Efficacité des jeux de files d'attente distribués et des algorithmes de découverte de chemin

Josu Doncel

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)*

Présentée et soutenue le *30/03/2015* **par :**

Josu DONCEL

Efficiency of Distributed Queueing Games and of Path Discovery Algorithms

---

JURY

Germain GARCIA	Professeur d'Université	Président
Johanne COHEN	Chargée de Recherche	Rapporteur
Rachid EL-AZOUZI	Professeur d'Université	Rapporteur
Urtzi AYESTA	Chargé de Recherche	Examinateur
Bruno GAUJAL	Directeur de Recherche	Examinateur
Bruno TUFFIN	Directeur de Recherche	Examinateur
Olivier BRUN	Chargé de Recherche	Directeur de thèse
Balakrishna J. PRABHU	Chargé de Recherche	Directeur de thèse

École doctorale et spécialité :
EDSYS : Informatique 4200018
Unité de Recherche :
Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS-CNRS)
Directeur(s) de Thèse :
Olivier BRUN et *Balakrishna J. PRABHU*
Rapporteurs :
Johanne COHEN et *Rachid EL-AZOUZI*

To my parents
and to Idoia

Acknowledgments

In the beginning of 2012, I took the decision of moving to Toulouse in order to start a PhD in mathematical modeling of communication networks, supervised by Olivier Brun and Balakrishna J. Prabhu. I thought that doing a PhD with them was a great opportunity for me. From the first day, I have noticed the correctness of this belief. Their support and advices during my PhD have been very important for the development of the thesis. They have shown me many tools that I consider they will be very useful for my future research. I am very happy to have worked with Olivier and Bala these three years and I hope we can continue sharing time in front of the blackboard of their office.

I must say that Urtzi Ayesta has been also very important in this thesis. First, he introduced me to Olivier and Bala. Then, he has been very implicated in my work and we have had numerous (scientific and personal) discussions from which I have learned many things. I would like to thank him all the patience with me.

I wish to thank the members of the jury for accepting to participate in my PhD defence. I thank Rachid El-Azouzi and Johanne Cohen for reviewing my thesis manuscript and for their comments. I also acknowledge Bruno Gaujal and Bruno Tuffin for the time and the travel they have done to assist to the Ph.D. defense and Germain Garcia for taking the time to examine my work.

This thesis has been developed in the SARA group of the LAAS-CNRS laboratory. The ambience of this group has been perfect to do research, specially, in the office A45. I am grateful for the help that Christopher Thraves Caro provided me in the last part of my PhD. I also feel lucky for having collaborated with Josselin Vallet, Henda Ben Cheikh and Tatiana Seregina in different projects. Working together with them has showed me the importance of group work. I thank the organizators and participators of Evalperf working group and RC theme sessions, since the research discussions with them have been very fruitful. I wish to thank Martin Erauskin, Matthieu Jonckheere and the rest of the visitors of the group for their time to come to LAAS and the discussions in the laboratory and in the afterwork. I am grateful to Tom, the fifth member of the office A45, to Ahmad, to Cedric, to Ane, to Maialen and to the rest of the students of the group for all the discussions in the corridors of the laboratory and at lunch time. I would also like to acknowledge the projects that have founded my PhD, SOP and Panacea. It has been an honor to have worked in this context.

It is the moment to mention some people with whom I have lived many adventures

in the last three years. I am grateful to my friend Aloña who made me discover that *la ville rosse* it is a city that can be enjoyed a lot. Also, I would like to thank the people that have become "mi familia tulusana": Grego, Bartu & Jime, Quintero, Oswaldo, Isidre, Alberto, Isma & Rakel, Marti, Nuria & Roger, Paqui & Dani, David & Laura, Remi... I would like also thank all the students and teachers of the Basque courses of "Denak Bat" association: Olaia, Maddi, Aitzi, Cyrielle, Yoann, Jon, Alice, Romain... All the activities we have prepared together helped me to discover new issues about the basque culture. I am also grateful to my friends of my birth place, because they have always received me with a smile, even if I live on the other side of the Pyrenees. Finally, I would like to thank to Philippe and Marie Claire, as well as the other members of the Derrier le Boulevard group, since they showed me that sometimes it is better to finish in the second place.

The decision of starting this Ph.D. has been taken together with Idoia Garaizabal. Even though her research is far from the subject of this thesis, I can claim that she is the co-author of this thesis. *Pre, eskerrik asko laguntza guztiagatik. Nire ondoan sentitu zaitut momentu oro eta inor baino gehiago bultzatu didazu tesi hau bukatzeko. Lan bikaina egiten ari zara postdoc bat lortzeko eta espero dut laster aurkitzea postu bat. Zuk mila bider esan bezala, "Animo, mi amor!".*

To finish, I would like to say a few words about Iñaki and Mari Jose, i.e. my parents. *Aita y ama, aunque no entendais mucho de lo que se trata investigación ni del tema de esta tesis, os habeis interesado mucho en mi trabajo y me habeis apoyado durante estos tres años. Os agradezco mucho todo eso. Quiero que sepais que sois modelo y referencia para mi y que vuestros consejos los considero muy valiosos.* I also thank you to the rest of the members of my family for their support and, specially, to Martina, my niece, since sharing time with her makes me feel as a happy as a child.

That was just the beginning of the thesis.

Abstract - Résumé

Abstract:

This thesis deals with the efficiency of distributed resource sharing algorithms and of online path discovery algorithms. In the first part of the thesis, we analyze a game in which users pay for using a shared resource. The allocated resource to a user is directly proportional to its payment. Each user wants to minimize its payment while ensuring a certain quality of service. This problem is modelled as a non-cooperative resource-sharing game. Due to lack of analytical expressions for the underlying queuing discipline, we are able to give the solution of the game only under some assumptions. For the general case, we develop an approximation based on a heavy-traffic result and we validate the accuracy of the approximation numerically.

In the second part, we study the efficiency of load balancing games, i.e., we compare the loss in performance of noncooperative decentralized routing with a centralized routing. We show that the PoA is very pessimistic measure since it is achieved in only pathological cases. In most scenarios, distributed implementations of load-balancing perform nearly as well as the optimal centralized implementation.

In the last part of the thesis, we analyze the optimal path discovery problem in complete graphs. In this problem, the values of the edges are unknown but can be queried. For a given function that is applied to paths, the goal is to find a best value path from a source to a given destination querying the least number of edges. We propose the query ratio as efficiency measure of algorithms that solve this problem. We prove a lower-bound for any algorithm that solves this problem and we proposed an algorithm with query ratio strictly less than 2.

Résumé:

Cette thèse porte sur l'efficacité des algorithmes distribués de partage des ressources et des algorithmes de découvert de chemin en ligne. Dans la première partie de la thèse, nous analysons un jeu dans lequel les utilisateurs paient pour utiliser une ressource partagée. La ressource allouée à un utilisateur est directement proportionnel à son paiement. Chaque utilisateur veut minimiser son paiement en assurant une certaine qualité de service. Ce problème est modélisé comme un jeu non-coopératif de partage des ressources. A cause du manque des expressions analytiques de la discipline de file

d'attente sous-jacente, nous pouvons résoudre le jeu que sous certaines hypothèses. Pour le cas général, nous développons une approximation basée sur un résultat fort trafic et nous validons la précision de l'approximation numériquement.

Dans la deuxième partie, nous étudions l'efficacité des jeux de balance de charge, c'est à dire, nous comparons la perte de performance de routage non coopératif décentralisé avec un routage centralisé. Nous montrons que le PoA est une mesure très pessimiste car il est atteint que dans des cas pathologiques. Dans la plupart des scénarios, les implémentations distribués de balance de charge effectuent presque aussi bien que la mise en œuvre centralisée optimale.

Dans la dernière partie de la thèse, nous analysons problème de découverte chemin optimal dans les graphes complets. En ce problème, les valeurs des arêtes sont inconnues, mais peuvent être interrogés. Pour une fonction donnée qui est appliquée à des chemins, l'objectif est de trouver un meilleur chemin de valeur à partir d'une source vers une destination donnée interrogation le plus petit nombre de bords. Nous vous proposons le rapport de requête en tant que mesure de l'efficacité des algorithmes qui permettent de résoudre ce problème. Nous prouvons une limite inférieure pour ne importe quel algorithme qui résout ce problème et nous avons proposé un algorithme avec un rapport de requête strictement inférieure à 2.

Contents

List of Figures

List of Tables

1 Introduction

In modern telecommunication networks, resources need to be shared among users. A resource in these networks can be, for example, the processing capacity of a computer system, the power of wireless transmitters or the bandwidth of communication paths. The way that users share the resource defines the performance of the system. The users can share the resources in a way that the performance of the communication system is optimum, whereas as inappropriate resource sharing can lead to significant performance degradations.

In this thesis, we analyze the efficiency of telecommunication networks. In particular, the question we aim to answer is how the resources of each component of the network are allocated and how close to optimal is the performance that is derived from this resource sharing. This is considered an important problem in networking and many researchers have been working in this area for years. The most important tools that are applied to investigate the efficiency of telecommunication networks are queueing theory, game theory and graph theory.

Arrivals and service requirements of users in telecommunication systems are random. Hence, *queueing theory*, the set of probabilistic techniques that study waiting lines or queues, is a fundamental tool to analyze these systems. Queueing models are used to study the processing times, waiting times and number of jobs in the queues, under some assumptions regarding the jobs arrivals and service requirements.

Game theory studies the strategic decision making of users that behave rationally. If we consider that agents of queueing systems are rational and can take autonomous decisions, the techniques of game theory can be applied to those queueing systems.

These games are known in the literature as queueing games. The books [50] and [68] as well as the survey [10] present many interesting queueing games. We focus on distributed queueing games, where each player is selfish and performs individual optimisation for its own jobs. These types of games are also known in the literature as non-cooperative queueing games or decentralized queueing-games.

A classical tool to study the properties of networks is *graph theory*. A graph consists of a set of nodes that are connected with edges. Two computers that communicate using Internet can be modelled as two nodes connected by an edge. Thus, communication networks can be modelled as a graph. The analysis of telecommunication networks using graphs has a long history. There is a lot of research in graph optimisation and many algorithms have been proposed in the literature as a solution for efficient search problems.

Our work is at the intersection of the three above mentioned scientific disciplines. Furthermore, the obtained results can be classified into two different categories. For the first category, queueing theory and game theory are needed to analyze distributed queueing games. First, we analyze the users' competition for the capacity of a single server (see Chapter 3). Then, we study the efficiency of non-cooperative load balancing when several users compete for the capacity of several servers (see Chapter 4). The research of this part has been funded by the SOP project of the ANR [5]. On the other hand, we use graph theory to address the problem of optimising a network where the values of the edges are initially unknown (see Chapter 5). This work has been done in the framework of a European project called PANACEA [4].

