathbf{B}$ cannot have a null kernel for obvious rank reasons.

Algorithm 5 FD-ADMM (multi-path)

procedure OF DOMAIN m
  RECEIVE $z_{qp} \quad \forall q \in I_p \quad \forall p \in \mathcal{P}(m)$
  $\tilde{z}_p \leftarrow \frac{1}{I_p+1}\left(\sum_{q\in I_p} z_{qp} + x_p\right) \quad \forall p \in \mathcal{P}(m)$
  for $j \in \mathcal{J}(m)$ **do**
    $u_p^j \leftarrow u_p^j + \frac{1}{\lambda}(z_p^j - \tilde{z}_p) \quad \forall p, j \in \mathcal{J}_p$
    $z^j \leftarrow \text{LINK\_PROJ}(j, \tilde{z} - \lambda u^j)$
  end for
  for $r \in \mathcal{R}(m)$ **do**
    $\nu_p \leftarrow \nu_p + \frac{1}{\lambda}(x_p - \tilde{z}_p) \quad \forall p \in P_r$
    $\boldsymbol{x} \leftarrow \text{prox}_{\lambda g_r}(\tilde{\boldsymbol{z}} - \lambda \boldsymbol{\nu})$
  end for
  SEND $z_{mp} = x_p + \sum_{j\in\mathcal{J}_p\cap\mathcal{J}(m)} z_{jp}$ to domains $q \in I_p \quad \forall p \in \mathcal{P}(m)$
end procedure

results of Chapter 2 yields a lower bound on the aggregate bandwidth allocation $\mathbf{y}^*$, and not the path-wise allocation $\mathbf{x}^*$, which in passing, is not uniquely defined. In order to do so, we redefine the utopia point[3] as follows:

$$\forall r \in \mathcal{R} \quad \mathbf{u}_r \coloneqq \max_{y=\mathbf{B}x, y_{s\neq r}=0, \mathbf{A}x\leqslant \mathbf{c}} y_r. \tag{4.17}$$

Also, for each request $r \in \mathcal{R}$, the restricted problem includes all the requests $s \in \mathcal{R}$ that have some path $q \in P_s$ sharing a link with some path $p \in P_r$:

$$\mathcal{R}^r \coloneqq \{s \in \mathcal{R}; \exists q \in P_s, \exists p \in P_r, \mathcal{J}_q \cap \mathcal{J}_p \neq \varnothing\}. \tag{4.18}$$

Lastly, the local midpoint $\mathbf{p}$ is defined likewise on the new restricted sets $\mathcal{R}^r$:

$$\forall r \in \mathcal{R}, p_r \coloneqq \frac{w_r}{\sum_{s\in\mathcal{R}^r} w_r} u_r. \tag{4.19}$$

First, one can get easily an extension of the formulas in Theorem 2 (Chapter 2):

[3]The notion itself is unchanged: the utopia allocation of a request is the aggregate bandwidth it would receive along the total set of its paths if they were alone in the network.

THEOREM 5
Let $\mathbf{y}^*$ denote the optimal aggregate allocation of the multi-path problem. Let $r_0 \coloneqq \operatorname{argmin}_{s\in\mathcal{R}} p_s$. Then, $\mathbf{x}^*$ can be lower bounded as follows:

- if $\alpha \geqslant 1$, $\quad \forall r \in \mathcal{R} \quad y_r^* \geqslant d_r(\alpha) \coloneqq p_{r_0}^{1-1/\alpha} p_r^{1/\alpha}$
- if $0 < \alpha \leqslant 1$, $\quad \forall r \in \mathcal{R} \quad y_r^* \geqslant d_r(\alpha) \coloneqq \left(\dfrac{w_r u_r}{\sum_{s\in\mathcal{R}^r} w_s u_s^{1-\alpha}} \right)^{1/\alpha}$.

Now, we have the aggregate version of Lemma 2 which can be obtained likewise, and in the same vein as for the uni-path model, we thus have the series of lower bounds that permit one to compute a more tighter lower bound on the restricted problems:

COROLLARY 4
Let $\alpha \geqslant 1$. Let $\mathbf{a}(0)$ be a lower bound and the sequence $\mathbf{a}(n)$ be defined by the formula:

$$\forall r \in \mathcal{R} \quad a_r(n+1) \coloneqq \left(\frac{w_r u_r}{\sum_{s\in\mathcal{R}^r} w_s a_s(n)^{1-\alpha}} \right)^{1/\alpha}. \tag{4.20}$$

Then, $(\mathbf{a}(n))_n$ is a sequence of lower bounds for the α-fair resource allocation problem for $\alpha \geqslant 1$.

In all generality, we have the following formula that can be obtained just like Proposition 1.

PROPOSITION 3
Let $\alpha \geqslant 1$. Let $\mathbf{y}^*$ denote the optimal aggregate allocation of the problem. Then, $\mathbf{y}^*$ can be lower bounded as follows:

$$\forall r \in \mathcal{R}, \quad y_r^* \geqslant \frac{w_r^{1/\alpha} u_r^{1/\alpha}}{\sum_{s\in\mathcal{R}} w_s^{1/\alpha} u_s^{1/\alpha-1}}. \tag{4.21}$$

The possible improvement of Proposition 3 to an equivalent form of

Proposition 1 (Chapter 2) is not straightforward due to the lack of path-wise sensitivity results in the multi-path setting, the non-unicity of the path-wise allocation being the major difficulty here. In practice, we chose the limit of the series $(a(n))_n$ as a lower bound to define our penalty parameters.

4.3.2 Imposing sparsity patterns

In this section, we show how FD-ADMM can be adapted to address the relevant problem of limiting flow reconfiguration. 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 overwhelming flow reconfiguration rules that can cause Quality-of-Service degradation or system instability [59]. Therefore, we use ideas of sparse optimization 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. The presentation of the (rich) theory on sparse optimization goes beyond the scope of this thesis. Hence, we limit ourselves to the adaptation to our setting. The reader is referred to [60] for an extensive theory. For simplicity, we switch back to the uni-path model of the fair resource allocation problem. Consequently, we discard the notations of the last section and switch back to the ones in Chapter 3.

Sparsity-inducing regularization

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 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 objective, and thus satisfy the demands with a better fairness measure as they evolve. However, the number of permitted flow reconfiguration may be limited. Therefore, we introduce a switching cost to limit the number of reconfiguration. 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 x^0 be a feasible path-wise allocation and assume the actual resource allocation of the demands follows x^0. Now, the traffic demands have changed and the network has to recompute a new (uni-path) allocation $x^\sharp$ 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:

$$\|\mathbf{x}^0 - \mathbf{x}^*\|_0 \leqslant \kappa, \iff \sum_p \mathbf{1}(x_p^0 \neq x_p^*) \leqslant \kappa, \tag{4.22}$$

where $\|u\|_0 = \text{Card}\{p, u_p \neq 0\}$ is called the *zero-norm*[4] of a vector usually denoted ℓ_0. Adding the constraint of Eq. (4.22) into the problem gives rise to a problem structure with integral constraints, and falls out of the scope of the classic ADMM. We consider here a relaxation of this problem that is still tractable with the method.

We can control the zero-norm (4.22) by adding the most natural sparsity inducing penalty induced by the Θ-scaled ℓ_1-norm $\Theta\|\mathbf{x}^0 - \mathbf{x}^*\|_1$, 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 $\{\boldsymbol{\nu} \text{ s.t. } \|\boldsymbol{\nu}\|_0 \leqslant 1\}$. We therefore consider Problem (3.32) (Chapter 3), with an extended expression of the function H:

$$H(\mathbf{x}, \boldsymbol{z}) = \sum_{r \in \mathcal{R}} g_r(x_r) + \sum_{j \in \mathcal{J}} \iota_j(\boldsymbol{z}_j) + \Theta \sum_{r \in \mathcal{R}} |x_r - x_r^0| \tag{4.23}$$

Change in FD-ADMM

With this extended formulation, the changes in FD-ADMM only occur in the proximity operator calculus. Indeed, we combine the regularization term with the fairness measure by designing a unique variable for the two terms. Therefore, FD-ADMM will operate the exact same update rules than in Algorithm 3 (Chapter 3), except that line 10 is replaced with:

$$x_r \leftarrow \text{prox}_{\lambda g_r + \lambda\Theta|\cdot - x_r^0|}(\tilde{z}_r - \lambda\nu_r). \tag{4.24}$$

The proximal calculus (4.24) is quite simple to execute. Indeed, let $\mathbf{u} = \boldsymbol{z} - \lambda\boldsymbol{\nu}$ the proximal point in (4.24) is the unique point $\mathbf{x}$ verifying:

$$\forall r \in \mathcal{R}, \quad x_r = \underset{x}{\text{argmin}}\, \beta(x) \coloneqq g_r(x) + \Theta|x - x_r^0| + \frac{1}{2\lambda}\|x_r - u_r\|^2. \tag{4.25}$$

As usual, the function β to minimize is strictly convex and coercive, therefore admits a unique solution. The only trick is that it is not differentiable at x_r^0. Therefore, we can check:

- if $x_r > x_r^0$, then x_r is a critical point, and verifies the first order condition, after simplification:

$$x_r^{\alpha+1} + (\Theta\lambda - u_r)x_r^\alpha - \lambda w_r = 0. \tag{4.26}$$

[4] This is an abuse of terminology as it is not a norm.

- if $x_r < x_r^0$, then x_r is a critical point, and verifies the first order condition, after simplification:

$$x_r^{\alpha+1} - (\Theta\lambda + u_r)x_r^{\alpha} - \lambda w_r = 0. \tag{4.27}$$

- else, $x_r = x_r^0$.

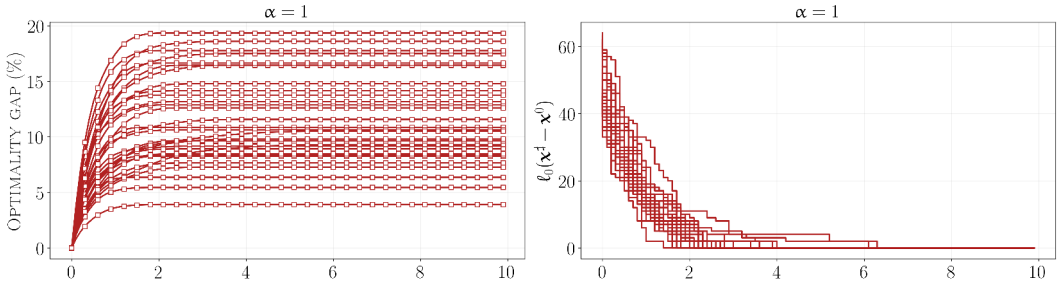
Thus, in our cases $\alpha = 1, 2$ or 3, we still have to find a positive root of a polynomial of degree $\alpha + 1$. Given the form of the polynomial, we have already observed that it admits one and only one positive root. Also, by convexity, we know that $x \mapsto \sup\{\partial\beta(x)\}$ is non-decreasing. Moreover, the point x_r is a critical point if and only if $0 \in \partial\beta(x_r)$. It is thus easy to check where x_r compared to x_r^0 by a comparison of the left and right derivatives of β at x_r^0:

- if the left and right derivatives have opposite signs, then the proximal point is x_r^0.;

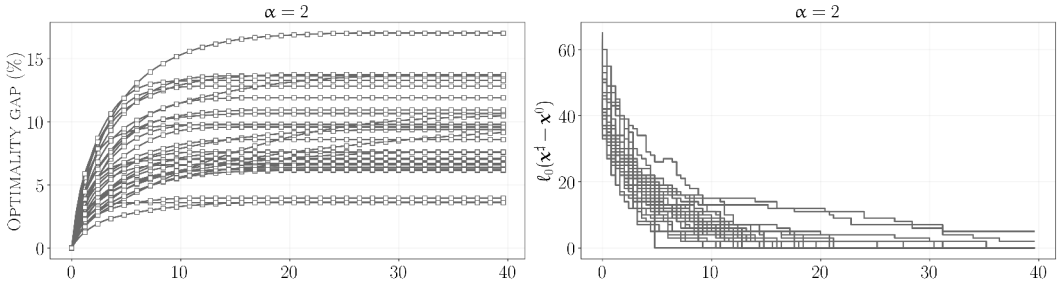
- if the left and right derivatives are both non-negative, then $x_r \leqslant x_r^0$ and verifies (4.27);

- if the left and right derivatives are both non-positive, then $x_r \geqslant x_r^0$ and verifies (4.26);

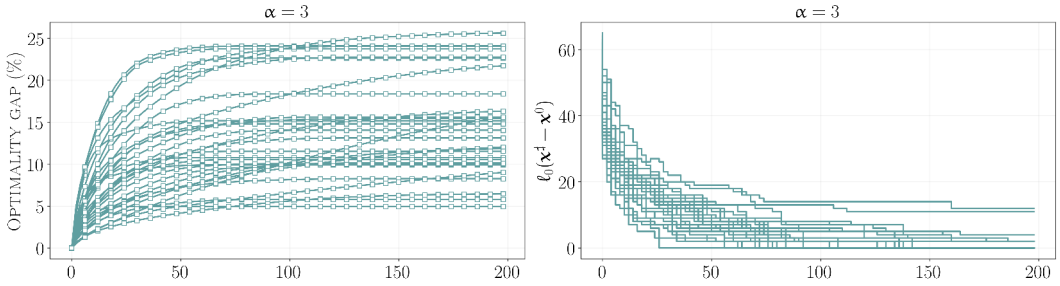
Thus, the term-wise minimization of the function $H(\cdot, z)$ now also takes into account an incentive to stay near the point $\mathbf{x}^0$. Of course, a proper tuning of the parameter Θ is necessary to enforce the real budget κ. The larger Θ, the smaller the number of re-sized paths. We will show this effect in the next paragraph, dedicated to the experimentation.