We first describe briefly the problems we analyze. Then, we present the structure that follows this thesis.

1.1 Efficiency in Telecommunication Networks

1.1.1 Competition in a Single Server

We analyze the strategic decision making of users that compete for the capacity of a server. When a server gives equal priority to the users, all of them are served at the same rate. Since the server has a fixed capacity, if the number of users that want to get service in this server increases, they all get a slow service. However, a user may pay for a higher priority in the queue and thus obtain a faster service. In the model we study, each user is able to pay to increase its priority in order to get a faster service. The goal of each user is to minimize its payment while ensuring a certain Quality of Service (QoS). Hence, there is a competition among the users for the capacity of the server that can be analyzed under the framework of non-cooperative game theory.

The QoS constraint of this model is very complicated to analyze in a general setting. Thus, the derived game can be solved in closed-form only for particular cases. Nevertheless, we can approximate the original game by a more tractable game under

the assumption that the load of the system is close to one. We solve the latter game in the general case and we present how this result allows us to give an approximation of the original game.

1.1.2 Non-Cooperative Load Balancing

In the second work, we compare the performance of different load balancing architectures in server farms. First, we consider a centralized architecture where the incoming traffic arrives to a single dispatcher, that aims to minimize the mean response time of the jobs in the system. On the other hand, we study a system where there are more than one dispatcher and each of them receives a portion of the total incoming jobs. We assume that these dispatchers are selfish and seek to minimize the mean response time of their own jobs. This system is known as non-cooperative load balancing, selfish routing or decentralized architecture. For the decentralized architecture, a non-cooperative game can be formulated where each dispatchers is a player that balances the load so as to minimize the mean response time of its own jobs.

The objective is to quantify the loss in performance of the non-cooperative load balancing with respect to the centralized architecture. This comparison has been previously carried out using the so-called Price of Anarchy, an oft-used worst-case measure of the inefficiency of non-cooperative decentralized architectures [62]. For the game under consideration, we show that the Price of Anarchy is an overly pessimistic measure of performance and propose a new measure, called the inefficiency. Using this new measure, we show that non-cooperative load balancing is efficient in most cases, and becomes inefficient only in pathological cases.

1.1.3 Path Discovery Algorithms

In the third part, we analyze algorithms that search for the optimal path between two given nodes. We assume that the values of the edges of the graph are initially unknown, but can be discovered by querying an oracle. We present the Optimal Path Discovery problem. An algorithm that solves this problem seeks to minimize the number of queried edges for discovering an optimal path between two given nodes.

In the literature, the number of queries is the common measure of the performance of algorithms that solve this type of problems. We propose as a new measure the query ratio, that is the ratio between the number of queried edges and the smallest number of edges required to solve the problem. We thus analyze the query ratio of algorithms that solve this problem in complete graphs, and propose an algorithm with a query ratio upper bounded by 2.

1.2 Structure of the Thesis

The problems we analyze are divided in two parts. In Chapter 3 and Chapter 4, we investigate the properties of non-cooperative games that arise from the selfish behaviour of agents in communication networks. In Chapter 5, we study algorithms that optimize the performance of a network where the value of the edges are initially unknown.

Let us explain more in detail the content of each chapter.

- In Chapter 2, we introduce the main concepts we will use in the problems we analyze. First, we present the basic notions of queueing theory. We describe the single server model and then we explain the multi-server models, where there are more than one server. We also show how non-cooperative games can be defined in queueing networks and we present the main definitions of distributed queueing games. Finally, we give a summary of the graph theory concepts we require to analyze the problem addressed in Chapter 5.
- In Chapter 3, we analyze a non-cooperative game where a set of users compete for the capacity of a server, as described in Section 1.1.1.
- In Chapter 4, we investigate the performance of the non-cooperative load-balancing, as described in Section 1.1.2.
- In Chapter 5, we study algorithms that optimize the performance of a given source-destination pair when the value of the edges is initially unknown, as described in Section 1.1.3.
- In Chapter 6, we summarize the most relevant results of the models we analyze in this thesis. Besides, we give interesting issues that can be studied as future research.

The results of Chapter 3 have been published in [33]. The work presented in Chapter 4 was published in [32] and [31]. The work of Chapter 5 is described in [21], that has been submitted for publication.

2 Methodology

The objective of this chapter is to present the required methodologies in this thesis so as to make this manuscript is a self-contained document. As mentioned in the introduction, our work is at this intersection of three scientific disciplines: queueing theory, game theory and graph theory. We first introduce some basic results from queueing theory in Section 2.1. Section 2.2 is devoted to non-cooperative game theory. Finally, graph theoretic definitions required for our work on the Optimal Path Discovery problem are presented in Section 2.3.

2.1 Queueing Theory

2.1.1 Single Server Model

The single server model is widely studied in queueing theory [25, 46]. A single server system can be modelled as a queue, where the jobs arrive and wait in the queue until they are served. Once a job is served, it leaves the system. The performance of a single server is commonly measured by the sojourn time of jobs and the number of customers. The sojourn time of jobs is defined as the time between a job arrival to the system until its departure. The number of customers refers to the number of jobs in the system, either in the queue or getting service.

Since we refer to it several times, we first introduce a special type of random variables, the exponentially distributed one. We say that a random variable X has an exponential distribution of parameter β, if it verifies that $\mathbb{P}(X \leq x) = 1 - \exp(-\beta x)$ for $x \geq 0$, and $\mathbb{P}(X \leq x) = 0$ for $x < 0$. An important property of the exponentially

distributed random variables is that the value of their mean and second moment are known. If X is an exponential random variable of parameter β, we have that

$$\mathbb{E}(X) = \frac{1}{\beta},\ \mathbb{E}(X^2) = \frac{1}{\beta^2}.$$

There are four elements that characterize the behaviour of a queue: the incoming jobs process, the distribution of the service times, the buffer size and the scheduling policy.

In our work, we usually assume that the incoming jobs follow a Poisson process of rate λ. This means that the inter-arrival times of the jobs to the queue are independent and are exponentially distributed with parameter λ. In queueing theory, Poisson processes are widely used since they verify the memoryless property, which means that the number of job arrivals in a given time interval is independent of the number of arrived jobs in any other past non-overlapping interval. Thus, if we consider that arrival of jobs follows a Poisson process, the number of incoming jobs to the system from a given time onwards does not depend on the number of jobs arrived before this moment.

The service requirement of a job is defined as the time that it requires to be served. We sometimes say that the service requirements are exponentially distributed with parameter μ, which means that the time that jobs require to be served has an exponential distribution. A more general case will assume that the jobs are served according to distributions that are independent and identically distributed (i.i.d.) with mean $1/\mu$. We also refer to this case as generally distributed service time requirements.

The buffer size is defined as the maximum number of jobs that can be waiting in the queue. Hence, if the buffer size is K at most $K+1$ jobs can be present in the system. Besides, we say that the buffer size is infinite when there is no limitation on the number of jobs that can wait in the queue.

According to Kendall's notation, if the incoming traffic is Poisson and the buffer size is infinite, we say that a server is a $M/M/1/\infty$ queue when the service times are exponential, and a $M/G/1/\infty$ queue when the service time is general.

The scheduling policy characterizes how the jobs that are in the queue are served. Depending on the scheduling policy that is applied, the performance of the queue changes. Hence, it is very important for the performance of telecommunication networks to implement the right scheduling policies in the queues.

We say a scheduling policy is pre-emptive if the scheduler allows a process to be interrupted in the middle of its execution to serve other jobs. On the contrary, non pre-emptive scheduling ensures that the execution of jobs that are getting service is never stopped.

Many scheduling policies have been studied in the literature of the single server model. We mention some of them below and we refer to [91] for a recent survey of scheduling in single server queues.

- **First Come First Served**: is a non pre-emptive scheduling policy that serves jobs in order of arrival.
- **Least Come First Served**: is a non pre-emptive scheduling policy that serves jobs arrived the most recently.
- **Shortest-Remaining-Processing-Time**: is a pre-emptive discipline that assigns the whole capacity of the server to the customer with smallest remaining service time.
- **Random Order of Service**: is a non pre-emptively scheduling policy that serves a job chosen uniformly at random from the queue.
- **Discriminatory Random Order of Service**: is a multiclass generalization of the Random Order of Service discipline, where customers belonging to different classes have different selection probabilities [53, 57].
- **Processor Sharing** is a scheduling policy that serves all customers simultaneously at the same rate.
- **Generalized Processor Sharing**: is a multi-class generalization of the Processor Sharing model, that guarantees a minimum service rate to each class. When there are no jobs in one of the classes, its share of the capacity is distributed among the active classes [71].
- **Discriminatory Processor Sharing**: is a multi-class generalization of the Processor Sharing model, where all jobs present in the system are served simultaneously at rates that depend on the priorities of the classes g_i and the number of customers of each class.

Scheduling policies of a particular interest in this thesis are the Processor Sharing (PS) and Discriminatory Processor Sharing (DPS) disciplines. We remark that the main difference between them is that the PS queue shares the capacity equally among all the jobs, while in the DPS queue we can prioritize the service of some jobs. It is also important to note that the PS model is a particular case of the DPS queue that is achieved when all the weights are equal.

We present the performance obtained for the scheduling policy of PS and DPS in Figure 2.1 and Figure 2.2, respectively. In both cases, the input flow consists on four green jobs, two purple jobs and two red jobs. All the jobs have the same size. One job arrives to the queue when there is no job of the same color in the queue. Thus, at most, one job of each type is getting service. We observe that, in both figures, the input flow is the same and that the output flow is different. In Figure 2.1, all the jobs that are getting service share equally the capacity of the queue. Moreover, when the service of all purple and red jobs is finished, all the capacity of the server is for the green jobs. On

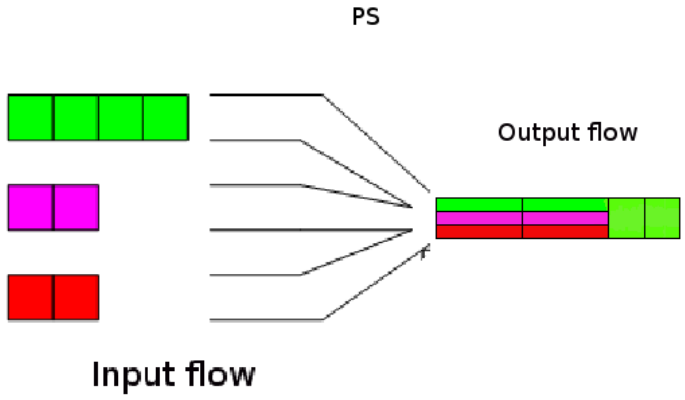

Figure 2.1: Scheduling in a server without priorities. PS queue case.