$\alpha = 1$.



$\alpha = 2$.



$\alpha = 3$.

Figure 4.3 | The achieved optimality gap (left panel) of the sparse solution $\mathbf{x}^\sharp$ (with reference the optimum with the new weights $\mathbf{w}^\sharp$), and its ℓ_0-difference with the initial allocation $\mathbf{x}^0$ versus the regularization term Θ (right panel), for $\alpha = 1, 2, 3$.

Illustration on the examples

We illustrate the effect of the sparsity inducing ℓ_1-norm on the optimal value and solution $\mathbf{x}^\sharp$ on the instances that have been considered in the simulations of Section 4.2. The setting is as follows. The traffic is considered to be set at a value x^0, which is the optimal solution for the considered instance. Then we model a change in the traffic requirements by changing all the weights w_r. To do so, we simply draw uniformly a new value $w_r^\sharp$ for each $r \in \mathcal{R}$. Then, we compute the new optimally fair solution without switching cost ($\Theta = 0$). Taking this value as reference, for various values of Θ, we run our modified algorithm to find an allocation that trades off fairness and switching costs and yields $\mathbf{x}^\sharp$, and plot the corresponding optimality gap. Also, we show the number of reconfigured paths $\ell_0(\mathbf{x}^0 - \mathbf{x}^\sharp)$. The results of this experiment are shown in Figure 4.3.

From the results, it is clear that the regularization term Θ permits to achieve all the levels of sparsity of the vector $x^0 - x^s$harp. For no sparsity at all, $\Theta = 0$ permits the algorithm to change the solution without switching costs and we get the optimally fair solution for the new traffic setting. The algorithm is not sensitive to a change of w at all if Θ is large enough. Of course, the threshold value for Θ as from where the allocation will not move will depend on $w - w^\sharp$, and on the instance.

General extensions

Lastly, we would like to give a general and convenient way of adding specific sets of constraints/regularization terms to the problem. We have seen that the ℓ_1 regularization term was easy to add and gave rise to a tiny modification of the proximal update rule. This is due to the fact that it is very straightforward fo carry the proximal calculus of the function β. One can imagine that a generic regularization term of the form $\rho(x)$ may not result in such simplicity in the structure of the function β. Although the only requirement is that the function $g + \rho$ should still be convex, it might be easier to compute separately prox_g and prox_ρ rather than $\text{prox}_{g+\rho}$ (for example, think of a non-separable function ρ). This is why we give the general trick to feature complicating constraints/regularizations to the fair resource allocation problem:

- For a new regularization $x \mapsto \rho(x)$, create a new variable x^ρ.
- Redefine χ as the indicator function of the set

$$\{(x, x^\rho, z); z_{jr} = x_r = x^\rho_r \quad \forall r \in \mathcal{R} \forall j \in \mathcal{J}_r\}.$$

- The modified FD-ADMM thus reads:

$$\begin{aligned} \min \ & \sum_{r\in\mathcal{R}} g_r(x_r) + \rho(x^\rho) + \sum_{j\in\mathcal{J}} \iota_j(z_j) + \chi(x', x^{\rho\prime}, z') \\ \text{s.t.} \ & \begin{pmatrix} x \\ x^\rho \\ z \end{pmatrix} - \begin{pmatrix} x' \\ x^{\rho\prime} \\ z' \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \end{aligned} \tag{4.28}$$

4.4 Concluding remarks

This chapter was organized into two parts.

In a first part, we reviewed the major results around the convergence of ADMM and used the structure of the fair allocation problem in order to provide a satisfactory penalty tuning for our FD-ADMM. The strong convexity of the fairness functions provide C-ADMM with a linear convergence property with a factor that depends on its penalty parameter λ (Theorem 4). Therefore, this convergence rate can be optimized with respect to λ which gives an optimal λ^* that guarantees best the worst case convergence rate

value (Corollary 3). The theoretical optimality of λ^* was evaluated and validated by simulations (Figure 4.2). Unfortunately, our distributed version, FD-ADMM, does not have the properties of C-ADMM and we cannot theoretically provide an optimal penalty tuning. For this kind of problem, residual balancing is a very common trick to make the algorithm performance less dependent of it. However, convergence is guaranteed *only* if the penalty is ultimately fixed. Therefore, residual balancing is a technique that the algorithm can use in order to estimate an accurate penalty from the first iterates. We compared the first iterations of FD-ADMM with residual balancing, and without residual balancing for different initialization values of the penalty parameter λ. It turned out that the optimal penalty λ^* for C-ADMM provides very satisfactory results for FD-ADMM, and specifically, values slightly below λ^* gave more precise solutions than with residual balancing (Figure 4.1). The conclusion of these observations is that FD-ADMM works better in the fist iterations with a fixed penalty computed with the help of C-ADMM, than with residual balancing, that requires a fixed run at the end anyway.

In a second part, we showed that FD-ADMM was versatile enough to address requirements that could be specific to the concrete problem in hand. We presented first the natural extension of the fair resource allocation to the multi-path setting and gave the corresponding modified algorithm and the new update rules to conduct. We checked that the new centralized formulation had the same structure than C-ADMM and concluded that the centralized algorithm had the same convergence properties (Proposition 2). Also, in order to adapt FD-ADMM to this use case, we conducted the same decomposition method as in the one in Chapter 3, this time yielding a not strictly convex fair multi-variate function per request. We then extended the results of Chapter 2 that give a lower bound, not on the path-wise allocation, but the aggregate allocation of each request. In the same way, this bound permits to derive the multi-path penalty λ^*.

Lastly, we showed with the example of sparse regularization how FD-ADMM could be adapted to limit the number of resized paths in a real-time scenario. This feature might be of interest in networks where too much flow re-sizing may cause instability and quality degradation and where a flow reconfiguration step-by-step is more desirable. In the same vein, we reformulated the problem with the introduction of a ℓ_1 regularizer and showed the corresponding update rules. We implemented the changed in FD-ADMM and showed the effect of the regularization term Θ on the number of re-sized paths (Figure 4.3). The results showed that the value of Θ could enforce all the spectrum of desired sparsity. Of course, a precise quantification of the effects of Θ is another interesting technical question. More generally, we gave the global method to equip FD-ADMM with another set of complicating constraints on the variable $\mathbf{x}$ and/or another more complicated regularizer.

Therefore, we argue that our algorithm FD-ADMM is well equipped and conditioned to tackle the problem of fair resource allocations in real-time with time-varying traffic requirements. Indeed, the previous chapters, and

the present one, showed that the algorithm can provide feasible (uni-path or multi-path) solutions on demand (Chapter 3) while continuously approaching the optimum, and can choose which path to reconfigure with a simple ℓ_1-regularization that provides sparse structured modifications. FD-ADMM thus responds to the real-time fairness and optimizes resources globally, and the whole procedure can be massively distributed following any domain distribution of the Software-Defined Network.

Thus, fairness as a resource sharing policy was addressed on networks of capacitated links and we showed how to tackle it in real-time. FD-ADMM is fit to tweak the resource allocation in real-time while taking account of a reconfiguration budget not to trespass, when the traffic requirements vary on-the-fly. Another parameter that can vary in practical scenarios is the network link capacities. Until now, we always assumed the capacity vector $\mathbf{c}$ was fixed and known by the SDN controller. In reality, a fixed capacity vector is a bold simplification of the resource allocation challenge. Indeed, the network equipment is scaled and partitioned between many users (service providers) and this partitioning of the resources might be dynamically adjusted to best fit the different users' requirements. In this context, one can adapt the resource allocation problem by defining a varying capacity vector. This immediately leads us to define a stochastic version of our resource allocation problem that takes into consideration a potential uncertainty on the parameter $\mathbf{c}$. In this situation, the notion of feasibility is replaced with the one of *safety*, which is essentially a probabilistic notion of feasibility. The object of the next and last chapter of this thesis is therefore the safe and fair resource allocation in uncertain environments.

Chapter 5

Safe fair allocation under environment uncertainties

REGARDLESS OF ITS distributed structure, we considered so far in this thesis an SDN controller as a central entity that had access to all global parameters such as flow information, path structure, link capacities. This unprecedented feature of SDN makes the whole paradigm very desirable for intelligent network control and management. We proposed FD-ADMM, an algorithm that computes, distributively, the α-fair resource allocation problem, given a link-path incidence matrix $\mathbf{A}$, a per link capacity vector $\mathbf{c}$, a set of connection requests $\mathcal{R}$ each with a positive weight $\boldsymbol{w}$. We always assumed the controller had full knowledge of all these parameters, and most of all, that they are fixed.

However in practice, the information that the controller detains on the actual network state is maintained through dynamic parameters that are measured actively and cannot be updated without active monitoring through measurements.

A typical parameter is the remaining available bandwidth on the links of the network. The available bandwidth of a path characterizes the amount of bandwidth that can be allocated along that path without deteriorating the bandwidth of other paths already established before-hand (*e.g.* from a higher priority class of traffic, some Bandwidth on Demand service, other service provider sharing the same physical resources, *etc*) that meets it on some link. Thus the available bandwidth, modeled all thorough this thesis by the fixed capacity vector $\mathbf{c}$, is in fact by definition, a dynamic value.

The maintenance and optimization of inherently dynamic services with rigid performance requirements (QoS) such as video streaming, where adaptive re-routing of flows and re-allocation have a crucial role in QoS guarantees and enhancement, cannot bypass the benefits of available bandwidth monitoring [61, 62]. Several approaches have been proposed for available bandwidth estimation. They are generally based on throwing probing packets through the network and which permit to deduce an estimation of the remaining capacity based on the effects on the links and routers. Among the

developed estimation tools, one can cite Abing [63], Assolo [64], Pathload [65] and PathChirp [66], to name only a few. However, those methods [67] simultaneously suffer a lack of precision, which requires a single estimation to rely on many measurements, and a tendency to generate non-negligible probing traffic per measurement. For instance, Pathload generates from 2.5MB to 10MB per traffic estimation. Thus, they often produce poorly reliable measurements. More precise and reliable estimation tools leveraging the new possibilities with SDN are emerging, such as SOMETIME [68], the first of which was proposed by Megyesi et al. [69], where messages are sent from the controller through the Southbound API to poll counters in the switches and deduce the current bandwidth utilization and derive the available bandwidth at the link level. In the latter paper, the authors show the accuracy (less than 5% error compared to the ground truth) of their method to estimate individual[1] available bandwidth under various traffic conditions and Southbound communication delays and polling periods.

In all generality, we therefore extend our model with a varying available bandwidth scenario, typically by introducing the existence of a background traffic on the network, and the SDN controller is allocating the remaining bandwidth between a set of flows.

In this case, the link capacities form uncertain parameters on which, through measurements, we have a certain amount of information: we assume in this chapter that the network has been observing the background traffic enough time to obtain, for each link, an estimation of the behavior of the remaining capacity in terms of a distribution. This means that the network is able to predict that with probability p, the link j has a remaining capacity less or equal than z_j.

Thus mathematically, it is acceptable for us to assume each link comes with a probability distribution function on its available bandwidth. When the background traffic has a cyclic pattern, it might be considered to have a timely capacity vector (as, as shown in [69], not more than 5 seconds polling can lead to very satisfactorily accurate estimations), that is, we might know a precise capacity vector for each time, for instance, 5 seconds. But we mentioned in the last chapter how it might be costly to reconfigure tunnel sizes too often. It might on the other hand be too inefficient to compute the fair allocation considering the worst case for all measurements made thorough the cycle. Thus, we can define a typical *configuration time-scale* at the end of which a new allocation has to be made, considering a new estimation of the capacity vector. In between this time scale, the allocation can be readjusted (with a fixed estimated capacity vector) when needed in the fashion of Chapter 4 (with or without switching costs) when the weight vector $\boldsymbol{w}$ varies. Therefore, we pose the problem of computing an allocation based on the measurements that are available to the controller, at the end of the configuration time-scale.

To this aim, it is required to extend the α-fair resource allocation model to account for those uncertainties. We point out two fundamental point of

[1] meaning, from each link.

views of how to approach such uncertain environments.

- **Worst scenario:** a robust solution is wanted. In this case, independent of the background traffic, the resource allocation should *never* violate the capacity constraints. We take the minimum value of the possible realizations of each link capacity and thus optimize over the corresponding reduced set. This defines the *robust* solution, which is ultimately conservative (as the optimal solution on a reduced feasible subset is of course sub-optimal);
- **Violation tolerance:** we do not need a robust solution, but a solution that one can guarantee, that with probability at least $1 - \varepsilon$, will respect the capacity constraints, where $\varepsilon \in [0, 1]$. In this way, we sacrifice some feasibility guarantees and of course permit to improve the efficiency of the system (in terms of objective value, here, in terms of fairness). The worst case scenario thus corresponds to the situation $\varepsilon = 0$.

The violation tolerance point of view is quite attractive in practice. It is often acceptable to have a solution that would not violate the constraints *most of the time*, and in the meantime, sacrificing a little bit of feasibility helps improve the quality of the solution *at all times*. It is thus tempting to relax the notion of feasibility to the alternative notion of ε-safety:

DEFINITION 5 (ε-safety)
Assume that the capacity vector $\mathbf{c}$ is a random variable in $(\mathbf{R}^{|\mathcal{J}|}, \mathbf{P})$, where $\mathbf{P}$ is a probability measure. We say that the vector $\mathbf{x}$ is ε-*safe* if:

$$\mathbf{P}(\mathbf{A}\mathbf{x} \leqslant \mathbf{c}) \geqslant 1 - \varepsilon, \text{ or equivalently } \mathbf{P}(\mathbf{A}\mathbf{x} > \mathbf{c}) \leqslant \varepsilon. \qquad (5.1)$$

Thus, when the vector $\mathbf{x}$ satisfies Equation (5.1), we say that $\mathbf{x}$ is ε-*safe* (or simply *safe* when there is no ambiguity on ε, which will be the case). On the contrary case, we will say that $\mathbf{x}$ is ε-*unsafe*, or simply *unsafe*. An attractive way to capture the notion of ε-safe solutions to an optimization problem with uncertainties is chance constrained programming.

Nonetheless, chance constrained programs are generically non-convex, not even tractable, and under the most general assumptions, give rise to NP-hard problems. As will be shown, our extension model is based on chance-constrained programming and invokes convex optimization programs with many additional binary variables, which make it much more difficult to solve. Tractable and efficiently solvable approximations of chance constrained programs can be obtained through convexification tools, but are often too conservative, returning sub-optimal solutions within an *inner* approximation of the set of safe points. Thus, we propose a polishing routine, Polish, to improve greedily such inner approximations, and build an algorithm, *Safe*, to compute a safe allocation that improves a convex *outer* approximation of the mixed binary convex program. In a certain way, Polish mimics

the behavior of the well-known greedy algorithm to maximize sub-modular functions [70, 71, 72], a general iterative ascent method that uses the concept of marginal value[2] to decide greedily which direction to choose in the ascent, but while only estimating and not evaluating all marginal values in order to go quicker[3].

This last chapter is organized as follows. Section 5.1 introduces the concept of chance constrained programming. Considering our setting and according to [69], we design the safe and fair resource allocation problem with individual chance constraints and discrete probability densities. Then, in Section 5.2, we describe the mixed binary convex optimization problem that solves the problem optimally. This problem being intractable, we present the natural lower bound given by relaxing the binary constraints, efficiently computable as a convex optimization problem. Next (Section 5.3) we introduce the concept of sensitivity analysis with the help of which we define our polishing algorithm, after which the *Safe* algorithm is designed (Section 5.4). Finally, Section 5.5 is dedicated to the validation of *Safe* by comparison with classic inner approximations (that will be introduced thereby), and Section 5.6 concludes the chapter.

[2] Briefly, let $F : 2^{\mathcal{X}} \to \mathbf{R}$ be a general set-function, $\mathcal{X}$ being a discrete finite set. For $S \subset \mathcal{X}$, the marginal value function of F at x maps a set S to the value $F(S \cup \{x\}) - F(S)$. We say that F is sub-modular if its marginal value functions are all non-increasing (for the partial order $\subset$).

[3] In fact, the standard greedy algorithm requires to compute all possible marginal values in order to choose the best ascent direction. In our case, as the reader will see later, this requires to solve $|\mathcal{J}|$ instances of the (classic) fair resource allocation problem at each polishing iterations, which we reduce to only *one* by only estimating the best marginal value instead of finding it.

IN A NUTSHELL

We extend the α-fair resource allocation problem to account for uncertainties in the available bandwidth over the set of links of the network. The variability of the remaining available bandwidth lies beneath an introduced existence of a background traffic over the network, that the SDN controller is able to monitor but not control (higher priority class already established, other application sharing the same resources, *etc*), on top of which it is allocating bandwidth fairly between a set of flows. In this situation, we argue that the SDN controller may build information on the possible available bandwidth over individual links in form of general (discrete) density functions, and extend our model to account for this form of uncertainty in the capacity vector $\mathbf{c}$, that is now seen as a random variable, although so far assumed as fixed and known (typically, no background traffic). In this chapter, we introduce briefly the framework of chance constrained programming with right-hand side uncertainties, and extend the notion of feasibility to the one of ε-safety. We design an algorithm, *Safe*, that given a collection of density functions $\mathbf{p}_j$ for each link, and a total tolerated violation risk ε, returns an ε-safe allocation. *Safe* relies on a polishing sub-routine that takes unsafe solutions back to the safety set, improves safe solutions in terms of fairness, based on the sensitivity of the α-fair resource allocation problem. We show numerically that the fairness of this safe allocation is satisfactory, making *Safe* a good candidate as an efficient heuristic.

5.1 Introduction on chance constraints programming

Chance-constrained programming is a very natural way to formulate the problem of safe allocations based on parameters uncertainties. In this chapter, we focus on our introduced use-case where the uncertainties are limited to the network capacities. Mathematically, this means that our optimization problem contains uncertainties on the right-hand side of the inequality constraints. Let us fix a probability measure $\mathbf{P}$ on the natural (borelian) measured space structure of $\mathbf{R}^{|\partial|}$. The general chance-constrained problem with right-hand side uncertainty can be formulated as the following:

$$\begin{aligned} &\min\ g(\boldsymbol{x}) \\ &\text{s.t. } \mathbf{P}(\boldsymbol{A}\boldsymbol{x} \leqslant \mathbf{c}) \geqslant 1-\varepsilon, \end{aligned} \tag{5.2}$$

where $\mathbf{c}$ is a random variable of the measured space $\mathbf{R}^{|\partial|}$.

The problem posed by (5.2) is commonly called joint-chance-constrained problem. The typical feature lying behind the word “joint" is that the tolerance ε controls the reliability level of the vector $\mathbf{x}$ regarding a set of (more than one) probabilistic constraints, depending on a random variable $\mathbf{c}$ in

the ground space $\mathbf{R}^{|\mathcal{J}|}$. In all generality, the joint reliability level $\mathbf{P}(\mathbf{A}\mathbf{x} \leqslant \mathbf{c})$ is challenging to even evaluate point-wise as it requires the knowledge of the law of the joint variable $\mathbf{c}$; this makes the general joint-chance constrained optimization very difficult to solve (without further assumptions on the probability distributions that can for instance guarantee convexity of the program - see for instance [73]).

In practice, the monitoring of the background traffic permits to maintain information on the remaining capacities of each link of the network. Therefore, an explicit individual probability distribution from the separate observations is more directly available. Fair enough, it is still generically difficult to evaluate the probability distribution of the joint variable $\mathbf{c}$ from the individual probability distributions of the single variables c_j without further assumption on a possible correlation or independence of the different variables c_j, as this would lead us to restrict the structure of the matrix $\mathbf{A}$, and also on the background traffic that we want to keep unknown. We therefore argue that the individually chance constrained version of the problem is more adapted to the present model:

$$\begin{aligned} &\min g(\mathbf{x}) \\ &\text{s.t. } \mathbf{p}_j((\mathbf{A}\mathbf{x})_j \leqslant c_j) \geqslant 1 - \varepsilon_j, \end{aligned} \tag{P_1}$$

where $\mathbf{p}_j$ is a probability measure on $\mathbf{R}$, and $\varepsilon_j \in (0,1)$ is a *tolerated violation risk* for constraint j.

5.1.1 Model design

Variable tolerated violation risks

On top of the practical interest for our problem, instead of having a block probabilistic constraint as in (5.2), we now come with the constraint-wise separated uncertainties. For each link j, let $\varepsilon_j \in [0,1]$ be a tolerated violation risk. Each link capacity c_j is now seen as a positive random variable with density $\mathbf{p}_j$ and, for simplicity, with support in some fixed interval $[c_{min}, c_{max}]$ not containing[4] 0. The fair resource allocation problem under these conditions, with individually chance constraints can be cast as the following program:

$$\begin{aligned} &\min\ g(\mathbf{x}) \\ &\text{s.t. } \int_{c_{min}}^{(\mathbf{A}\mathbf{x})_j} \mathbf{p}_j \leqslant \varepsilon_j. \end{aligned} \tag{5.3}$$

Now, it is clear that $z \mapsto \int_{c_{min}}^{z} \mathbf{p}_j$ is an increasing function. This means that, if c_j is defined as the ε_j-quantile of the distribution $\mathbf{p}_j$ (that is, if $c_j \coloneqq \inf\{z, \int_{c_{min}}^{z} \mathbf{p}_j > \varepsilon_j\}$), then the feasible set of Problem 5.3 is equal to:

[4] In this way, the robust problem is well-posed and bounded.

$$\{\mathbf{x} : \mathbf{A}\mathbf{x} \leqslant \mathbf{c}\}. \tag{5.4}$$

We call c^j the *value-at-risk* of the j-th variable with risk ε_j and we denote it $\mathrm{VaR}_j\ \varepsilon_j$. As we have seen in the precedent chapters, we can solve (5.4) efficiently. It is important to note that the risk $\underline{\varepsilon} := (\varepsilon_j)_{j\in\mathcal{J}}$ is a design parameter rather than a design variable. This means that the values of $\varepsilon_j, j \in \mathcal{J}$ need to be decided and fixed in advance. Then only, we fall into the framework of convex optimization, by (5.4).

A typical approximation of (5.2) with individually chance-constraints and fixed tolerated violation risks is the Bonferroni approximation. The Bonferroni approximation uses the union bound to split the joint chance constrained program (5.2) into the constraints of (P_1) with $\varepsilon_j = \frac{\varepsilon}{|\mathcal{J}|}$. Indeed, if each constraint of Problem (P_1) is respected with this value of the individual risks ε_j, then the union bound ensures that the constraint of (5.2), where $\mathbf{P}$ is any joint probability measure on the product space $\mathcal{R}^{|\mathcal{J}|}$ generated by $(\mathbf{p}_j)_j$, is respected. However, the Bonferroni approximation responds to the individual risk distribution quite blindly – all the constraints are added the same amount of risk, regardless of the problem structure. In reality, some constraints may not even improve the objective if we loosen them, because they are not tight in the beginning, so it should be of no use to attribute risks on those. Further, if we take into consideration only the t tight constraints (by the solution to the robust problem, in the robust problem) in the Bonferroni approximation, it is not clear that ε_j/t is the best attribution. Clearly, it is more attractive to attribute the individual risks ε_j taking into consideration how well they could improve the objective.

This is why a generalization of the Bonferroni approximation to arbitrary values of the ε_j is relevant.

Now, it is valid to question particular choices of $\underline{\varepsilon}$. For instance, if $\underline{\varepsilon} = 0$, we get the robust problem. Although by definition, a solution to the robust problem is always feasible (that is, never violates the capacity constraints), the robust problems deprives the resource allocation scheme of many possible risk tolerant choices, which may give, a highly conservative optimal solution with unnecessary precaution as the worst case scenario may be attained (i) either rarely, or (ii) at worst, not on all links at the same time. Therefore, we pay the price of absolute and simultaneous individual feasibility, which might not even be required in practice, for efficiency improvement in terms of objective. On the other hand, although positive values of ε_j can sacrifice some feasibility to improve effectively the fitness of the solution, a bad choice of the ε_j can provide no improvement at all: for instance, choosing $\varepsilon_j > 0$ on an un-tight constraint does not improve the solution (this statement will be precised in the next section).

Therefore, we define the vector $\underline{\varepsilon}$ as a design variable, so that an "optimal" set of tolerated violation risks can be chosen.

Practical discrete model

In our model, the *true* value of a link's remaining capacity varies with time, because the background carried flow is itself variable. Our vision of the true value can only be estimated via (a finite number of) measurements.

We thus assume that the link capacities of the links are random variables with discrete support and values following a probability distribution. We assume that a certain number of measurements have been done during the past such that a number of realizations of the link capacities are available along with their *empirical* probability. For each link $j \in \mathcal{J}$, c_j has the set of realizations $\{\xi_j^1, \dots, \xi_j^{n_j}\}$, with respective probabilities $\{p_j^k\}_{k=1,\dots n_j}\}$, where $n_j \geqslant 1$. We assume the scenarios are sorted as $\xi_j^1 \leqslant \dots \leqslant \xi_j^{n_j}$.

Therefore, for a given load $z \in \mathbf{R}_+$, the measure:

$$\mathbf{p}_j(z) \coloneqq \sum_{z > \xi_j^k} p_j^k$$

expresses the probability that the remaining capacity c_j is below z. For any $\varepsilon_j \in (0,1)$, the value-at-risk of the j-th variable with risk ε_j for the distribution $\mathbf{p}_j$ is:

$$\mathrm{VaR}_j(\varepsilon_j) = \xi_j^{q_j}, \text{ where } q_j \coloneqq \sup\{k = 1 \dots n_j \text{ s.t. } \mathbf{p}_j(\xi_j^k) \leqslant \varepsilon_j\}.$$

The discrete distribution formulation with fixed risk vector $\underline{\varepsilon}$ then reads:

$$\begin{aligned} &\min\ g(\boldsymbol{x}) \\ &\text{s.t. } \boldsymbol{A}\boldsymbol{x} \leqslant \mathrm{VaR}(\underline{\varepsilon}), \end{aligned} \tag{5.5}$$

where $\mathrm{VaR}(\underline{\varepsilon})_j \coloneqq \mathrm{VaR}_j(\varepsilon_j)$ is the *value-at-risk vector*.

The robust solution of (P_1) is the resource allocation that maximizes the objective while satisfying the worst-case scenario: $\underline{\varepsilon} = 0$. It therefore reads:

$$\begin{aligned} &\min\ g(\boldsymbol{x}) \\ &\text{s.t. } \boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{\xi}^1, \end{aligned} \tag{R}$$

where $\boldsymbol{\xi}^1 \coloneqq (\xi_j^1)_{j \in \mathcal{J}}$.