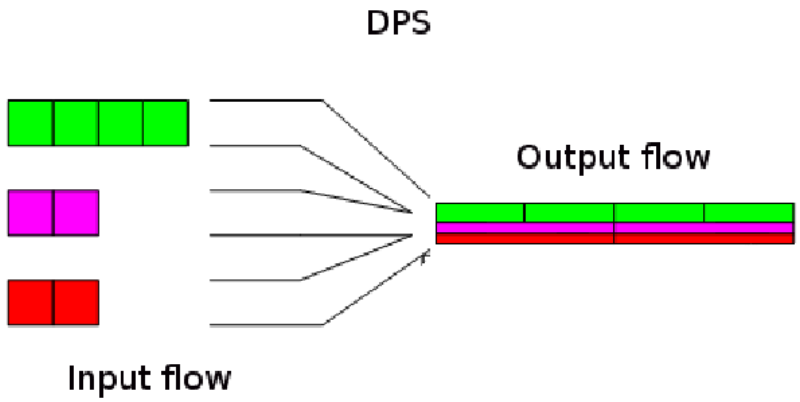

Figure 2.2: Scheduling in a server with priorities. DPS queue case.

the other hand, for the DPS queue, the green jobs have a bigger priority in the queue which means that they are served faster. Besides, red and purple jobs have the same priority and they are served at the same speed. As a consequence of the different speed of services, all the jobs finish at the same time.

We now turn to the expression of the mean response time for these scheduling policies. We denote by T_i the random variable representing the response times of jobs of class i in a PS queue. The expression of the mean response times in a PS queue, i.e., $\mathbb{E}(T_i)$, is well-known in the literature [48] and is given by

$$\mathbb{E}(T_i) = \frac{1/\mu}{1-\rho}, \tag{2.1}$$

where ρ is the total load in the system and μ is the service rate of the server.

We also present response time results of the DPS queue. We denote by $T_i(\mathbf{g};\rho)$ the random variable corresponding to the response time of a class-i job in a DPS queue for the vector of weights $\mathbf{g} = (g_1, \ldots, g_R)$ when the load in the system is $\rho < 1$. The mean response time is denoted by $\overline{T}_i(\mathbf{g};\rho) = \mathbb{E}(T_i(\mathbf{g};\rho))$. The authors in [34] prove that for exponential service time distributions, the mean response time is the solution of a system of equations.

Proposition 2.1.1 ([34]). *In the case of exponentially distributed required service times, the unconditional average response times satisfy the following linear system of equations:*

$$\overline{T}_k(\mathbf{g};\rho)\left(1 - \sum_{j=1}^{R} \frac{\lambda_j g_j}{\mu_j g_j + \mu_k g_k}\right) - \sum_{j=1}^{R} \frac{\lambda_j g_j \overline{T}_j(\mathbf{g};\rho)}{\mu_j g_j + \mu_k g_k} = \frac{1}{\mu_k}, \quad \text{with } k = 1, \ldots, R \tag{2.2}$$

A solution to this system of equations is only known for the case $R = 2$. In this case the solution is:

$$\overline{T}_1(\mathbf{g};\rho) = \frac{1}{\mu_1(1-\rho)}\left(1 + \frac{\mu_1\rho_2(g_2 - g_1)}{\mu_1 g_1(1-\rho_1) + \mu_2 g_2(1-\rho_2)}\right), \tag{2.3}$$

$$\overline{T}_2(\mathbf{g};\rho) = \frac{1}{\mu_2(1-\rho)}\left(1 + \frac{\mu_2\rho_1(g_1 - g_2)}{\mu_1 g_1(1-\rho_1) + \mu_2 g_2(1-\rho_2)}\right). \tag{2.4}$$

The response time of jobs in a DPS queue when the load is close to one has been also studied in the literature. Indeed, simple expressions are obtained for this case. In fact, in [93], the authors use this heavy-traffic result to approximate $T_i(\mathbf{g};\rho)$. The mentioned result reads:

Proposition 2.1.2 ([44]). *When scaled with $1-\rho$, the response time of class-i jobs has a proper distribution as $\rho \to 1$.*

$$(1-\rho)\; T_i(\mathbf{g};\rho) \xrightarrow{d} T_i(\mathbf{g};1) = X \cdot \frac{\mathbb{E}(B_i)}{g_i}, \quad i \in \mathcal{C}, \tag{2.5}$$

where $\xrightarrow{d}$ denotes convergence in distribution and X is an exponentially distributed random variable with mean

$$\mathbb{E}(X) = \frac{\sum_k \lambda_k \mathbb{E}\left(B_k^2\right)}{\sum_k \lambda_k \mathbb{E}\left(B_k^2\right)\frac{1}{g_k}}. \tag{2.6}$$

Proposition 2.1.2 implies that for sufficiently high load, the response time distribution in a DPS queue can be approximated by an exponential random variable, that is,

$$T_i(\mathbf{g};\rho) \approxeq \frac{T_i(\mathbf{g};1)}{1-\rho} \overset{d}{=} \frac{\mathbb{E}(B_i)}{g_i(1-\rho)}X, \tag{2.7}$$

and for the mean response time we obtain that

$$\overline{T}_i(\mathbf{g};\rho) \approx \frac{\mathbb{E}(B_i)}{g_i(1-\rho)} \frac{\sum_k \lambda_k \mathbb{E}\left(B_k^2\right)}{\sum_k \lambda_k \mathbb{E}\left(B_k^2\right) \frac{1}{g_k}}. \tag{2.8}$$

In the above derivation, we have ignored a technical subtlety. Indeed, in order for (2.8) to be valid, one needs to establish that the heavy-traffic limit and expectation can be interchanged, namely, $\lim_{\rho\to 1} \overline{T}_i(\mathbf{g};\rho) = \mathbb{E}(\lim_{\rho\to 1} T_i(\mathbf{g};\rho))$. In [88] the authors performed numerical experiments to validate the validity of this interchange. In Chapter 3 we will assume that the interchange is valid. In particular, for PS, it holds that $\overline{T}_i(\mathbf{g};\rho) = \mathbb{E}(B_i)/(1-\rho)$. Thus, from (2.5) and (2.6) we get $\overline{T}_i(\mathbf{g};1) = \mathbb{E}(B_i)$, and it follows that the approximation $\overline{T}_i(\mathbf{g};\rho) = \frac{\overline{T}_i(\mathbf{g};1)}{1-\rho}$ is exact.

2.1.2 The Multi-Server Model

Multi-server architectures have been extensively studied in the literature since they model a wide range of systems such as server-farms. In the multi-server model, a set of servers are in charge of giving service. Each server is modelled as a queue, using the notions presented in Section 2.1.1. An important role in the multi-server models is that of the routing agent or dispatcher, which controls the incoming jobs are sent to the servers. The objective of this agent is to optimize a certain metric of the system performance, for example the response times of incoming jobs.

The $M/G/c/K$ queue is a multi-server architecture with c servers, where the incoming jobs are Poisson, the buffer size is K and the service time requirements are general. In this model, at most $K+c$ jobs can be present in the system. If there are less than c jobs, some of the servers are idle. However, if the number of jobs is more than c, the jobs wait in a queue. The scheduling policy is the same for all the servers.

We can distinguish two broad categories of routing problems in multi-server systems. When the number of customers in all the queues is known, the dispatcher is faced with a routing problem in observable queues. In contrast, we say that the dispatcher solves a routing problem in non-observable queues when the number of customers in the servers is unknown for the routing agent.

In the observable case, there are different models that have been studied in the literature. We present the most important ones here:

- **Join the Shortest Queue**: the routing agent sends the flow to the queue with less number of customers.

- **Power of Two**: for all incoming jobs, the routing agent chooses $d \geq 2$ servers independently and uniformly at random and applies the Join the Shortest Queue policy to the chosen servers.

Routing in non-observable queues has been also widely analyzed. The Bernoulli routing is a routing policy that assigns a certain probability to each server (p_k) and sends the traffic to server k with probability p_k. This policy is simple to implement in multi-server systems. The Round-Robin scheduling sends the jobs to servers following a given order. This policy only requires the total number of servers and the last server where the last jobs has sent. Another option is that, instead of following a fixed order, the routing agent takes decisions on where to send incoming jobs,taking into account the previous routing action. The gain of performance caused by remembering the previous action is defined as Price of Forgetting [12].

Some scheduling policies for multi-server farms are specially designed for the case where the demand of incoming jobs is known. In [47, 27] the authors propose and analyze the Size Interval Task Assignment policy. In this model, each server gives service to jobs whose service demand is in a designated range.

In the multi-server architectures that we study, we consider that the load balancing is done using the Bernoulli splitting policy and the servers are PS queues.

2.2 Non-Cooperative Game Theory

In this section, we focus on the main concepts of non-cooperative games for the models we analyze. We first present a classical example for games that apply in telecommunication networks. Using this example, we revisit the main definitions of non-cooperative games. Finally, we describe best-response dynamics, which captures the evolution of player strategies when they behave in a selfish manner.

2.2.1 Pigou's Example

Alice has a message of size one and she wants to send it to Bob using Internet. The message is divided in packets of infinitesimal size. The Internet provider puts available only two routes. In the first route, the delay of each packet is equal to the traffic that traverses this route. The other route is slower and causes a delay of one second to each packet that traverses it. For example, if 30% of the packets use the fast route, then each of them will have a delay of 0.3 seconds and the delay of the rest of the packets is one second. Hence, the delay of the message is, in this case, 0.3x0.3+0.7x1.

We assume that each packet can choose to be transmitted by the fast or the slow route and aims to arrive to the destination with minimum delay. Each packet also knows the number of packets that use each route, which means that it can compute the delay obtained for both decisions. We can say that the packets are selfish and can take self-interested decisions. Hence, this situation that can be studied in the framework of non-cooperative games.