We now introduced our model with variable risk tolerances ε_j. Although the risk vector $\underline{\varepsilon}$ can lie in a general compact subset of $\mathbf{R}^{|\mathcal{J}|}$, we here do not make any assumptions on the particular structure of the probabilities $\mathbf{p}_j$ and the joint probability $\mathbf{P}$. For simplicity instead, we assume a given global risk budget of $\varepsilon > 0$ is given and we search for a vector $\boldsymbol{x}$ that gives the best possible solution for Problem (5.5), by considering altogether the possible choices of $\underline{\varepsilon}$ respecting the *risk budget* ε and that can be made to attribute individual risk tolerances ε_j. Specifically, the union bound trivially guarantees that if each constraint j is respected with probability at least $1 - \varepsilon_j$ at some $\boldsymbol{x}$,

then the joint chance constraint $\mathbf{Ax} \leqslant \mathbf{c}$ is respected with probability at least $1 - \varepsilon$, provided $\sum_{j \in \mathcal{J}} \varepsilon_j \leqslant \varepsilon$. Here again, the union bound might still be too conservative, as a constraint. Other constraints of the form $\mathbf{a}^T \underline{\varepsilon} \leqslant \varepsilon, \underline{\varepsilon} \geqslant \mathbf{0}$ can be used, where $\mathbf{a}$ is a positive vector. But the choice $\mathbf{a} = \mathbf{1}$ is very common and sensible.

Thus, we now have a new problem:

$$\begin{aligned} \min \ & g(\mathbf{x}) \\ \text{s.t. } & \mathbf{p}_j((\mathbf{Ax})_j) \leqslant \varepsilon_j \quad \forall j \in \mathcal{J} \\ & \sum \varepsilon_j \leqslant \varepsilon. \end{aligned} \tag{5.6}$$

5.2 A lower bound

5.2.1 Mixed integer convex program

When the support of the capacity distributions are finite, it is possible to formulate the problem with a mixed integer program. For this, we just need to define a binary variable b_j^k that activates or deactivates the k-th scenario for constraint j, which means, $b_j^k = 1$ if the capacity of link j is estimated at ξ_j^k, and $b_j^k = 0$ otherwise. Then, we have $\sum_{j \in \mathcal{J}} n_j$ constraints, of which only $|\mathcal{J}|$ are to be activated. The goal of the program is thus to find the $|\mathcal{J}|$ optimal constraints to activate, that give the best solution in terms of fairness, while respecting the global risk budget ε.

Terminology: The variable $\mathbf{b}$ will be referred to as the *scenario variable*. Let $\mathbf{k} \coloneqq \{k_j\}_{j \in \mathcal{J}}$ be a collection of integers such that $1 \leqslant k_j \leqslant n_j$, for all $j \in \mathcal{J}$. We also call $\mathbf{k}$ a scenario, interpreted as $b_j^k = 1 \iff k = k_j$.

We thus introduce the following optimization problem:

$$\min \ g(\mathbf{x}) \tag{5.7}$$

$$\text{s.t. } \mathbf{Ax} \leqslant \mathbf{z} \tag{5.8}$$

$$\sum_{k=1}^{n_j} \xi_j^k b_j^k = z_j \qquad \forall j \in \mathcal{J} \tag{5.9}$$

$$\sum_{k=1}^{n_j} q_j^k b_j^k \leqslant \varepsilon_j \qquad \forall j \in \mathcal{J} \tag{5.10}$$

$$\sum_{k=1}^{n_j} b_j^k = 1 \qquad \forall j \in \mathcal{J} \tag{5.11}$$

$$\sum_{j \in \mathcal{J}} \varepsilon_j \leqslant \varepsilon \tag{5.12}$$

$$\mathbf{x} \geqslant \mathbf{0}, b_j^k \in \{0, 1\}, \mathbf{z} \geqslant \mathbf{0}. \tag{5.13}$$

Equation (5.8) are the classic capacity constraints, except that now the activated capacity is a variable z whose components are chosen by the following constraints of Equation (5.9). The individual risk tolerances are enforced by Equation (5.10), where we define q_j^k as the probability that $c_j < \xi_j^k$:

$$q_j^k := \sum_{l=1}^{k-1} p_j^l, \tag{5.14}$$

with the convention that the empty sum equals 0. A unique scenario is activated per constraint, which is formalized by Equation (5.11). Finally, the individual risks of the program respect the total budget of risk ε, as simply indicates the constraints (5.12).

Overall, it is clear that (5.7)– (5.13) finds the optimal resource allocation fairness on a selected capacity set chosen so as to respect the total allowed violation tolerance budget ε. Unfortunately, this program is an instance of the optimized Bonferroni approximation approach which in general leads to NP-hard problems [74, 75]. Therefore, there is no hope of solving this problem optimally and efficiently for large instances.

This is why we first get a lower bound on the optimal value by relaxing the integrity constraints.

5.2.2 The convex relaxation

It is easy to derive the convex relaxation of Problem (5.7)–(5.13): we just define $b_j^k \in [0,1]$ instead of $b_j^k \in \{0,1\}$. In other words, we get a convex optimization problem at hand, that can again be addressed itself with an adaptation of FD-ADMM[5]. Thus, solving this relaxation can be done very efficiently, and it gives a first estimation of the optimal right-hand-side scenarios to activate. The relaxed problem therefore reads:

[5]Not immediate but doable: one needs to throw the all the variable right-hand-sides to the left side of the constraints in order to have a constant right-hand-side, gather all the inequality constraints to the form $\mathbf{A}'(\mathbf{x}, \mathbf{b}, z) \leqslant \mathbf{d}$ and then include the additional constraints (5.11) in the fashion of Section 4.3.2

$$\min\ g(\boldsymbol{x}) \tag{5.15}$$
$$\text{s.t. } \boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{z} \tag{5.16}$$
$$\sum_{k=1}^{n_j} \xi_j^k b_j^k = z_j \qquad \forall j \in \mathcal{J} \tag{5.17}$$
$$\sum_{k=1}^{n_j} q_j^k b_j^k \leqslant \varepsilon_j \qquad \forall j \in \mathcal{J} \tag{5.18}$$
$$\sum_{k=1}^{n_j} b_j^k = 1 \qquad \forall j \in \mathcal{J} \tag{5.19}$$
$$\sum_{j\in\mathcal{J}} \varepsilon_j \leqslant \varepsilon \tag{5.20}$$
$$\boldsymbol{x} \geqslant 0, b_j^k \in [0,1], \boldsymbol{z} \geqslant 0. \tag{5.21}$$

Of course, the value of the relaxed problem is a lower bound on the optimal value. An optimal solution with the fractional values $\tilde{b}_j^k$ has fractional activations of the different scenarios, and we know that for all j, $\sum_k b_j^k = 1$. Thus, one can build a binary vector $\boldsymbol{b}$ by rounding this fractional solution: the simplest solution would be to project $\tilde{\boldsymbol{b}}$ onto the set defined by (5.11) as well as binary constraints $b_j^k \in \{0,1\}$. This means that the highest value of $\tilde{b}_j^k$ is set to 1, while all other values are set to 0. Other rounding schemes can also take into account the constraints (5.9) and (5.10). In other words, they would require to find the projection of the point $\tilde{\boldsymbol{b}}$ onto the discrete set described by those constraints. But this gives problems that are also intractable in general.

Instead, in the next section, we design a polishing routine that can take the rounded solution as described above and locally improves it by (i) taking a step back into the safe set if the rounded scenario does not satisfy constraints (5.12), and/or (ii) improving the solution by searching for neighboring scenarios that are safe and that augment the value of the program. The polishing routine is based on sensitivity analysis and mimics the well-known greedy algorithms employed to maximize submodular functions [70, 72].

5.3 A polishing routine based on sensitivity analysis

In this section, we introduce formally the question of sensitivity of the objective with respect to the right-hand-side (link capacities) of the constraints. For this, we once again need the dual variables $\boldsymbol{\lambda}$ associated to the capacity constraints. This discussion will help us to build a polishing routine that approaches the critical set $\{\sum_j \varepsilon_j = \varepsilon\}$ from the outside or the inside in order to step back into the safety set/improve best the objective from a conservative

solution.

Let us refresh the notations by reminding the basic problem of fair allocation and its dual.

The α-fair resource allocation problem reads:

$$\begin{aligned} \min\ & g(\mathbf{x}) \\ \text{s.t. } & \mathbf{A}\mathbf{x} \leqslant \mathbf{c}, \\ & \mathbf{x} \geqslant \mathbf{0}. \end{aligned} \tag{5.22}$$

If $\boldsymbol{\lambda}$ denotes the dual variable associated with the capacity constraints $\mathbf{A}\mathbf{x} \leqslant \mathbf{c}$, then the dual of (5.22) reads:

$$\begin{aligned} \max\ & h(\boldsymbol{\lambda}) \\ \text{s.t. } & \boldsymbol{\lambda} \geqslant \mathbf{0}, \end{aligned} \tag{5.23}$$

where $h(\lambda) \coloneqq \min_{\mathbf{x}\geqslant\mathbf{0}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda})$, and $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) \coloneqq g(\mathbf{x}) + \boldsymbol{\lambda}^T(\mathbf{A}\mathbf{x} - \mathbf{c})$ is the Lagrangian function of (5.22).

We remind that the primal-dual pair (5.22)–(5.23) problem responds to the strong duality theorem and that the primal-dual optimal corresponding pair $(\mathbf{x}, \boldsymbol{\lambda})$ (is unique and) verifies the KKT conditions:

- complementary slackness: $\boldsymbol{\lambda}^T(\mathbf{A}\mathbf{x} - \mathbf{c}) = 0$,
- optimality: $x_r = \left(\frac{w_r}{\sum_{j\in\mathcal{J}_r} \lambda_j}\right)^{1/\alpha}$,
- dual feasibility: $\boldsymbol{\lambda} \geqslant \mathbf{0}$,
- primal feasibility: $\mathbf{A}\mathbf{x} \leqslant \mathbf{c}$.

Applying the KKT conditions to the robust problem (R), one can remark the well-known but important property: if the constraint j is not active, that is if $(\mathbf{A}\mathbf{x}^*)_j < \xi^1_j$, then $\lambda^*_j = 0$. This property implies that a positive dual variable λ^*_j is a certificate of the tightness of the constraint j. Therefore, if we want to add some more risk on a constraint, that is, to augment the corresponding risk tolerance ε_j, we should be looking at the constraints with a positive optimal dual variable, because the tight constraints are the good candidates for bottlenecks that can improve the objective when liberated, not the untight constraints. Unfortunately, the KKT conditions do not permit to quantify this observation, nor do they tell what can be deduced when $\lambda^*_j = 0$. The constraint might as well be tight. Sensitivity analysis responds partially to this question.

Let us introduce the following general *perturbed problem*, with reference to the robust problem (R):

$$\begin{aligned} \min\ & g(\mathbf{x}) \\ \text{s.t. } & \mathbf{A}\mathbf{x} \leqslant \boldsymbol{\xi}^1 + \mathbf{u}. \end{aligned} \tag{5.24}$$

This problem was introduced in Chapter 2 where we discussed the behavior of the *optimal solution* x^* when the problem was perturbed. Here, we are interested with the behavior of the *optimal value* of the perturbed problem with respect to the perturbation. Let p denote the *value function*:

$$\text{For all } u \in \mathbf{R}^{|\mathcal{J}|} \text{ such that } \xi^1 + u \geqslant 0, \quad p(u) \coloneqq \min_{Ax \leqslant \xi^1 + u} f(x).$$

The link between p and our chance constrained problem is the following. For a risk $\underline{\varepsilon}$, let $c(\underline{\varepsilon})$ be its value-at-risk, that is, $c_j(\varepsilon) \coloneqq \mathrm{VaR}_j(\varepsilon_j)$. Then, the optimal value of Problem (5.5) for $c^* = c(\underline{\varepsilon})$ is $p(c(\underline{\varepsilon}) - \xi^1)$. Therefore, solving (5.5) is equivalent to minimizing $p(c(\underline{\varepsilon}) - \xi^1)$ subject to the constraints $\sum \varepsilon_j \leqslant \varepsilon$, where ε is a chosen joint risk tolerance. Of course, it is not trivial to tackle the direct minimization of p over such packing constraints, for $p(c(\cdot) - \xi^1)$ is, as a sub-routine, not immediate to evaluate. Nonetheless, the value function p gives precious information on the sensitivity of the program with respect to the right-hand-side.

5.3.1 Regularity and sensitivity

We have the first convexity property of the value function p:

PROPOSITION 4
The value function p is convex.