In Figure 2.3, we illustrate this problem which is known in the literature as *Pigou's*

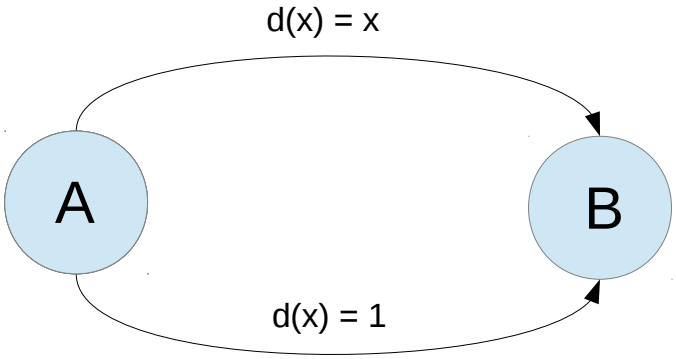

Figure 2.3: Pigou's example.

example. Node A represents the transmitter of the message, i.e., Alice and node B represents the receiver, i.e., Bob. The edge above in the fast route where the delay depends on the number of packets it traverses and the edge below represents the slow route.

2.2.2 Non-cooperative Games

A non-cooperative game is characterized by the following elements:

- a set of players (or agents),
- a set of actions (or strategies) that players can choose,
- a cost incurred by each player which depends on its current action and the actions of the other players.

An important assumption in non-cooperative game theory is that the players are selfish. The goal of each player is to minimize its own cost, which depends on the actions of all the players. If there is only one player the resulted situation is an optimization problem.

An interesting class of games is that of potential games. A game is said to be a potential game if the optimization problem that selfish users satisfy can be expressed as a global optimization problem. The objective function of this global optimization problem is called a potential function. An important result for this type of games is that there always exists a Nash equilibrium. However, it is not always easy to find the potential function.

We also define a symmetric game as a game where every player is identical with respect to the game. This means that the identity of the player does not change the costs obtained by the players.

The Equilibrium

We now present the notion of equilibrium, which is a very important concept in non-cooperative games. A set of strategies is said to be an equilibrium if no player can decrease its cost by changing its strategy unilaterally.

We observe that an equilibrium in the Pigou's example is transmitting all the packets by the fast route. This situation is an equilibrium since if one packet changes its strategy and starts using the slow route, its delay will never decrease (the delay of the slow route is 1 second). We observe that in the equilibrium the delay of the message is 1x1 seconds.

The Price of Anarchy

It may happen that a central agent is able to set decisions so that the total cost in the system is minimized. This situation is said to be a social optimum (or social welfare). For the Pigou's example, the social optimum is the distribution of packets so that the complete packet is sent with minimum delay. It follows that the social welfare for Pigou's case is to send half of the message on each route. This result is the solution of an optimization problem which is simple to solve. The delay of the message in the social optimum is thus 0.5x0.5+0.5x1 seconds, i.e., 0.75 seconds.

Pigou's example shows an important property: the outcome of selfish behaviours need not optimize social welfare. In fact, the Price of Anarchy (PoA) [62] is the standard measure of the inefficiency of non-cooperative algorithms. The PoA is defined as the ratio of the cost function in the worst equilibrium over the cost in the social welfare. If the value of the PoA is close to one we say that the obtained equilibrium is efficient. On the other hand, if the value of the PoA is high we say that the equilibrium is not efficient. In the Pigou's example, the Price of Anarchy is the ratio between 1x1 and 0.5x0.5+0.5x1, which is 4/3.

Routing Games

Throughout this thesis, we will be interested in games where the players route traffic. This type of games are known in the literature as routing games. Routing games can be classified as atomic and non-atomic games depending on the amount of traffic the users control. In the atomic case each user controls a non-infinitesimal amount of traffic, see [79, 26] for some interesting atomic games. An equilibrium that arises in the atomic case is called a Nash equilibrium. If the users can send a portion of the traffic to different routes, we say it is an atomic splittable game. In contrast, if the players cannot split the flow, we have an atomic non splittable game. On the other hand, if each user controls a negligible amount of the incoming traffic and can take its own decision, then we have a non-atomic game. The corresponding equilibrium is known as the Wardrop equilibrium [90], where response times are minimal and equal on all routes. The non-atomic case is

sometimes studied as a particular case of the atomic case, where the number of players grows to infinity.

Pigou's example is a non-atomic routing game. Hence, the equilibrium we obtain in this game is a Wardrop equilibrium.

2.2.3 Best-Response Dynamics

In non-cooperative game theory, it is assumed that the users behave rationally, i.e., each player seeks to minimize its cost. An important question is that of the dynamics that lead rational users to converge to an equilibrium.

The best-response of a user is the action it takes in order to minimize its cost when the actions of the other players are fixed. The best-response can be seen as the rational strategy that a selfish player chooses, when the action of the others cannot vary. The equilibrium can be also seen as a fixed point for the best-response, since all the users doing best-response do not modify their strategy at this point.

We refer to sequential best-response dynamics as the set of rational actions that the players take, when they do best-response in a round-robin way. We are also interested in the convergence of the best-response dynamics, which means that a fixed point is reached when users do best-response a finite number of times.

2.3 Graph Theory

Graph theory studies the properties of graphs. Many situations can be modelled by graphs. We only explain a few of them here. A first example is the social networks, where users and their friendships can be characterized using nodes and edges. In the transportation theory, the cities and the routes that connect them can be also modelled by a graph. Graphs are also used to model telecommunication networks such as Internet. In fact, telecommunication networks can be modelled as a graph where the nodes are the devices that transmit or forward data.

We use graph theory to find the optimal path in a telecommunication network. Many researchers have been interested in studying the properties of graphs and in optimizing the performance of telecommunication systems using graphs. In the following sections, we describe the main definitions of graph theory as well as well-known techniques for the shortest path problem in graphs, such as Dijkstra's algorithm and A^* algorithm. Prior to that, we present the problem of the seven bridges of Königsberg.

2.3.1 Seven Bridges of Königsberg

The problem of the seven bridges of Königsberg is a very famous problem. The city of Königsberg was located in Prussia and in this city there was a river that divided the

city into four separate landmasses, where one of them was an island. These regions were connected by seven bridges as shown in the left part of Figure 2.4. Residents of the city wondered if it were possible to leave home, cross each of the seven bridges only once, and return home. Leonard Euler solved this problem using a methodology that is considered as the birth of graph theory.

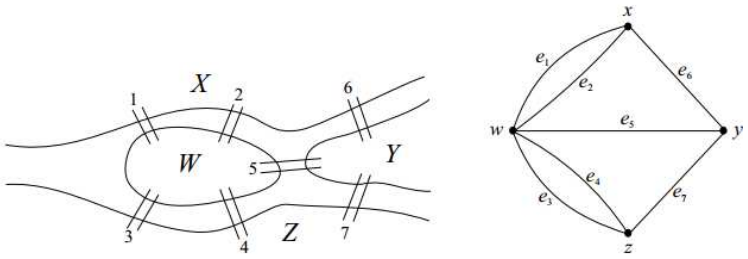

Figure 2.4: The problem of the seven bridges of Königsberg.

The solution is based on representing each of the land masses as a vertex and representing each bridge as an edge connecting the vertices corresponding to the land masses. The new situation is depicted in the right part of Figure 2.4, which is a graph of four nodes and seven links. The problem is thus reduced to find a path that starts and finishes in the same node such that it traverses each edge only once. Euler showed that there is no such path.

2.3.2 Fundamental Definitions of Graph Theory

A graph is defined by a pair $G = (V, E)$, where V is the set of nodes and E is the set of edges. The number of nodes in the graph or size of the graph is defined as $|V|$ and we say that there are n nodes in the graph when $|V| = n$. Two graphs $G = (V, E)$ and $H = (V', E')$ are said to be equal if $V = V'$ and $E = E'$. A graph is null if $n = 0$ and a graph is trivial if $n = 1$. A graph is finite if $|V|$ and $|E|$ are finite. A complete graph is a graph in which every pair of nodes is connected by an edge.

There are two important concepts of graphs that we present in the following lines: connectivity and directivity. A graph is connected if there is always a path between any two nodes. The connectivity of the graph ensures us that no node is isolated and thus, for any pair of nodes there is a path that connects them. Otherwise, the graph is called disconnected. Graphs can be classified as directed or undirected graphs. In undirected graphs the edges have no orientation. Hence, for undirected graphs, an edge from u to v is the equivalent to an edge from v to v, i.e., the edges are not ordered pairs, but sets $\{u, v\}$ of vertices. If the edges have a direction, the graphs are defined as directed graphs. In this case, the edges are also known as arrows. If an edge $e = (u, v)$ is oriented from u to v, then v is called the head and u is called the tail of the edge.

We define the degree of a node by the number of edges incident to it. In a directed

graph, we distinguish the in-degree and out-degree of a node, depending on the number of edges that come into and out of that node, respectively.

For a given edge, the nodes that it connects are called endpoints of the edge. If the two endpoints of an edge are the same node, the edge is called a loop. We say that a graph has parallel edges if at least two edges have the same endpoints. A graph is simple if there are no loop or multiple edge. In simple graphs, each edge is characterized by its endpoints. The graphs we consider in this thesis are simple, complete and undirected, as represented in Figure 2.5.

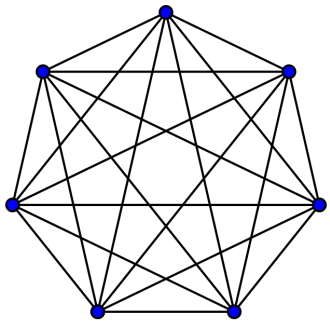

Figure 2.5: Simple complete undirected graph of 7 nodes.

For a given graph $G = (V, E)$, we say that the graph $H = (V', E')$ is a subgraph of G if $V' \subseteq V$ and for all $e \in E' \subseteq E$, the endpoints of e belong to V'. Besides, the graph G is also called supergraph of H, if H is a subgraph of G.

A path in G as a finite ordered set of nodes in which all nodes are distinct. A path that connects nodes u and v is a path whose first node is u and last node is v. An edge belongs to a path if and only if the edge is formed by two consecutive nodes in the path. A cut is a partition of the set V of nodes into two disjoint subsets. The cut-set of a cut is the set of edges whose end points are in different subsets of the partition.

A walk is a list $v_0, e_1, v_1, \ldots, e_k, v_k$ of vertices and edges that verifies that for $1 \leq i \leq k$, the edge e_i has endpoints v_{i-1} and v_i. We also define a trail as a walk with no repeated edges. When the first and last vertex of a walk or trail are the same, we say that it is a closed trail or a circuit.