Proof. For a perturbation u, we denote by C_u the corresponding feasible set in (5.24). Let u and v be two perturbations and $t \in [0, 1]$. We have:

$$tC_u + (1-t)C_v \subset C_{tu+(1-t)v}.$$

Therefore, for x_u and $x_v \in C_u$ and C_v, respectively;

$$\begin{aligned} p(tu + (1-t)v) &= \min_{x \in C_{tu+(1-t)v}} g(x) \\ &\leqslant g(tx_u + (1-t)x_v) \\ &\leqslant tg(x_u) + (1-t)g(x_v). \end{aligned} \tag{5.25}$$

This being true for all $x_u \in C_u$, $x_v \in C_v$, we thus have the property:

$$p(tu + (1-t)v) \leqslant tp(u) + (1-t)p(v).$$

□

PROPOSITION 5
For any perturbation $\mathbf{u}$, there exists an unique primal-dual pair $(\mathbf{x}_\mathbf{u}, \boldsymbol{\lambda}_\mathbf{u})$ to (5.24). Moreover, for all couple of perturbations $(\mathbf{u}, \boldsymbol{\nu})$, we have the sensitivity inequality:

$$p(\boldsymbol{\nu}) - \boldsymbol{\lambda}_\mathbf{u}^T(\mathbf{u} - \boldsymbol{\nu}) \geqslant p(\mathbf{u}) \geqslant p(\boldsymbol{\nu}) - \boldsymbol{\lambda}_\boldsymbol{\nu}^T(\mathbf{u} - \boldsymbol{\nu}). \tag{5.26}$$

Proof. By optimality of the primal-dual pair $(\mathbf{x}_\mathbf{u}, \lambda_\mathbf{u})$, we have:

$$g(\mathbf{x}_\mathbf{u}) + (\mathbf{A}\mathbf{x}_\mathbf{u} - \boldsymbol{\xi}^1 - \mathbf{u})^T\boldsymbol{\lambda}_\mathbf{u} \leqslant g(\mathbf{x}_\boldsymbol{\nu}) + (\mathbf{A}\mathbf{x}_\boldsymbol{\nu} - \boldsymbol{\xi}^1 - \mathbf{u})^T\boldsymbol{\lambda}_\mathbf{u}. \tag{5.27}$$

Moreover, the complementary slackness ensures that the inner product on the left-hand-side of (5.27) is zero. Also, $\mathbf{A}\mathbf{x}_\boldsymbol{\nu} - \boldsymbol{\xi}^1 - \mathbf{u} \leqslant \boldsymbol{\nu} - \mathbf{u}$, as $\mathbf{x}_\boldsymbol{\nu} \in C_\boldsymbol{\nu}$. Therefore, plugging these two observations into the above inequality, and recalling that $\boldsymbol{\lambda}_\mathbf{u} \geqslant \mathbf{0}$, we get:

$$g(\mathbf{x}_\mathbf{u}) \leqslant g(\mathbf{x}_\boldsymbol{\nu}) + (\boldsymbol{\nu} - \mathbf{u})^T\boldsymbol{\lambda}_\mathbf{u}. \tag{5.28}$$

Thus,

$$p(\mathbf{u}) - p(\boldsymbol{\nu}) \leqslant -(\mathbf{u} - \boldsymbol{\nu})^T\boldsymbol{\lambda}_\mathbf{u}, \tag{5.29}$$

which is the left inequality. The right inequality is obtained by interchanging $\mathbf{u}$ and $\boldsymbol{\nu}$. □

As a direct consequence, $-\boldsymbol{\lambda}_\boldsymbol{\nu}^T$ is a sub-gradient of p at $\boldsymbol{\nu}$.

The last proposition gives us an insight on the behavior of the value function. Let $\boldsymbol{\nu} \leqslant \mathbf{u}$. From perturbation $\boldsymbol{\nu}$ to $\mathbf{u}$, we know that we cannot improve the objective (as a convex minimization problem, to improve means to decrease) to below the value $p(\boldsymbol{\nu}) - \boldsymbol{\lambda}_\mathbf{u}^T(\mathbf{u} - \boldsymbol{\nu})$. Therefore, when a dual variable $\lambda_{\boldsymbol{\nu},j}$ associated to a constraint j is very small, we know that the interest of adding a perturbation to the corresponding constraint is very limited. On the other hand, when the dual variable is very high, we cannot quantify the added value to a perturbation, but we know at least that there is no such limitation to the improvement. However, in the latter case, if we tighten the constraint j, the last proposition tells us that the objective is guaranteed to deteriorate greatly. All in all, we can qualify the sensitivity of the perturbed problem as follows:

1. If $\lambda_{\boldsymbol{\nu},j}$ is big, tightening the constraint j deteriorates the objective greatly.
2. If $\lambda_{\boldsymbol{\nu},j}$ is small, loosening the constraint j does not improve much the objective.

5.3.2 Polishing routine

The two above properties permit to define a simple iterative algorithm that adds (positive or negative) perturbations greedily to the capacity vector

while keeping track of the optimal dual variable.

Algorithm 6 Polish

Input: A scenario $\mathbf{b}$, the corresponding risk and capacity vector $\mathbf{u}$

$k_j \coloneqq$ the unique index $k \in [1, n_j]$ s.t. $b_j^k = 1 \quad \forall j \in \mathcal{J}$.
while risk $> \varepsilon$ **do** ▷ The scenario $(k_j)_j$ is unsafe
  $(\mathbf{x_u}, \boldsymbol{\lambda_u}) \leftarrow$ the new primal dual pair on $C_{\mathbf{u}}$.
  $j = \operatorname{argmin}_j \lambda_{uj}$
  $k_j \leftarrow k_j - 1$
  risk $\leftarrow$ risk $- p_j^{k_j}$
  $u_j \leftarrow \xi_j^{k_j}$
end while ▷ Now, the current scenario $(k_j)_j$ is safe.
while risk $< \varepsilon$ **do**
  $(\mathbf{x_u}, \boldsymbol{\lambda_u}) \leftarrow$ the new primal dual pair on $C_{\mathbf{u}}$.
  $\mathcal{H} \coloneqq \{j; \text{risk} + p_j^{k_j} \leqslant \varepsilon\}$
  if $\mathcal{H} = \varnothing$ **then**
    Terminate
  else
    $j = \operatorname{argmax}_{j \in \mathcal{H}} \lambda_{uj}$
  end if
  risk $\leftarrow$ risk $+ p_j^{k_j}$
  $k_j \leftarrow k_j + 1$
  $u_j \leftarrow \xi_j^{k_j}$
end while

Output: $(\mathbf{x_u}, \lambda_u, \text{risk})$

Algorithm 6, which we hereafter refer to as Polish, is in fact inspired of the well-known greedy algorithm that finds a solution to the problem of maximization of submodular functions. Basically, a submodular function has non-decreasing marginal values. The idea is therefore, from a suboptimal solution, to add the positive perturbation to the problem that maximizes the (non-negative) marginal value. Conversely, starting from a superoptimal solution (hence, unsafe), one can add negative perturbations to adequate constraints in order to step back into the safe set. In this case, the adequate constraints are the ones with the smallest marginal values (so that the tightening of the constraint costs as little objective value as possible). Thus, knowing that the marginal value is non-decreasing, we have the guarantee that at each step, the best "greedy" decision is taken, that is, the perturbation is added to the constraint that improves immediately the objective the most. It is clear that this scheme does not necessarily lead to the optimal solution. However, it permits to build, or improve, a solution locally in reduced time and in an online fashion. The difference with (our) Polish, is that instead of computing all possible marginal values (which means solving at most $|\mathcal{J}|$ problems to compare their values), we evaluate what could be the best possible marginal value by means of the dual optimal variable. Given

our discussion in the previous Section, we know where it is more interesting to add/remove perturbation: see observations 1. and 2. above. This does not guarantee that the constraint with the greatest dual optimal value leads to the best marginal improvement, or that the smallest dual optimal value leads to the smallest degradation of the objective value when the corresponding constraint is tightened, though. Polish can be somehow seen as a blurry way to enforce the general greedy algorithm. Per extension, one can imagine, for instance, a hybrid polishing consisting of a combination of the two, where the marginal values are effectively computed for the q best candidates with the greatest (or lowest for negative perturbations) values of λ_{uq} at each step of the iterative process. We will nevertheless demonstrate that the approach of Polish gives satisfactory results. Thus, we are able to define an heuristic algorithm that computes a safe and fair resource allocation solution.

5.4 Safe and α-fair resource allocation problem

We are now ready to define our *Safe* algorithm. First, we compute the relaxed solution given by relaxing the binary constraints on the scenario variable $\mathbf{b}$ (line 1). Thus, we solve Problem (5.15)– (5.21). This gives a fractional scenario, $\tilde{\mathbf{b}}$, that we next round by projecting on the space $\{\mathbf{b} \in \{0,1\}^{|\mathcal{J}|}; \forall j \in \mathcal{J} \sum_k b_j^k = 1\}$ (line 2). This results in taking, for each j, the greatest value of $\tilde{b}_j^k, k = 1 \dots n_j$ and setting it to 1, while all other values are set to 0. Consequently, we get a new binary scenario $\mathbf{b}$, that forms the starting point for Polish (line 3).

Of course, the scenario $\mathbf{b}$ might be unsafe, *i.e.*, the vector $\mathbf{b}$ might not respect the constraints (5.10) and (5.12) anymore. But this situation is controlled by the first loop (Algorithm 6, line 2).

At the end of Polish, it is clear that the returned scenario vector $\mathbf{b}^*$ is safe. Then, the first safe scenario vector that Polish encountered, was improved greedily with the help of the sub-gradient $-\boldsymbol{\lambda}^*$ of the value function p at the current fair allocation (corresponding to the current activated scenario). The three stages of *Safe* are summarized in Algorithm 7.

Algorithm 7 *Safe*

1: $\tilde{\mathbf{b}} \leftarrow$ the fractional optimal scenario of Problem (5.15)– (5.21).
2: $\mathbf{b} \leftarrow$ rounded $\tilde{\mathbf{b}}$ by projecting onto the binary space with constraint (5.11).
3: $(\mathbf{x}^*, \mathbf{b}^*) \leftarrow$ the final α-fair allocation onto the corresponding chosen scenario by Polish (Algorithm 6).

Output: $(\mathbf{x}^*, \mathbf{b}^*)$

It is crucial to remark that the optimal solution in line 1 corresponds to a *lower bound* of the optimal value (we have a minimization problem at hand, and we have relaxed some constraints). Conversely, the output of Polish is

always an *upper bound* on the optimal value, as the solution is always ε-safe. Therefore, we possess two estimations of the optimal value, a lower bound, and an upper bound. If those two are close enough, we have a guarantee on the actual performance of *Safe*, based on the gap between the lower bound and the upper bound. Thus, in the next section, we conduct simulations to illustrate the performance of *Safe*.

5.5 Numerical results

In this section, we show our simulation results in the evaluation of *Safe*. First, we describe the setting in terms of the instances that we consider, as well as the distributions $\mathbf{p}_j$.

5.5.1 Settings and benchmarks

The instances that we consider are all built on the Fat-Tree [76] model[6] with a number of $k = 4$ or 8 pods. This gives small networks of 32 links and average-sized networks of 256 links, respectively.

To generate requests, we concatenate to the topology a *root node* that models the connection of the network to the exterior through the core nodes. Then, for each of the $k^3/4$ server nodes, two connection requests are generated from this node to the root node on top of the core nodes through 4 different paths. The links between the root and the core nodes do not form part of the problem and were just a tool to generate paths from the same requests with different endpoints from the core nodes.

Thus, for each server node, we modeled two connection requests that go down from and up to a fictional external network (the Internet, or other data centers, for instance, in a WAN) through different paths, and we are therefore in the multi-path setting. As a consequence, each instance comes with a set of $k^3/2$ requests $r \in \mathcal{R}$ each with a uniformly drawn weight w_r under the same conditions as in the simulations of Chapter 3.

Concerning the probability densities $\mathbf{p}_j$, we did not find well-documented natural models. Therefore, we created two typical settings that we describe below, under which we evaluate the algorithm:

1. First, we assumed each link capacity follows a Poisson distribution with a specific mean value per link drawn uniformly at random within the interval $(30, 100)$. The choice of the Poisson distribution was arbitrary and was made in order to have a mean value with high probability as well as a typical decrease on both sides of the mean value.

2. Second, we built synthetic capacity distributions by throwing, for a chosen capacity c_0, a background traffic generated following a birth and death process. Specifically, for one process of N units of time, we

[6] This topology model was originally proposed in [76] and was advocated for data center topologies in [77].

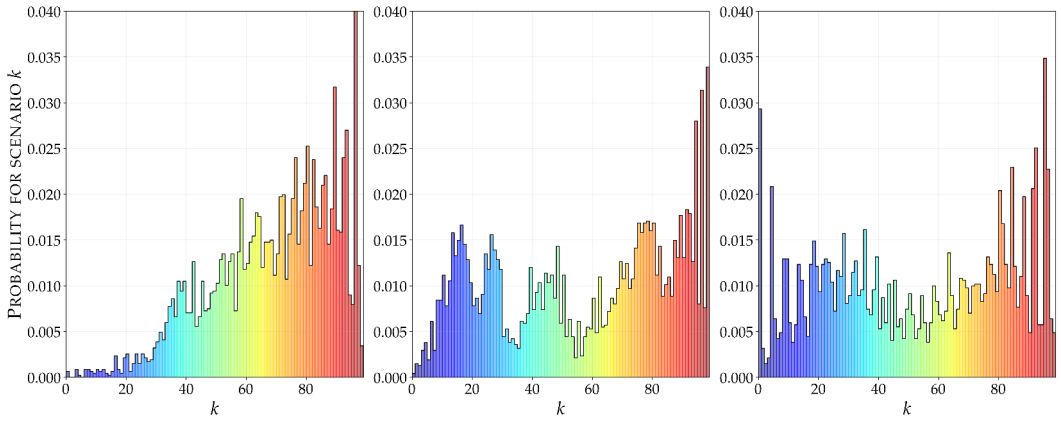


Figure 5.1 | Some examples of the generated distributions $\mathbf{p}_j^k$, for a number of bins $K = 100$.

generated a background traffic t that increased of b_+ at rate $\beta \in [0, 1]$, and decreased of b_- at rate $\delta \in [0, 1]$. In order to always have a well-defined robust solution, we always bounded $t \leqslant \nu c_0$, with $\nu \in [0, 1]$. Then, we built the histogram based on the realizations of the number $c_0 - t$ and converted the result into a probability density. This process was very simple and yet gived various density structures (with one or many spikes, with a most likely extreme value and a vanishing tail to the other extreme, *etc*). Some examples are illustrated in Figure 5.1. In practice, we used $b_+ = 1.1$, $b_- = 1.15$, $\beta = .33$, $\delta = .5$, and $\nu = 0.9$.

In both cases, the density functions of each link has a number K of bins that we also vary. This parameter illustrates the effects of the precision of the monitoring of the available bandwidth on the performance of *Safe*: the greater the number of bins, the more precise the distribution, but the more the created scenarios. K is in fact an instance of the numbers n_j (introduced in the definition of the discrete model, see Section 5.1.1), that for simplicity, we keep all equal within the same instance.