We also present some notation. A path that connects two nodes u and v is denoted, in general, by $P_{(u,v)}$ and we say that an edge $e \in E$ belongs to a path if $e \in P_{(u,v)}$. The set of all possible paths from u to v is denoted by $\mathcal{P}_{(u,v)}$. We denote by $f(e)$ the value of the edge $e \in E$.

For any set of edges $H \subseteq E$, we define the value of H as $F(H)$, where $F : 2^E \to [0, \infty)$ is a function that depends on the values of the edges of H, i.e., $F(H) := F((f(e))_{e \in H})$. An optimal path between nodes u and v is a path $P^*_{(u,v)}$ in $\mathcal{P}_{(u,v)}$ that optimizes its performance, i.e., maximize or minimize the value $F(P^*_{(u,v)})$. If the optimization goal is to minimize an additive function of the form $F(H) = \sum_{e \in H} f(e)$, we say that the

optimal path is the shortest path, i.e., the path with least cost.

Another interesting case is the widest-path problem. In this case, given two nodes, the goal is to find the path between those nodes that maximizes the value of the minimum-value edge in the path. Using the previous notation, we can also characterize the widest path problem as follows: the optimization goal is to maximize and the function F we consider is the minimum, i.e., $F(H) = \min_{e \in H} f(e)$. This problem is also known in the literature as bottleneck shortest path problem or the maximum capacity path problem [89].

2.3.3 Shortest Path Algorithms

Algorithms that search for the shortest path is a widely studied issue in graph theory. In this section, we describe two algorithms that are mentioned several times: Dijkstra's algorithm [30] and A^* [49]. The objective of both techniques is to find a shortest path. In our explanations, the graphs under consideration are undirected.

Dijkstra's Algorithm

For a given source node, Dijkstra's algorithm finds the path with shortest cost between that node and every other nodes. Here, we explain a particular case of this algorithm that finds the shortest paths from a single node to a single destination node. This is achieved stopping the algorithm once the shortest path to the destination vertex has been determined.

A description in pseudo-code of Dijkstra's algorithm is represented in Algorithm 1. We observe that Dijkstra's algorithm works iteratively. In the first step, the closest node from the starting point is selected. Then, in each round, it examines the not-yet-examined nodes from the node that has been selected in the previous step, and selects the closest one among those that has not been previously selected. This process continues until it reaches the destination node.

Dijkstra's algorithm finds a shortest path from the starting point to the destination, as long as none of the edges have a negative cost.

Algorithm 1 Dijkstra's algorithm from a single node to a single destination.

```
1: INPUT G, source, target
2: INITIALIZE selected[source]=1 and selected[i]=0, for all i except for source;
   s_node=source;
3: while selected[target] is zero do
4:    COMPUTE cl_node = closest not selected neighbour from s_node .
5:    UPDATE s_node = cl_node.
6:    UPDATE selected[s_node]=1.
7: end while
```

The output of the presented algorithm is the distance of a shortest path from two nodes. If we also need to compute a shortest path, small modifications in the code can be done in order to store the predecessor of each node. Then, when the algorithm finishes, it obtains the predecessor of the destination node and, so on, until some node's predecessor is the start node.

We now briefly explain some interesting properties of the shortest path found by Dijkstra's algorithm. First, we observe that the subpath of any shortest path is itself a shortest path. Second, we observe that the obtained solution satisfies the triangle inequality, which means that the shortest path between u and v is less than the distances of the shortest paths between u and x and of the shortest path between x and v. Finally, we remark that the running time of Dijkstra's algorithm is of order of $|V| + |E|$.

The A^* Algorithm.

A very popular method for finding a shortest path between two given nodes is the A^* algorithm. This algorithm uses estimations of the distance of the nodes to the destinations. Hence, for a node n, we define the cost of this node as the following sum

$$f(n) = g(h) + h(n),$$

where $g(n)$ is the cost to get to the node from the source and $h(n)$ is the estimation of the cost from the node to the destination. The second term of the latter sum is the heuristic that the A^* algorithm uses.

The heuristic that the algorithm uses has a very important role in the algorithm. In fact, a bad estimation of the costs leads to a big augmentation of the processing time required to achieve the shortest path. A common assumption for the A^* algorithm is that the heuristics are monotonic or consistent. In fact, if this condition does not hold it can occur that the algorithm visits nodes that have been previously visited.

We now explain how the A^* algorithm works. It starts from the source node, computes the f value of its neighbours and it orders these nodes according to their f value. Hence, in each step, the node with minimum f value is chosen and the algorithm calculates the f value of its neighbours, updating the order of the nodes, if required. The algorithm finishes when the destination node is the node with the lowest f value or when all the nodes have been visited.

As in the Dijkstra's algorithm, the presented algorithm only gives us the distance of the shortest path. However, as in the case of Dijkstra's algorithm, we can also modify the algorithm to obtain the shortest path. To do that, we need to register the predecessor of all the selected edges.

We also observe that Dijkstra's algorithm and the A^* are not unrelated. If the values of the h function is zero for all the nodes, A^* algorithm coincides with Dijkstra's one.

2.4 Conclusion

In this chapter, we have introduced the methodologies that will be used in the other chapters. The concept and definitions from graph theory will be in Chapter 5 used when addressing the optimal path discovery problem, while queueing and game theory are used in the two next chapter to address games where a number of users compete for sharing a resource. In the next chapter, we consider the case of a single server shared by non-cooperative users with relative priorities.

3

Single Server with Relative Priorities

We investigate a game in which users pay for using a shared resource. The share of the resource allocated to a user is directly proportional to the payment it makes. Each user wants to minimize its payment while ensuring a certain Quality of Service (QoS). This problem is modelled as a non-cooperative resource-sharing game.

This chapter is organized as follows. In Section 3.1 we introduce the problem and the main contributions of this work. In Section 3.2 we put our work in the context of the existing literature. In Section 3.3 we describe the model and we formulate the resource-sharing game. In Section 3.4 we focus on the solution of the game and we analyze its efficiency. For the cases in which this game cannot be solved in closed-form, we define in Section 3.5 a game for the heavy-traffic regime, i.e., when the load in the system approaches one. Then, we show that the obtained result in the heavy-traffic case can be used to get an approximation of the original game. In Section 3.6, we present numerical experiments to assess the accuracy of the presented approximation. Finally, we present the main conclusions of this work in Section 3.7. The proofs of the most relevant results of this work are given in Appendix 3.A.

3.1 Resource Competition in a Single Server

In this part of thesis, we aim to analyze a queueing game in a single server. As we said in Section 2.1.1, an interesting case is given when all the jobs in the system are served simultaneously and get a portion of the capacity. Thus, in the latter case, there is no job waiting in the queue. These models are known in the literature as resource-sharing,

processor-sharing or capacity-sharing systems. The model we analyze is of this type.

In file hosting services, there is a provider whose available bandwidth is shared among all the users that want to get service in the same time. This situation can be modelled as a resource-sharing system since all the users use a fraction of the available bandwidth. In this model, the users are the agents that compete for the capacity of the provider, i.e., they are the players of the resource-sharing game. The players are also called selfish agents.

In these systems, users can subscribe a premium service so as to increase the upload/download speed. In our work we assume that users can choose their payments, that is directly proportional to their allocated bandwidth (or priority in the queue). This payment is lower-bounded by the minimum price to get access to service. Hence, each user aims to choose its priority level so as to minimize its own payment, while guaranteeing that the mean response time of its jobs does not exceed its deadline (QoS condition).

Bandwidth models have been widely studied in the literature, see [6] and [76] for surveys on this topic. A common model to analyze bandwidth sharing systems is the Processor Sharing (PS) queue. In this discipline, all the users are served simultaneously and at the same rate, i.e., there is no service differentiation among users. In our study, the users pay to get a faster service and, as a consequence, we need to investigate a model that differentiates the speed of service of different users. Hence, in this work we assume that the capacity is shared according to the Discriminatory Processor Sharing (DPS) discipline [58]. As we explained in Section 2.1, the DPS model is a multiclass generalization of the PS model. The main characteristic of the DPS discipline is that it gives service to all the users simultaneously, but the speed of service of each user depends its relative priorities. Thus, since the payment of a user is directly proportional to its priority, if we change the payments of the users, we can control the service rates of different users.

We now give an example of the game that we aim to analyze in this chapter. Consider there are three users (A, B and C) that compete for the resource of a server. Users A, B and C have deadlines equal to 4 seconds, 10 seconds and 20 seconds, respectively. We consider that the service requirements of all the users is exponentially distributed with equal parameter $\mu = 1/2$. Besides, the load of each user is $\rho_A = 0.1$, $\rho_B = 0.2$ and $\rho_C = 0.3$. If all the users pay the minimum price, then all the users have the same priority. Thus, we can use the expression (2.1) to derive that the mean response time of the jobs of all the users is the same and equal to 5 seconds. We observe that this case is not an equilibrium since the QoS condition of user A is not satisfied. User A has therefore incentive to deviate by increasing its payment so as to meet its deadline.

In this model, a user doing best-response consists on the following: *(i)* if the mean response time of its jobs is less than its deadline, it diminishes its payment until its payment is the minimum price or the mean response time of its jobs equals its deadline;

(ii) if the mean response time of its jobs exceeds its deadline, the user increases its payment until the mean response time of its jobs equals its deadline. Hence, if user B or C do best-response, they do not change their payment since they are paying the minimum price and their QoS condition is satisfied. However, if user A does best response, it increases its payment until the mean response time of its jobs equals its deadline. Thus, after the best-response of user A, the mean response times of jobs of users A, B and C can be obtained as a solution of the system of equations given in (2.2) and are 4, 5.2 and 5.2 seconds, respectively. Besides, the payment of user A is 1.875 times the minimum price. We observe that in this case the QoS condition of all the users is satisfied. We therefore claim that this is an equilibrium since users B and C are paying the minimum price and, if user A decreases its payment, its QoS condition is not satisfied.

The total benefit of the system, or simply total benefit, is defined as the sum of the payments of the users. The provider can set the payments of all the users so that the total benefit of the system is minimized and all users satisfy their own response time constraint. A set of payments that satisfies this condition is called social optimum or social welfare of the system. We observe that, by definition, the total benefit in an equilibrium cannot be less than the total benefit in the social welfare. In this context, the Price of Anarchy (PoA) is defined as the ratio of the total benefit in the equilibrium to the social welfare. We say that the set of payments of selfish agents (or the competition) is efficient if the total benefit in the equilibrium and in the social welfare coincide, i.e. when the PoA is one.

The main goal of this work is to study the properties of the non-cooperative resource-sharing game that arises from the interaction of the various users that compete for the capacity of a resource. We are interested, for example, in the required conditions that a game must verify to ensure the existence of the equilibrium. Another important aspect we address is the uniqueness of the equilibrium. We also look at the dynamics of the best-response algorithm in this resource-sharing game.

The main contributions of this chapter are summarized in Table 3.1. We give the necessary and sufficient conditions for the existence of the equilibrium of the game for exponential service times and arbitrary number of users. For general service times and two players, we show that the equilibrium is unique and that the Price of Anarchy is one. When the number of players is two and the service times are exponentially distributed, we characterize the unique equilibrium of the game. We prove that the dynamics of best-response converge in two settings: (i) for two users, exponential service times and any initial point and (ii) arbitrary number of users, general service times and feasible initial point. For the rest of the cases, given the difficulty of this model, we use heavy-traffic results for DPS from [44] and [74] to obtain tractable expressions for the mean response time in the system. Even though of approximate nature, we believe that the heavy-traffic approach allows to derive interesting insights into the performance of the system. Using the heavy-traffic approximation, we characterize the sufficient and

Contributions	Original Game		Heavy-Traffic Game	
	N. Players	Serv. Times	N. Players	Serv. Times
Feasibility	Arbitrary	Exponential	Arbitrary	General
Existence of NE	Arbitrary	General	Arbitrary	General
Uniqueness of NE	2	General	Arbitrary	General
NE Characterization	2	Exponential	Arbitrary	General
Price of Anarchy	2	General	Arbitrary	General
BR Convergence (feasible point)	Arbitrary	General	Arbitrary	General
BR Convergence (any point)	2	Exponential	2	General

Table 3.1: Summary of the main contributions of this chapter.

necessary conditions for the game to have a Nash equilibrium, and then show that this equilibrium is unique and fully characterize it. Interestingly, we show that players can be ordered in a decreasing order with respect to the ratio between the mean size requirement and their constraints on the response time and that in equilibrium, the prices that users pay decrease as this ratio decreases. Furthermore, we prove that the Price of Anarchy of the heavy-traffic game is always one. We then explain how the heavy-traffic solution can be used to obtain an approximate solution to the original problem. The numerical experiments illustrate that when the various users have a similar ratio between the mean size and response time constraint, then the heavy-traffic approximation predicts satisfactorily the outcome (both in terms of equilibrium prices and performance) of the original game. However, when the disparity of the users increases the error in predicting the equilibrium prices can be very significant, but in spite of this, the heavy-traffic approximation remains quite accurate regarding the performance. The numerical results show that the dynamics of the best-response also converge outside the two settings described above.

3.2 Related Work

The single server queue is a classical model in queueing theory and many results can be remarked regarding the policy that gives an optimal performance of the system. We know that this policy in a single server queue changes depending on the available information of the scheduler to give service. For example, if the scheduler knows the service time of the incoming jobs, the optimal policy is the Shortest-Remaining-Processing-Time discipline [83] and [84]. On the other hand, if the scheduler does not know the remaining service time of jobs but the distribution of the service times is known, then the Gittins index policy is optimal [42, 43].

We are interested in the scheduling policies in a single server where the capacity of a server is shared among a certain number of users. The PS policy is the basic model and, given its wide range of applications, it has attracted significant attention from networking researchers, see for example [17, 95, 94]. In this work, we consider

that the incoming jobs are served according to the DPS discipline, which is multiclass generalization of the PS discipline. The DPS model has been studied for many years, see [56, 24], and it is shown to be a good discipline to model flow level performance and bandwidth-sharing applications, see [8] for an extensive overview of the literature on DPS. Some basic results on PS and DPS queues have been given in Section 2.1.

In a seminal paper, Fayolle et al. proved that for exponential service time distributions, the mean response time of the DPS queue is the solution of a system of equations [34]. The mean response times is given in closed-form only for the case of two classes. For general service time distributions the results are scarce. In [34] the authors showed that the derivative of the mean conditional (on the service requirement) response time of the various classes satisfies a system of integro-differential equations. To the best of our knowledge, there are no known tractable results on the distribution of the response time. However, we here mention some interesting approaches that have been proposed in the literature. The authors in [20] give closed-form expressions for the mean response time in the DPS queue when there is admission control. In [18] the authors analyze response time asymptotics in the DPS queue. The DPS queue in heavy-traffic was considered in [44] (see also [74] and [88]). The authors prove that, when scaled with $1-\rho$, the response time has a proper distribution as $\rho \to 1$ (see Section 2.1). Using this heavy-traffic result, the authors in [55] obtain a polynomial approximation of the mean response time in the DPS queue.

The resource-sharing game under consideration is complicated to analyze in a general setting since, as we said before, the underlying queueing model has no closed-form expression for the mean processing times of the jobs. That is why results on strategic behaviour of users in systems with relative priorities are so scarce. Some exceptions are given now. In [54] the authors consider two types of applications in a DPS queue that compete to be served and they analyze how optimal prices can be found. In [51] the authors consider a DPS model with two classes, entry fees and waiting costs and the server is allowed to set the priorities vector so as to maximize the profit. A more recent work is [93] and [92], where the authors define a game for the DPS queue where each user seeks to minimize the sum of the expected processing cost and payment. Given the difficulty in analysing the model, the authors propose a heavy-traffic approximation, i.e. when the system is critically loaded, of the problem.Even though we also assume the DPS model for the sharing of the capacity, the problem we consider is different from the rest since, in our formulation, each user aims at minimizing its payment while ensuring its jobs to be served before a certain deadline.

The main application of this model is when the resources of a single server are shared among users. As another possible application of this model we have the study of the behaviour of non-cooperative users in the Information Centric Networking (ICN) model since the ICN has been recently modelled by the DPS queue [82]. In the ICN model, some parts of the contents that users wants to access from a server are stored in intermediate buffers of the network. The objective of the ICN is to reduce the load

in the server and provide a better service to the users.

3.3 Model Description

3.3.1 Notations

Consider a game in which a single server of unit capacity is shared among R classes (or users). Let $\mathcal{C} = \{1, 2, \ldots, R\}$ be the set of classes. We assume that the arrival process of jobs of class i is Poisson with rate λ_i and that the service requirements of jobs are i.i.d. and have an arbitrary distribution with mean $\mathbb{E}(B_i)$ and second moment $\mathbb{E}\left(B_i^2\right)$. For the case of exponential service time distributions, we will use the notation $\mathbb{E}(B_i) = \mu_i^{-1}$ and $\mathbb{E}\left(B_i^2\right) = 2/\mu_i^2$. We define the total incoming traffic of the system by $\lambda = \sum_{i=1}^{R} \lambda_i$. Let $\rho_i = \lambda_i \mathbb{E}(B_i)$ be the load of class i and the total load of the system be $\rho = \sum_{i=1}^{R} \rho_i$.

The processing capacity of the server is shared amongst jobs according to the DPS discipline, that is, all jobs present in the system are served simultaneously at rates controlled by a vector of priorities (or weights). If there are N_i jobs of class i present in the system, then class-i jobs are served at rate

$$r_i(N_1, \ldots, N_R) = \frac{g_i}{\sum_{j=1}^{R} g_j N_j}. \tag{3.1}$$

We observe that the rate allocated to each class depends on the relative priorities of the classes (or weights). Hence, by changing the weights, one can effectively control the instantaneous service rates of different job classes. For example, by setting the weight of a class close to infinity, one can give preemptive priority to this class. The possibility of providing different service rates to users of various classes makes DPS an appropriate model to study the performance of heterogeneous time-sharing systems.

We describe our game in the following section.

3.3.2 Game Formulation

We assume that the service provider (or the server) proposes to each class $i \in \mathcal{C}$ the choice of its weight g_i in exchange of a payment per-unit-of-work proportional to the chosen weight. Let $T_i(\mathbf{g}; \rho)$ be the random variable corresponding to the response time of a class-i job in a DPS queue for the vector of weights $\mathbf{g} = (g_1, \ldots, g_R)$ when the load in the system is $\rho < 1$. The quality-of-service metric of class i is the probability of its jobs missing a given deadline d_i. Class i then wants to ensure that this probability is below a certain threshold $\alpha_i \in (0, 1)$ while paying as little as possible for this service. Formally, class-i solves the problem

$$\begin{aligned} \min_{g_i \geq \epsilon} \quad & \rho_i g_i \\ \text{subject to} \quad & \mathbb{P}\left(T_i(\mathbf{g};\rho) > d_i\right) \leq \alpha_i. \end{aligned} \tag{OPT-P}$$

The quantity ϵ is the minimum price a class has to pay in order to get access to the service. It follows from (3.1) that the service rate every class gets for a vector $\theta\,\mathbf{g}$ is independent of the common factor $\theta > 0$ and as a direct consequence of this, we have that at least one user pays ϵ in the Nash equilibrium (if it exists).

We emphasize that the constraint in (OPT-P) is a *soft constraint* on the deadlines. In other words, even if some jobs miss their deadlines, these jobs stay in the system until completion, but in the long term at most a fraction α_i of class-i jobs will miss their deadline.

As explained in Section 2.1, the probability of jobs missing a deadline in a DPS queue has no easy-to-compute closed-form expression. One could then consider a game in which the constraints are based on the mean response time of tasks $\overline{T}_i(\mathbf{g};\rho) = \mathbb{E}(T_i(\mathbf{g};\rho))$. The optimization problem (OPT-M) then gets modified as follows:

$$\begin{aligned} \min_{g_i \geq \epsilon} \quad & \rho_i g_i \\ \text{subject to} \quad & \overline{T}_i(\mathbf{g};\rho) \leq c_i. \end{aligned} \tag{OPT-M}$$

The modified game (OPT-M) is not completely unrelated to the original game (OPT-P) as we shall argue next. As we saw in Section 2.1, when the load is high enough, the response time in the DPS model converges to an exponentially distributed random variable and thus

$$\mathbb{P}\left(T_i(\mathbf{g};\rho) > d_i\right) = \mathbb{P}\left(T_i(\mathbf{g};1) > (1-\rho)d_i\right) = e^{-\frac{(1-\rho)d_i}{\overline{T}_i(\mathbf{g};1)}},$$

which implies that

$$\mathbb{P}\left(T_i(\mathbf{g};\rho) > d_i\right) \leq \alpha_i \iff -\frac{(1-\rho)\,d_i}{\overline{T}_i(\mathbf{g};1)} \leq \log \alpha_i.$$

Since $\alpha_i \in (0,1)$, we have $\log \alpha_i < 0$ and, hence, we obtain the following equivalent constraint $\overline{T}_i(\mathbf{g};1) \leq \tilde{c}_i = -\frac{(1-\rho)d_i}{\log \alpha_i}$.