As the exact problem is intractable and impossible for us to solve exactly within reasonable computation time (tens of minutes), we compare the solutions of *Safe* with the ones of the relaxed problem (5.15)– (5.21), that we from now on name *Relaxed*, together with the ones of the Bonferroni inner approximation, that we call *Bonferroni*. Also, we compare with the worst-case solution of (R), *Robust*.

5.5.2 Link capacity with Poisson distributions

Optimality gap for various instances

For each value of $\alpha = 1, 2$ or 3, we first generated a total of 250 instances, all under the same setting. Each run of the algorithm *Robust*, *Safe*, *Relaxed* or *Bonferroni* on an instance yielded one single point. Although only small

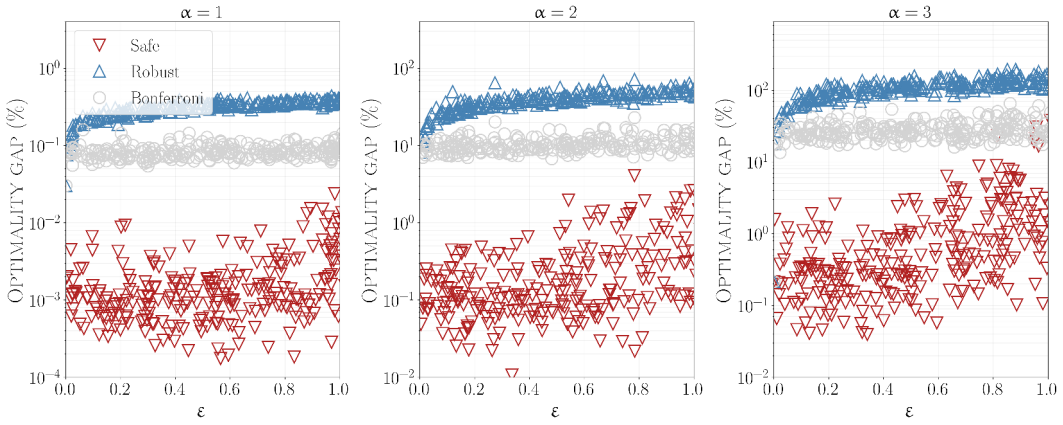


Figure 5.2 | Achieved gaps (with reference the *relaxed* optimum) for the simulations under setting 1.

values of ε are interesting (for instance, $\varepsilon \leqslant \frac{1}{2}$) we generated instances with $\varepsilon \in (0, 1)$. Thus, one instance corresponds to a particular collection of:

1. the collection of distributions $\mathbf{p}_j$, that here have an individual mean value,
2. the tolerated risk ε,
3. the number of bins (or scenarios per link) K.

Other than those three parameters, all the instances are identical.

We remind that the relaxed problem gives a lower bound on the optimal value, and that *Robust*, *Safe* and *Bonferroni* each give an upper bound, as they give by definition safe allocations. We here call *estimated optimality gap* the actual gap between a solution and a reference value, here of course taken as the relaxed value. We drive the reader's attention onto the important fact:

- A good estimated optimality gap guarantees a good actual optimality gap (between a solution and the actual optimum, that we cannot compute here) of a solution.
- A bad estimated optimality gap *does not mean that the solution has a bad actual optimality gap – with reference the true optimum*. Indeed, it might just be that the relaxation is a bad lower bound.

This remark being done, we hereafter say "optimality gap", instead of "estimated optimality gap", without fear of misleading by confusion.

In Figure 5.2, we represented the optimality gap achieved by Robust, *Safe* and *Bonferroni* for all the generated instances, for the three values of α.

From the results, it is clear that *Safe* outperforms considerably the *Bonferroni*. For $\alpha = 1$, although the performed gaps are all good (less than 1%), the value of the gap for *Robust* shows that the proportionally fair allocation value does not seem too sensitive to the added perturbations in the capacity

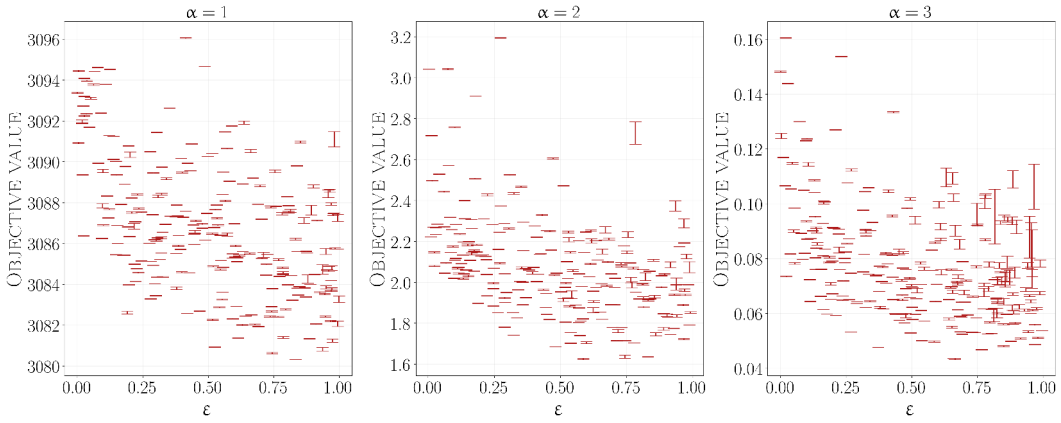


Figure 5.3 | Bounding the optimal value under setting 1. The bottom of each error bar corresponds to the value obtained by *Relaxed*, whereas the top corresponds to the value obtained by *Safe*.

vector. This might also be specific to the distribution models, and the topology. Notwithstanding, *Safe* provides a gap of nearly two orders of magnitude smaller *Bonferroni*. The benefit of *Safe* is more obvious for the cases $\alpha = 2$ and 3. Likewise, the obtained gap is two orders of magnitude smaller than the one for *Bonferroni*, that performs with a gap of around 50%. In the two latter cases, the gaps provided by *Robust* show that there is a substantial possible improvement of α-fairness for $\alpha = 2, 3$, motivating the central idea of this chapter, of relaxing strict feasibility to ε-safety, which *Safe* operates with very satisfactory results.

In order to provide a practical appreciation of how the two approaches, *Relaxed*, and *Safe*, bound the true optimal value, we plotted as error bars the achieved values, in Figure 5.3.

Overall, according to Figures 5.2 and 5.3, it is clear that the approach of *Safe* permits one to get relatively satisfactory estimations of the safe fair allocation under uncertain available bandwidth knowledge in setting 1.

Number of convex problems to solve

Lastly, the reader has already seen that the application of *Safe* algorithm implies to solve a certain number of the α-fair resource allocation problem. Following the work that has been done in the precedent chapters, we have indeed demonstrated that it is now easy to solve such problems very efficiently. Nevertheless, we show, for the same experiment, the number of iterations achieved by *Safe*, if need be, in order to make the rounded scenario from 6–line 2 safe (which we call number of *outer iterations*), as well as the number of iterations achieved in order to make the feasible scenario from 6–line 9 (which we call number of *inner iterations*). Hence, the notion of iteration here corresponds to the computation of the dual variables, which in turns can be done by solving optimally an instance of the fair resource allocation problem.

Unfortunately, for the two last introduced figures 5.3 and 5.4, it was not

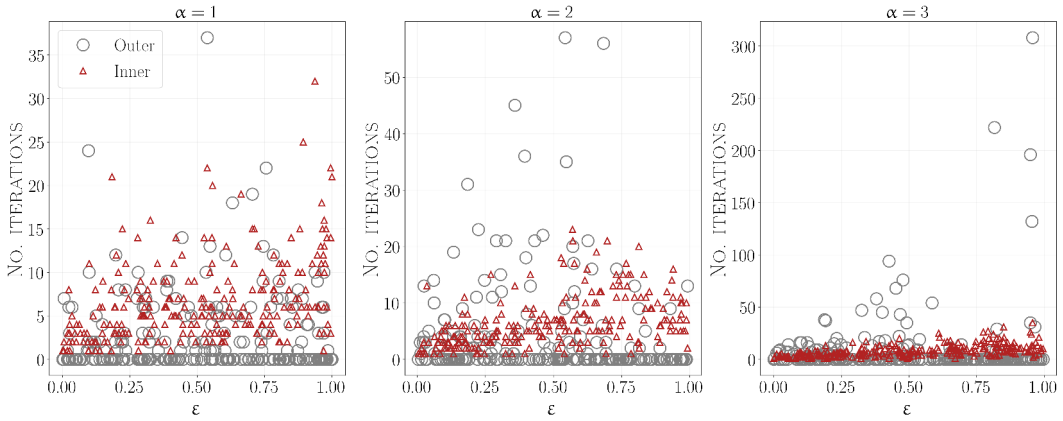


Figure 5.4 | Achieved number of inner and outer iterations by *Safe* during polishing under setting 1.

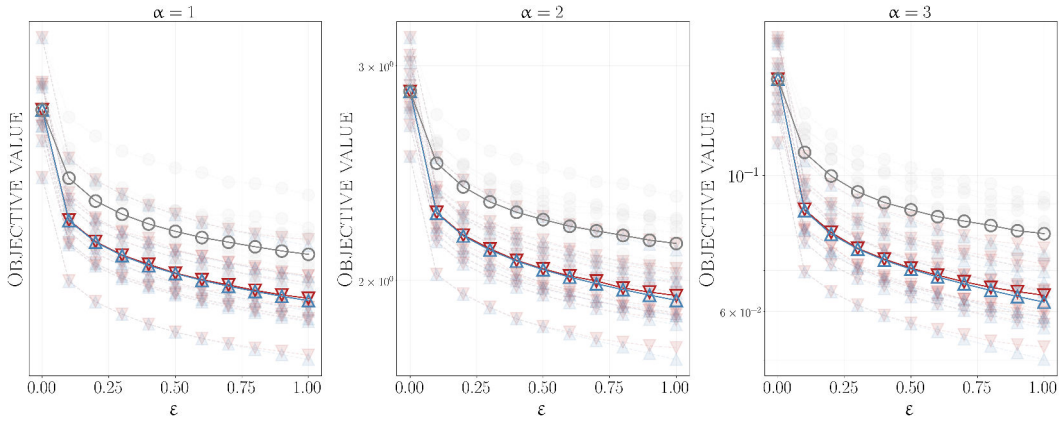


Figure 5.5 | The fairness value for *Safe* (red), *Relaxed* (blue) and *Bonferroni* (gray) against the risk ε under setting 1.

obvious to derive a possible structure of the behaviors with respect to a parameter. Arbitrarily, we represented all the figures as clouds against their respective values of the tolerated risk ε.

In particular, this is why we illustrated, on Figure 5.5, on a same fixed instance, the evolving values of *Safe* and *Bonferroni*, as well as the one of *Relaxed*, against the risk ε. This figure comforts the intuition that while the risk ε grows, one is able to improve the fairness, and confirms the expected performance of *Safe* against the ones of *Bonferroni*. Ten curves are illustrated each corresponding to a fixed instance, and while the specifics are plotted with translucent markers, the average curve is shown in solid lines and empty markers.

5.5.3 General distributions

We now present the same results under the setting 2.

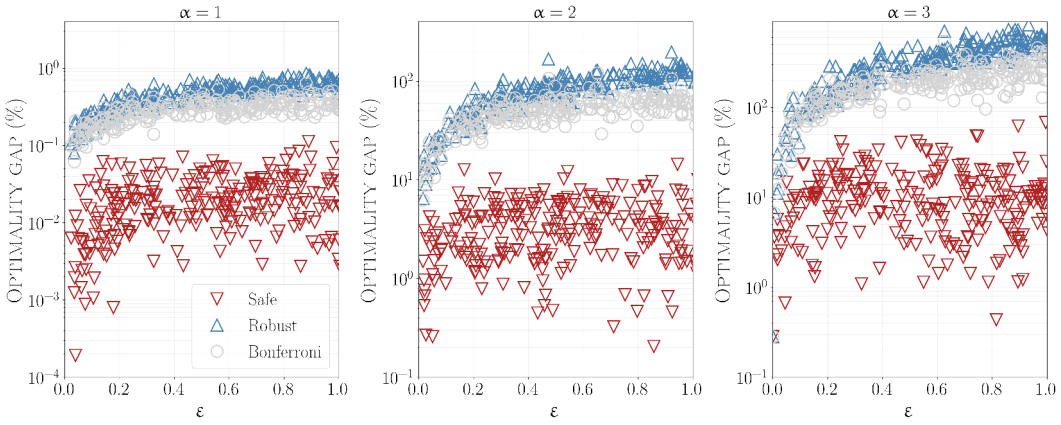


Figure 5.6 | Achieved gaps (with reference the *relaxed* optimum) for the simulations under setting 2.

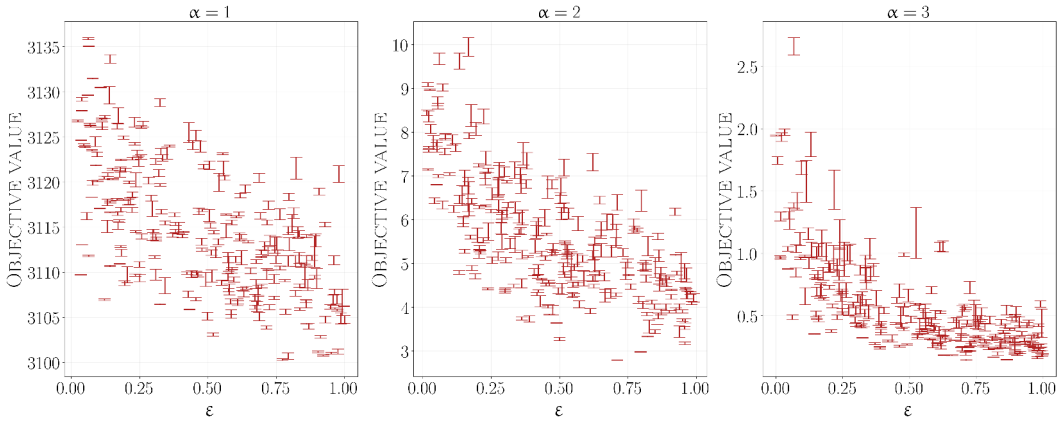


Figure 5.7 | Bounding the optimal value under setting 2. The bottom of each error bar corresponds to the value obtained by *Relaxed*, whereas the top corresponds to the value obtained by *Safe*.

Optimality gap for various instances

Figure 5.6 shows the achieved gaps for different values of ε under setting 2. Likewise, we also plotted the values of the lower and upper bounds obtained for each instance, in Figure 5.7

We observe the same improvement as in Section 5.5.2 from *Bonferroni* to *Safe* (approximately two orders of magnitude better). Now, the achieved gap for *Bonferroni* grows quickly with ε, and for $\alpha = 2, 3$, goes substantially above 50% as from a tolerated budget of 0.2, and 0.1, respectively.

Thus, we can see that when it is profitable to trade strict feasibility for objective improvement, *Safe* is able to perform considerably better than the *Bonferroni*. For $\alpha = 2$, the gap between the allocation of *Safe* and the lower bound given by *Relaxed* does not exceed 10%. For $\alpha = 3$, it is within the same order of magnitude, although some realizations might go up to 60%.

In Figure 5.7, the lower and upper bounds on the true optimum are shown in the same fashion as for setting 1 to illustrate in absolute values

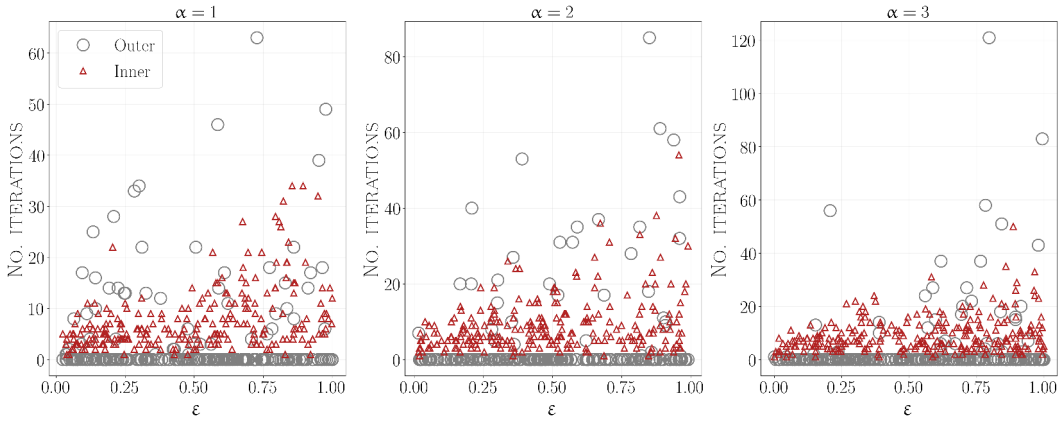


Figure 5.8 | Achieved number of inner and outer iterations by *Safe* during polishing under setting 2.

the results of Figure 5.6.

Number of convex problems to solve

Just as for setting 1, we represented in Figure 5.8 the number of inner and outer iterations achieved with the Polish sub-routine by *Safe*. We can observe that the large majority of instances led to a safe rounded scenario vector as output of the rounding sub-routine. This fact is encouraging as an safe rounded scenario leads to an instantly available safe allocation, whereas it is more problematic to polish an unsafe scenario, especially for some cases, for $\alpha = 3$, where hundreds of outer iterations are carried. This observation can also be made for setting 1.

All in all, this analysis leads us to the conclusion, although this situation occurs in practice quite rarely, that it might be of interest to integrate some additional constraints in the rounding sub-routine so that it outputs at all times (almost) safe scenarios.

Lastly, we give an equivalent illustration to Figure 5.5, of an evolution of the values for *Safe*, *Bonferroni*, and *Relaxed*, on ten fixed instances for a varying risk ε. Likewise, the specifics are shown with translucent markers whereas we build a solid curve with empty markers to show the average evolution. The same conclusions as in setting 1 apply.

5.6 Concluding remarks

The α-fair resource allocation problem was extended to take account of the varying remaining available bandwidths over the links of the network. To tackle this setting, we introduced an extension of the model in the form of individually chance constrained programming with variable individual tolerance risks and formalized the problem with introduced binary variables that activate the different possible scenarios. Due to the binary variables, the problem is in fact highly intractable, and a heuristic, *Safe*, was designed.

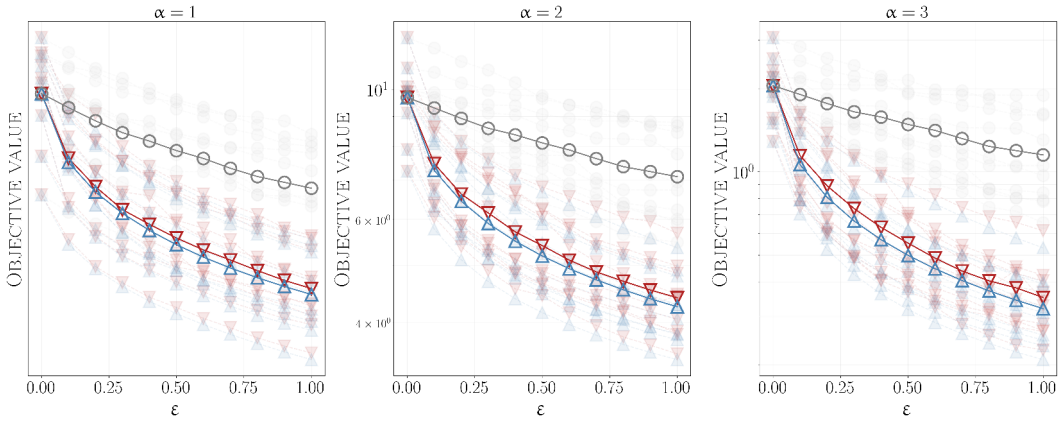


Figure 5.9 | The fairness value for *Safe* (red), *Relaxed* (blue) and *Bonferroni* (gray) against the risk ε under setting 2.

Safe was based on the relaxation of the binary program and a polishing subroutine that finds a scenario improving greedily, first safety and then optimality, with the help of the sensitivity of the (classic) problem with respect to the capacity constraints. It is inspired from the greedy heuristic for the maximization of submodular functions, with the additional feature that the marginal values are only estimated through dual variables and not through explicit evaluation, which permit to perform quicker (possibly poorer) iterations.

We compared *Safe* with *Bonferroni*, a classic approach with fixed individual tolerated violation risks split equally among the constraints, over Fat-Tree type networks, and showed by simulation that the performance was two orders of magnitude better, in terms of (estimated) optimality gap. Moreover, with the help of the relaxed solution, we showed that together, *Relaxed* and *Safe*, permit to bound satisfactorily the optimal value of the mixed binary problem.

Conclusions and perspectives

Summary

With the rise of SDN technologies that can perform network management while globally optimizing resources, the research on network resource allocation has regained momentum. In this dissertation, we mainly studied an important instance of the network utility maximization problem, called the α-fair resource allocation problem. Motivated by the practical challenges of real-time resource allocation in distributed SDNs, we provided answers to the problems of transient infeasibility that can cause service degradation in terms of congestion, packet loss, and common undesired consequences of capacity violation. We also tailored our algorithms to a SDN distributed architecture, by ensuring locality of the system parameters and variables, while guaranteeing convergence in a distributed implementation where communication costs between sub-controllers come at a non-trivial cost in terms of delay.

In Chapter 2, the fair resource allocation problem was introduced formally, and we studied the α-fair allocation structure in order to derive a minimal value on the optimally fair bandwidth allocation, which was compared to the only other bound we could find in the literature. Our bound also proves useful to optimally tune our main algorithm for resource allocation in Chapter 4.

In Chapter 3, we reviewed the literature on algorithms that were designed to compute the α-fair resource allocation problem, and discovered that none of them could address simultaneously our three main requirements (defined in Section 1.3). Based on the desirable versatility and distributive properties of ADMM, we designed FD-ADMM to achieve our goals. Indeed, FD-ADMM distributes across any domain distribution of the SDN network and operates parallel projections on a link basis, thus requires no globally shared information, except the for one unique consensus point. Also, it reacts to changes in the traffic requirements, modeled by variations of the problem's coefficients on-the-fly, while always preserving feasibility and performing an acceptable optimality gap.

In Chapter 4, we considered practical refinements of FD-ADMM, starting with its tuning. Indeed, we discovered that the performance of FD-ADMM was highly dependent of the penalty parameter λ introduced in the augmented Lagrangian of the problem. We therefore summoned the results

of Chapter 2 that, other than giving insights on the structural properties of fairness, permitted us to derive a theoretically "optimal" penalty parameter to use in the centralized version C-ADMM, that happened to provide near-optimal convergence rate of the residuals to 0 for FD-ADMM. Further, in section 4.3, we presented an extended version of the model to the multi-path setting where requests can have their traffic split among several connection paths, as well as the related relevant problem of limiting flow reconfiguration by incorporating switching costs. In both cases, we presented in detail how to modify the algorithm. Also, we gave the general method to incorporate any kind of regularization term translating a special structure preference on the fair solution.

In Chapter 5, FD-ADMM was considered as a by-product able to solve instance of the α-fair resource allocation problem efficiently. We introduced the relevant situation where the network contains another traffic class over which it has no control, and the resource allocation has to be chosen over an unknown but estimable (via a probability density) remaining available bandwidth on each link. To model this situation, we adopted the chance constraints programming view and proposed a heuristic that trades off feasibility with efficiency while respecting an overall fixed budget of violation risk. The heuristic takes the solution of an outer approximation of the exact problem and improves it greedily by evaluating the sensitivity of the α-fair problem, and thus mimics the well-known greedy heuristic that maximizes a function with diminishing marginal values.

Perspectives

The starting model responds to the problem of allocating bandwidth over pre-established paths. A first natural extension could therefore take into account the traffic engineering problem, that is, to compute paths *and* their respective bandwidth allocation altogether, along with additional constraints such as delay, path length, or maximal number of paths for each request in a multi-path setting.

We would like to provide a definitive answer to the question posed in Chapter 2 about the bound improvement, although by experimentation, we found no counterexample at all. Also, the performance of the penalty parameter derived following the bound improvement of Chapter 2 in FD-ADMM is still a phenomenon that we do not understand fully. A first natural continuation of the works in this dissertation could be this topic.

Concerning Chapter 5, the model solved by *Safe* could be restructured to avoid the summoning of an inner iterative loop (such as FD-ADMM) at each iteration. We argued that an algorithm such as *Safe* should apply more at the end of some monitoring phase, thus not in real-time, as the building of an accurate estimation of the available bandwidth takes some time. Therefore, it is acceptable to use algorithms with less stressful time requirements. Nonetheless, algorithms with more efficient (particularly, without inner iterative processes) update rules are always more desirable. Thus, we

are interested in re-formulating Problem (5.7)– (5.13) in the general form of ADMM, in order to solve it with the framework. Since the problem now includes binary variables, it will not have the same convergence guarantees as C-ADMM of FD-ADMM. Also, such an algorithm would require Euclidean projections on discrete sets, which will require the development of specific heuristics with strict performance guarantees. The application of ADMM to non-convex problems is documented in the very recent literature (NC(Non Convex)-ADMM, [78]) and specifically requires careful study of the projection and proximal subroutines in order to be able to (i) guarantee the stabilization of the algorithms and (ii) provide estimates on the fitness of the solutions. In general, the guarantees of NC-ADMM is still, to the best of our knowledge, an area of active research.

Also, Chapter 5 tackles the problem of controlling the probability of violation of the constraints; however, it does not control explicitly the amount of violated resources. We are interested in providing answers to this question by including additional control features to *Safe* such as Quality of Experience for video applications in SDN [79, 80].

List of publications

1. Allybokus, Zaid, Konstantin E Avrachenkov, Jeremie Leguay and Lorenzo Maggi. "Real-Time Fair Resource Allocation in Distributed Software Defined Networks." 2017 29th International Teletraffic Congress (ITC 29) 1 (2017): 19-27. (Best Paper Award)

2. Allybokus, Zaid, Konstantin E Avrachenkov, Jeremie Leguay and Lorenzo Maggi. "Lower Bounds for the Fair Resource Allocation Problem." SIGMETRICS Performance Evaluation Review 45 (2017): 167-173.

3. Allybokus, Zaid, Konstantin E Avrachenkov, Jeremie Leguay and Lorenzo Maggi. "Multi-Path Alpha-Fair Resource Allocation at Scale in Distributed Software-Defined Networks." IEEE Journal on Selected Areas in Communications 36 (2018): 2655-2666.

Bibliography

[1] Diego Kreutz et al. "Software-defined networking: A comprehensive survey". In: *Proc. of the IEEE* 103.1 (2015), pp. 14–76.

[2] Mohammad Mousa, Ayman M Bahaa-Eldin, and Mohamed Sobh. "Software Defined Networking concepts and challenges". In: *2016 11th International Conference on Computer Engineering & Systems (ICCES)*. IEEE. 2016, pp. 79–90.

[3] Hyojoon Kim and Nick Feamster. "Improving network management with software defined networking". In: *IEEE Communications Magazine* 51.2 (2013), pp. 114–119.