Let $\overline{T}_i(\mathbf{g};1)$ be the mean response time of class-i jobs in heavy-traffic. We propose to use the heavy-traffic result presented in Section 2.1 as an approximation to (OPT-P) and (OPT-M).

$$\begin{aligned} \min_{g_i \geq \epsilon} \quad & \rho_i g_i \\ \text{subject to} \quad & \overline{T}_i(\mathbf{g}; 1) \leq \tilde{c}_i. \end{aligned} \tag{OPT-HT}$$

In the case $\tilde{c}_i = -\frac{(1-\rho)d_i}{\log \alpha_i}$ we will be approximating (OPT-P), and if $\tilde{c}_i = (1-\rho)c_i$ we will approximate (OPT-M).

We know that the response time in a DPS queue has no tractable form in a general setting. Hence, we are able to analyze (OPT-M) under certain assumptions, see Section 3.4. For the general case, we develop in Section 3.5 an approximation based on the game (OPT-HT). Our hope is that the solution of the game (OPT-HT) will give useful insights into the equilibrium properties of (OPT-P) and (OPT-M). We emphasize that the benefit of the heavy-traffic approximation is that the mean response time formulae have a nice closed-form expressions for general service time distributions and arbitrary number of classes whereas (OPT-M) has a simple structure only in case of exponentially distributed service times, while (OPT-P) does not appear to be tractable even for that case.

We now give some definitions.

Definition 3.3.1 (Achievability)**.** *A vector* $\mathbf{t}$ *of mean response times is said to be achievable if there exists a vector of weights* $\mathbf{g} > 0$ *for which the vector of mean response times is* $\mathbf{t}$*, i.e.,* $t_i = \overline{T}_i(\mathbf{g}; \rho)$*, for all* $i \in \mathcal{C}$*.*

Let $\mathcal{T} = \{\mathbf{t} : \mathbf{t} \text{ is achievable}\}$ denote the set of achievable vectors. We now define the feasibility of the vector of deadlines.

Definition 3.3.2 (Deadline feasibility)**.** *A vector of deadlines* $\mathbf{c} \in \mathbb{R}_+^R$ *is feasible if and only if* $\exists \mathbf{t} \in \mathcal{T}$ *such that* $\mathbf{t} \preceq \mathbf{c}$*, where* $\preceq$ *is the componentwise order.*

In the following, we say that a game is feasible if its vector of deadlines is feasible. We will also use the notion of a feasible weight vector, as defined below.

Definition 3.3.3 (Weight feasibility)**.** *A vector of weights* $\mathbf{g} \in \mathbb{R}_+^R$ *is feasible if and only if* $\overline{T}_i(\mathbf{g}, \rho) \leq c_i$ *for all* $i \in \mathcal{C}$*.*

Definition 3.3.4. *A class* i *will be considered* fair *if the response time it would obtain under PS,* $\mathbb{E}(B_i)/(1-\rho)$*, would satisfy its own constraint on the mean performance* c_i*, i.e., if*

$$\mathbb{E}(B_i)/c_i \leq (1-\rho).$$

It is known, see [8], that $\overline{T}_i(\mathbf{g}; \rho)$ is decreasing with g_i and increasing in g_j for $j \neq i$. This implies that for the particular case when $\mathbf{c} \in \mathcal{T}$, the unique performance point

that satisfies all the constraints is $\mathbf{c}$. To see this, observe that if $\mathbf{c}$ is achievable then $\overline{T}_i(\mathbf{g},\rho) = c_i$ for all i, and that reducing $\overline{T}_i(\mathbf{g},\rho)$ for one class implies that $\overline{T}_j(\mathbf{g},\rho)$ increases for another class j. It can similarly be shown that if the game is feasible and $\mathbf{c} \notin \mathcal{T}$, then the number of performance vectors satisfying all the constraints is always larger than one.

Without loss of generality, when studying (OPT-M) we assume that the classes are ordered in decreasing order of $\mathbb{E}(B_k)/c_k$, i.e., if $i < j$, then $\mathbb{E}(B_i)/c_i \geq \mathbb{E}(B_j)/c_j$. We observe that the ratio $\mathbb{E}(B_k)/c_k$ is the minimum acceptable throughput of a class-k job with a service requirement equal to the mean. In the case of exponential service time distribution, it becomes $c_1\mu_1 \leq c_2\mu_2 \leq \cdots \leq c_R\mu_R$. Equivalently, when studying (OPT-HT) we will assume that classes are ordered in decreasing order of $\mathbb{E}(B_k)/\tilde{c}_k$.

Nash equilibrium

Assuming that the game is feasible, a vector of weights $\mathbf{g}^{NE} = (g_1^{NE}, \ldots, g_R^{NE})$ is a Nash equilibrium for the game (OPT-M) if each class is paying the least possible amount while ensuring that its mean response time does not exceed its deadline. Thus, we can say that a vector of weights $\mathbf{g}^{NE}$ is a Nash equilibrium if

$$g_i^{NE} = \min\left\{g_i \geq \epsilon \ : \ \overline{T}_i(g_i, \mathbf{g}_{-i}^{NE};\rho) \leq c_i\right\},$$

for all $i \in \mathcal{C}$, where $\mathbf{g}_{-i}^{NE} = \left(g_1^{NE}, \ldots, g_{i-1}^{NE}, g_{i+1}^{NE}, \ldots, g_R^{NE}\right)$. We also define the best response of class-i is defined as

$$g_i^{*} = \min\left\{g_i \geq \epsilon \ : \ \overline{T}_i(g_i, \mathbf{g}_{-i};\rho) \leq c_i\right\},$$

where $\mathbf{g}_{-i} = (g_1, \ldots, g_{i-1}, g_{i+1}, \ldots, g_R)$.

We now give the necessary and sufficient conditions of a vector of weights to be a Nash equilibrium. Using that $\overline{T}_i(\mathbf{g};\rho)$ is decreasing with g_i and increasing in g_j for $j \neq i$, it follows that, for a given i,

$$g_i^{NE} > \epsilon \quad \Rightarrow \overline{T}_i(\mathbf{g}^{NE};\rho) = c_i, \tag{3.2}$$

$$g_i^{NE} = \epsilon \quad \Rightarrow \overline{T}_i(\mathbf{g}^{NE};\rho) \leq c_i. \tag{3.3}$$

It is shown in [13] that $\overline{T}_i(\mathbf{g};\rho)$ is decreasing in g_i and increasing with g_j. Hence, class which is paying more than ϵ is necessarily satisfying its constraint with equality. Otherwise, if it were to be satisfying the constraint with strict inequality, then it could pay less and still satisfy its deadline. On the other hand, a class which is paying the least possible price could be satisfying its deadline with strict inequality.

We now present some properties that satisfy the equilibrium when the game is feasible.

Proposition 3.3.1. *With general service time distributions and arbitrary number of players, if the game is feasible, then*

- *there exists a Nash Equilibrium, and*
- *the dynamics of best-response converge to a Nash Equilibrium if the starting point is feasible.*

Proof. We notice that the dynamics of best-response are given by increasing the weight of class i when $\overline{T}_i(\mathbf{g};\rho) > c_i$ and decreasing the weight of class i when $\overline{T}_i(\mathbf{g};\rho) < c_i$ and $g_i > \epsilon$. Assume that we start the best-response dynamics from a feasible point $\mathbf{g}$. If all constraints $\overline{T}_i(\mathbf{g};\rho) \leq c_i$ are satisfied as equality constraints (implying that the deadline vector $\mathbf{c}$ is achievable), then $\mathbf{g}$ is clearly a Nash equilibrium since no class can unilaterally decrease its weight and still satisfy its constraint. If, on the contrary, there is a nonempty subset $\mathcal{A} \subset \mathcal{C}$ such that $\overline{T}_i(\mathbf{g};\rho) < c_i$ for all classes $i \in \mathcal{A}$, then we have either $g_i = \epsilon$ for all $i \in \mathcal{A}$ or there are some classes $i \in \mathcal{A}$ such that $g_i > \epsilon$. In the former case, $\mathbf{g}$ is again an equilibrium since clearly no class can decrease its weight. In the latter case, the best-response for each class $i \in \mathcal{A}$ such that $g_i > \epsilon$ is to decrease its weight. Moreover, after each best-response, the current vector of weights remains feasible because by decreasing its weight a class can only improve the mean response times of the other classes. Thus, in that case the best-response dynamics generate a sequence of feasible weight vectors which is strictly decreasing in the lexicographic order. Since feasible weight vectors belong to the set $[\epsilon, \infty)^R$ which is closed on the left, we can conclude that the dynamics of best-response converge to a Nash Equilibrium when started from a feasible point. □

From the latter result it follows that, if the game is feasible, then an equilibrium exists. Hence, to characterize the existence of an equilibrium we obtain the conditions of a game to be feasible.

Efficiency of the Equilibrium

In this section, we study the efficiency of the equilibrium of (OPT-M). We first present the social welfare. Then, we concentrate on the Price of Anarchy and the Price of Stability, which are standard measures of the efficiency of the equilibrium. We finally show that the equilibrium is efficient if it is unique.

We define the social welfare (or social optimum) of the system as the strategy of the users such that the total payment is minimum. It is the vector of weights that solves

the following minimization problem:

$$\begin{aligned} \min_{(g_1,\dots,g_R)} \quad & \sum_{i=1}^{R} \rho_i g_i \\ \text{subject to} \quad & \overline{T}_i(\mathbf{g};\rho) \leq c_i, \text{for all } i = 1,\dots,R, \\ \text{and} \quad & g_i \geq \epsilon, \text{for all } i = 1,\dots,R. \end{aligned} \tag{SOC-M}$$

The main difference with respect to the game is that in the latter each user minimizes its own payment while in the social optimum the users coordinate to choose the weights that minimize the total payment. By definition, the total payment at the social optimum cannot be larger than that at a Nash Equilibrium.

The efficiency (or sub-optimality) of the equilibrium of (OPT-M) can be measured using the notions of Price of Stability (PoS) and Price of Anarchy (PoS) which are defined as:

$$PoS = \min_{\mathbf{g}\in\mathcal{G}_M} \frac{\sum_{i=1}^{R} \rho_i g_i}{\sum_{i=1}^{R} \rho_i g_i^{SOC}}, \tag{3.4}$$

$$PoA = \max_{\mathbf{g}\in\mathcal{G}_M} \frac{\sum_{i=1}^{R} \rho_i g_i}{\sum_{i=1}^{R} \rho_i g_i^{SOC}}, \tag{3.5}$$

where $\mathcal{G}_M$ denotes the set of Nash equilibria of (OPT-M) and $\mathbf{g}^{SOC}$ is any vector of weights that is socially optimal. From these definitions, it follows that $PoA \geq PoS \geq 1$, and $PoA = PoS$ in particular when the Nash equilibrium is unique.