[4] Sushant Jain et al. "B4: Experience with a globally-deployed software defined WAN". In: *ACM SIGCOMM Computer Communication Review*. Vol. 43. 4. ACM. 2013, pp. 3–14.

[5] Kok-Kiong Yap et al. "Taking the edge off with espresso: Scale, reliability and programmability for global internet peering". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM. 2017, pp. 432–445.

[6] Steven J Vaughan-Nichols. "OpenFlow: The next generation of the network?" In: *Computer* 44.8 (2011), pp. 13–15.

[7] Sandra Scott-Hayward. "Design and deployment of secure, robust, and resilient SDN Controllers". In: *Proceedings of the 2015 1st IEEE conference on network Softwarization (NetSoft)*. IEEE. 2015, pp. 1–5.

[8] William Stallings. "Software-defined networks and openflow". In: *The internet protocol Journal* 16.1 (2013), pp. 2–14.

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

[10] Fouad Benamrane, Mouad Ben Mamoun, and Benaini Redouane. "An East-West interface for distributed SDN control plane: Implementation and evaluation". In: *Computers & Electrical Engineering* 57 (Sept. 2016).

[11] Jun Huy Lam et al. "TLS Channel Implementation for ONOS's East/West-Bound Communication". In: *Electronics, Communications and Networks V*. Ed. by Amir Hussain. Singapore: Springer Singapore, 2016, pp. 397–403. ISBN: 978-981-10-0740-8.

[12] Pingping Lin, Jun Bi, and Yangyang Wang. "East-West bridge for SDN network peering". In: vol. 401. Jan. 2013, pp. 170–181.

[13] Tsung-Hui Chang et al. "Asynchronous distributed ADMM for large-scale optimization—Part I: Algorithm and convergence analysis". In: *IEEE Transactions on Signal Processing* (2016).

[14] Kévin Phemius, Mathieu Bouet, and Jérémie Leguay. "Disco: Distributed multi-domain SDN controllers". In: *Proc. IEEE NOMS*. 2014, pp. 1–4.

[15] Mathis Obadia et al. "Failover mechanisms for distributed SDN controllers". In: *NOF*. IEEE. 2014.

[16] Othmane Blial, Mouad Ben Mamoun, and Redouane Benaini. "An overview on SDN architectures with multiple controllers". In: *Journal of Computer Networks and Communications* 2016 (2016).

[17] *Akka*. URL: `https://akka.io/`.

[18] Jisoo Shin et al. "IRIS-HiSA: Highly Scalable and Available Carrier-Grade SDN Controller Cluster". In: *Mobile Networks and Applications* (2017).

[19] *OpenDayLight*. URL: `https://www.opendaylight.org/`.

[20] *The ONOS Project*. URL: `https://onosproject.org`.

[21] Udayasree Palle et al. *PCEP Extensions for MPLS-TE LSP Automatic Bandwidth Adjustment with Stateful PCE*. Internet-Draft draft-dhody-pce-stateful-pce-auto-bandwidth-09. Work in Progress. Internet Engineering Task Force, Nov. 2016. 25 pp.

[22] Bill McCormick et al. "Real time alpha-fairness based traffic engineering". In: *Proc. of ACM HotSDN*. 2014, pp. 199–200.

[23] Stephen Boyd et al. "Distributed optimization and statistical learning via the alternating direction method of multipliers". In: *Foundations and Trends in Machine Learning* 3.1 (2011), pp. 1–122.

[24] Anees Al-Najjar et al. "Link capacity estimation in SDN-based end-hosts". In: *2016 10th International Conference on Signal Processing and Communication Systems (ICSPCS)*. IEEE. 2016, pp. 1–8.

[25] Eitan Altman, Konstantin Avrachenkov, and Andrey Garnaev. "Generalized α-fair resource allocation in wireless networks". In: *2008 47th IEEE Conference on Decision and Control*. IEEE. 2008, pp. 2414–2419.

[26] Mikael Johansson and Lin Xiao. "Scheduling, routing and power allocation for fairness in wireless networks". In: *2004 IEEE 59th Vehicular Technology Conference. VTC 2004-Spring (IEEE Cat. No. 04CH37514)*. Vol. 3. IEEE. 2004, pp. 1355–1360.

[27] Aizaz U Chaudhry, Nazia Ahmad, and Roshdy HM Hafez. "Improving throughput and fairness by improved channel assignment using topology control based on power control for multi-radio multi-channel wireless mesh networks". In: *EURASIP Journal on Wireless Communications and Networking* 2012.1 (2012), p. 155.

[28] Abhinay Muthoo. *Bargaining theory with applications*. Cambridge University Press, 1999.

[29] Ao Tang, Jiantao Wang, and Steven H Low. "Is fair allocation always inefficient". In: *IEEE INFOCOM 2004*. Vol. 1. IEEE. 2004.

[30] Ken Binmore, Ariel Rubinstein, and Asher Wolinsky. "The Nash bargaining solution in economic modelling". In: *The RAND Journal of Economics* (1986), pp. 176–188.

[31] John F Nash. "The Bargaining Problem". In: *Econometrica* 18.2 (1950), pp. 155–162.

[32] Geoffroy de Clippel. "An axiomatization of the Nash bargaining solution". In: *Social Choice and Welfare* 29.2 (2007), pp. 201–210. ISSN: 1432-217X.

[33] T. Lan et al. "An Axiomatic Theory of Fairness in Network Resource Allocation". In: *2010 Proceedings IEEE INFOCOM*. 2010, pp. 1–9.

[34] Jeonghoon Mo and Jean Walrand. "Fair end-to-end window-based congestion control". In: *IEEE/ACM Transactions on Networking (ToN)* 8.5 (2000), pp. 556–567.

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

[36] Frank P Kelly, Aman K Maulloo, and David KH Tan. "Rate control for communication networks: shadow prices, proportional fairness and stability". In: *Journal of the Operational Research society* 49.3 (1998), pp. 237–252.

[37] Fernando Paganini et al. "Network stability under alpha fair bandwidth allocation with general file size distribution". In: *IEEE Transactions on Automatic Control* 57.3 (2012), pp. 579–591.

[38] Jelena Marasevic, Clifford Stein, and Gil Zussman. "A Fast Distributed Stateless Algorithm for alpha-Fair Packing Problems". In: *Proc. of ICALP), vol.55, pp.54–1*. 2016.

[39] Réka Albert and Albert-László Barabási. "Statistical mechanics of complex networks". In: *Rev. Mod. Phys.* 74 (1 2002), pp. 47–97.

[40] Eitan Altman, Konstantin Avrachenkov, and Sreenath Ramanath. "Multiscale fairness and its application to resource allocation in wireless networks". In: *Computer Communications* 35.7 (2012), pp. 820–828.

[41] Thomas Voice. "Stability of multi-path dual congestion control algorithms". In: *Proc. of Valuetools*. ACM. 2006.

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

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

[44] Daniel Pérez Palomar and Mung Chiang. "A tutorial on decomposition methods for network utility maximization". In: *IEEE Journal on Selected Areas in Communications* 24.8 (2006), pp. 1439–1451.

[45] Tae-Jin Lee and G. De Veciana. "A decentralized framework to achieve max-min fair bandwidth allocation for ATM networks". In: *IEEE GLOBECOM 1998*. 1998, 1515–1520 vol.3.

[46] Rajesh Sundaresan et al. "An Iterative Interior Point Network Utility Maximization Algorithm". In: *arXiv preprint arXiv:1609.03194* (2016).

[47] Bingsheng He and Xiaoming Yuan. "On the O(1/N) Convergence Rate of the Douglas-Rachford Alternating Direction Method". In: *SIAM J. Numer. Anal.* 50.2 (Apr. 2012), pp. 700–709. ISSN: 0036-1429.

[48] Wei Deng and Wotao Yin. "On the Global and Linear Convergence of the Generalized Alternating Direction Method of Multipliers". In: *Journal of Scientific Computing* 66.3 (2016), pp. 889–916.

[49] João FC Mota et al. "Distributed ADMM for model predictive control and congestion control". In: *Proc. of IEEE CDC*. 2012.

[50] C. Liang and F. R. Yu. "Distributed resource allocation in virtualized wireless cellular networks based on ADMM". In: *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2015, pp. 360–365.

[51] Alfredo N Iusem and Alvaro R De Pierro. "A simultaneous iterative method for computing projections on polyhedra". In: *SIAM Journal on Control and Optimization* 25.1 (1987), pp. 231–243.

[52] Yunmei Chen and Xiaojing Ye. "Projection onto a simplex". In: *arXiv preprint arXiv:1101.6081* (2011).

[53] Bo Wahlberg et al. "An ADMM Algorithm for a Class of Total Variation Regularized Estimation Problems". In: *IFAC Proceedings Volumes* 45.16 (2012). 16th IFAC Symposium on System Identification, pp. 83 –88. ISSN: 1474-6670.

[54] Caoxie Zhang, Honglak Lee, and Kang Shin. "Efficient distributed linear classification algorithms via the alternating direction method of multipliers". In: *Artificial Intelligence and Statistics*. 2012, pp. 1398–1406.

[55] João FC Mota et al. "Basis pursuit in sensor networks". In: *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2011, pp. 2916–2919.

[56] Donald Goldfarb, Shiqian Ma, and Katya Scheinberg. "Fast alternating linearization methods for minimizing the sum of two convex functions". In: *Mathematical Programming* 141.1-2 (2013), pp. 349–382.

[57] Damek Davis and Wotao Yin. "Faster convergence rates of relaxed Peaceman-Rachford and ADMM under regularity assumptions". In: *Mathematics of Operations Research* 42.3 (2017), pp. 783–805.

[58] BS He, Hai Yang, and SL Wang. "Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities". In: *Journal of Optimization Theory and applications* 106.2 (2000), pp. 337–356.

[59] Stefano Paris et al. "Controlling flow reconfigurations in SDN". In: *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE. 2016, pp. 1–9.

[60] Francis Bach et al. "Optimization with sparsity-inducing penalties". In: *Foundations and Trends® in Machine Learning* (2012).

[61] Tsung-Feng Yu, Kuochen Wang, and Yi-Huai Hsu. "Adaptive routing for video streaming with QoS support over SDN networks". In: *2015 International Conference on Information Networking (ICOIN)*. IEEE. 2015, pp. 318–323.

[62] Sangwook Bae, Dahyun Jang, and KyoungSoo Park. "Why is http adaptive streaming so hard?" In: *Proceedings of the 6th Asia-Pacific Workshop on Systems*. ACM. 2015, p. 12.

[63] Jiri Navratil and R Les Cottrell. "ABwE: A practical approach to available bandwidth estimation". In: *Proc. passive and active measurement workshop*. 2003.

[64] Emanuele Goldoni, Giuseppe Rossi, and Alberto Torelli. "Assolo, a new method for available bandwidth estimation". In: *2009 Fourth International Conference on Internet Monitoring and Protection*. IEEE. 2009, pp. 130–136.

[65] Manish Jain and Constantinos Dovrolis. "Pathload: A measurement tool for end-to-end available bandwidth". In: *In Proceedings of Passive and Active Measurements (PAM) Workshop*. Citeseer. 2002.

[66] Vinay Joseph Ribeiro et al. "pathchirp: Efficient available bandwidth estimation for network paths". In: *Passive and active measurement workshop*. 2003.

[67] Salcedo Morillo et al. "Overhead in available bandwidth estimation tools: Evaluation and analysis". In: (2017).

[68] Giuseppe Aceto et al. "Sometime: Software defined network-based available bandwidth measurement in monroe". In: *2017 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE. 2017, pp. 1–6.

[69] Péter Megyesi et al. "Available bandwidth measurement in software defined networks". In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. ACM. 2016, pp. 651–657.

[70] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. "An analysis of approximations for maximizing submodular set functions—I". In: *Mathematical programming* 14.1 (1978), pp. 265–294.

[71] Maxim Sviridenko. "A note on maximizing a submodular set function subject to a knapsack constraint". In: *Operations Research Letters* 32.1 (2004), pp. 41–43.

[72] Michel Minoux. "Accelerated greedy algorithms for maximizing submodular set functions". In: *Optimization techniques*. Springer, 1978, pp. 234–243.

[73] Arkadi Nemirovski and Alexander Shapiro. "Convex approximations of chance constrained programs". In: *SIAM Journal on Optimization* 17.4 (2006), pp. 969–996.

[74] James Luedtke, Shabbir Ahmed, and George L Nemhauser. "An integer programming approach for linear programs with probabilistic constraints". In: *Mathematical programming* 122.2 (2010), pp. 247–272.

[75] Feng Qiu et al. "Covering linear programming with violations". In: *INFORMS Journal on Computing* 26.3 (2014), pp. 531–546.

[76] Charles E Leiserson. "Fat-trees: universal networks for hardware-efficient supercomputing". In: *IEEE transactions on Computers* 100.10 (1985), pp. 892–901.

[77] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. "A scalable, commodity data center network architecture". In: *ACM SIGCOMM Computer Communication Review*. Vol. 38. 4. ACM. 2008, pp. 63–74.

[78] Steven Diamond, Reza Takapoui, and S Boyd. "A general system for heuristic minimization of convex functions over non-convex sets". In: *Optimization Methods and Software* 33.1 (2018), pp. 165–193.

[79] René Serral-Gracià et al. "An overview of quality of experience measurement challenges for video applications in IP networks". In: *International Conference on Wired/Wireless Internet Communications*. Springer. 2010, pp. 252–263.

[80] Michael Jarschel et al. "SDN-based application-aware networking on the example of youtube video streaming". In: *2013 Second European Workshop on Software Defined Networks*. IEEE. 2013, pp. 87–92.