We now show the relation that there exists between the social optimum and the equilibrium.

Lemma 3.3.1. *A social optimum is an equilibrium of* (OPT-M).

Proof. Let $\mathbf{g}^{SOC}$ be a social optimum. We first observe that any social optimum is a vector of weights $\mathbf{g}^{SOC}$ such that each component verifies one of the following equations:

$$\begin{aligned} \text{if } g_i^{SOC} > \epsilon, \quad & \Rightarrow \overline{T}_i(\mathbf{g}^{SOC};\rho) = c_i, \\ \text{if } g_i^{SOC} = \epsilon, \quad & \Rightarrow \overline{T}_i(\mathbf{g}^{SOC};\rho) \leq c_i. \end{aligned}$$

If it would exist i such that $g_i^{SOC} > \epsilon$ and $\overline{T}_i(\mathbf{g}^{SOC};\rho) < c_i$, then it would be possible to decrease g_i^{SOC} while still satisfying the constraint $\overline{T}_i(\mathbf{g}^{SOC};\rho) \leq c_i$, implying that $\mathbf{g}^{SOC}$ would not be the solution of (SOC-M).

These equations give the necessary conditions for a vector to be the social optimum. They are, in fact, the same as (3.2) and (3.3) which are the necessary and sufficient conditions for a vector to be a Nash equilibrium. It then follows that a social optimum is also a a Nash Equilibrium. □

From this result and the definitions of (3.4)-(3.5), it follows directly the following corollary.

Corollary 3.3.1. *The game* (OPT-M) *satisfies that*

- $PoS = 1$,
- *if the equilibrium is unique, then* $PoA = 1$.

3.4 Solution

This section is devoted to the analysis of the game (OPT-M). We first establish in Section 3.4.1 the necessary and sufficient conditions for the existence of the equilibrium of the game. We then study the efficiency of the Nash equilibrium in Section 3.4.2. We provide an explicit characterization of the Nash equilibrium in Section 3.4.3.

3.4.1 Existence of the Equilibrium

We focus on the existence of the equilibrium of the game (OPT-M). As we said in Section 3.3.2, if the game is feasible an equilibrium exists. We thus analyze the feasibility of the game.

For fixed traffic conditions, the game is feasible if the vector $\mathbf{c}$ of deadlines is such that there is an achievable vector $\mathbf{t}$ of performances such that $t_i \leq c_i$ for all $i \in \mathcal{C}$. For exponential service times, the set of achievable vectors for the DPS queue was characterized in [40]. In order to present their result, we first need to introduce some notations. Let $\mathcal{R} = \mathcal{P}(\mathcal{C}) \setminus \emptyset$, where $\mathcal{P}(\mathcal{C})$ is the power set of $\mathcal{C}$, be the set of all subsets of $\mathcal{C}$ except the empty set. We define $\boldsymbol{\rho}_r = \sum_{i \in r} \rho_i$, and

$$W_r = \frac{1}{1 - \boldsymbol{\rho}_r} \sum_{i \in r} \frac{\rho_i}{\mu_i}, \tag{3.6}$$

for all $r \in \mathcal{R}$.

With these notations, the result reads as follows. A vector $\mathbf{t}$ of performances is achievable if and only if

$$\sum_{i \in \mathcal{C}} \rho_i t_i = W_{\mathcal{C}}, \tag{3.7}$$

$$\sum_{i \in r} \rho_i t_i \geq W_r, \ \forall r \in \mathcal{R} \setminus \{\mathcal{C}\}. \tag{3.8}$$

The following result gives the necessary and sufficient conditions for the game (OPT-M) to be feasible.

Theorem 3.4.1. *Assuming exponential service times, the game* (OPT-M) *is feasible if and only if*

$$\sum_{i \in r} \rho_i c_i \geq W_r, \ \forall r \in \mathcal{R}. \tag{3.9}$$

Proof. See Appendix 3.A.1. □

Observe that the achievability and feasibility conditions are similar, the difference being that the constraint on the whole set has to be satisfied as an equality for achievability, whereas it can hold as a strict inequality for feasibility.

3.4.2 Efficiency of the Equilibrium

We now study the efficiency of the equilibrium of the game (OPT-M). According to the result of Corollary R.2.1, if the game (OPT-M) is feasible, there exists at least one Nash equilibrium. In the following result we present the number of equilibria of this game.

Proposition 3.4.1. *For the game* (OPT-M) *with general service times, we have that*

- *For an arbitrary number of classes, if* $\mathbf{c} \in \mathcal{T}$*, then there is an infinite number of equilibria.*
- *For a two-player feasible game such that* $\mathbf{c} \notin \mathcal{T}$*, there is a unique Nash equilibrium.*

Proof. See Appendix 3.A.2. □

We recall that the case $\mathbf{c} \in \mathcal{T}$ is very particular, since it implies that $\mathbf{c}$ will be the only performance point that satisfies all the constraints.

From the previous result on the uniqueness of the equilibrium and Corollary 3.3.1, we obtain that the efficiency of the equilibrium is as follows:

Corollary 3.4.1. *For the game* (OPT-M) *with general service times, we have that*

- *For an arbitrary number of classes, if* $\mathbf{c} \in \mathcal{T}$*, then* $PoA = \infty$*.*
- *For a two-player feasible game such that* $\mathbf{c} \notin \mathcal{T}$*, then* $PoA = 1$*.*

We observe that in the case where $\mathbf{c}$ is feasible, i.e., $\mathbf{c} \in \mathcal{T}$, the solution of the game is not efficient since there are infinite equilibria. However, if $\mathbf{c} \notin \mathcal{T}$ and for two classes, we have showed that the equilibrium is unique and this means that the equilibrium is efficient.

3.4.3 Characterization of the Equilibrium

In the previous section, we studied the efficiency of the equilibrium for general service times. We now aim to characterize the equilibrium when $\mathbf{c} \notin \mathcal{T}$.

We first notice that explicit expressions of the mean response times in a DPS queue are known only in the case of two classes and exponential service times, see Section 2.1. This restricts the set of cases in which an explicit solution to the game can be computed.

Proposition 3.4.2. *For the two-player game with exponential service times and $c_1\mu_1 \leq c_2\mu_2$, if the game is feasible and $\mathbf{c} \notin \mathcal{T}$, then the unique equilibrium is*

- $\mathbf{g^{NE}} = (\epsilon, \epsilon)$ *if class 1 is fair and*
- *otherwise,* $\mathbf{g^{NE}} = (g_1^{NE}, \epsilon)$*, where*

$$g_1^{NE} = \epsilon \, \frac{-\mu_1\rho_2 + \mu_2(1-\rho_2)\left[\mu_1 c_1(1-\rho) - 1\right]}{-\mu_1\rho_2 - \mu_1(1-\rho_1)\left[\mu_1 c_1(1-\rho) - 1\right]}.$$

Proof. See Appendix 3.A.3. □

We explain briefly the structure of the Nash equilibrium. Assuming feasibility, it follows from the ordering of the classes that at least class 2 is fair, i.e., $\overline{T}_2(\mathbf{g}^{NE}; \rho) \leq c_2$. If class 1 is also fair, then $(g_1, g_2) = (\epsilon, \epsilon)$ is the equilibrium. On the contrary, if the mean response time of class 1 for PS weights exceeds its deadline c_1, the class 1 must pay $g_1 > \epsilon$ per unit-of-work to ensure that its time constraint is satisfied.

In Section 3.3.2 we showed that the dynamics of the best-response algorithm converge to the equilibrium if the starting point is feasible, i.e., if $\overline{T}_i(\mathbf{g}; \rho) \leq c_i$, for all $i \in \mathcal{C}$. We now show that the dynamics of the best-response, starting from any point, converge to the Nash Equilibrium.

Proposition 3.4.3. *For the two-player game with exponential service times, if the game is feasible and $\mathbf{c} \notin \mathcal{T}$, the best-response dynamics converge to the Nash Equilibrium for any starting point.*

Proof. See Appendix 3.A.4. □

We now study how the equilibrium of (OPT-M) changes with the total load in the system. For an arbitrary number of users, we define ρ_E and ρ_F as the threshold values such that

- if $\rho \leq \rho_E$ then all classes are paying the minimum price ϵ,
- if $\rho_E < \rho \leq \rho_F$ the game is feasible and there is at least one class paying more than ϵ and
- if $\rho > \rho_F$ the game is not feasible.

Characterization of ρ_E

We now concentrate on the value of ρ_E. For general service time requirements, it follows from the ordering of the classes, $\frac{\mathbb{E}(B_1)}{c_1} \geq \frac{\mathbb{E}(B_2)}{c_2} \geq ... \geq \frac{\mathbb{E}(B_R)}{c_R}$, that if class 1 is fair, that is if

$$\frac{\mathbb{E}(B_1)}{c_1} \leq 1 - \rho,$$

then all the users are fair and the equilibrium is unique and equal to $(\epsilon, \ldots, \epsilon)$. In this case, the value of the PoA is one, which means that the game is efficient. Hence, the minimum value ρ_E such that at least one user pays more than ϵ is obtained when

$$\frac{\mathbb{E}(B_1)}{c_1} = 1 - \rho_E,$$

that is for $\rho_E = 1 - \frac{\mathbb{E}(B_1)}{c_1}$.

We remark that this expression of ρ_E holds for general services times and arbitrary number of classes. We also note that if $\mathbb{E}(B_1)/c_1$ is close to 0, then ρ_E is close to 1, implying that the PS solution $(\epsilon, \ldots, \epsilon)$ corresponds to the equilibrium for a large range of utilization rates.

Characterization of ρ_F

We present the value of the system load that makes the game not feasible. Assuming exponential service times, we gave in Theorem 3.4.1 the necessary and sufficient condition for the game to be feasible. Here, we use this result to characterize the value of ρ_F for exponential service times. Hence, we state that ρ_F is the minimum value of the system load verifying that

$$\exists r \in \mathcal{R} \text{ such that } \sum_{i \in r} \rho_i c_i < W_r.$$

Identical minimum acceptable throughput

A particular case of interest is obtained when all classes have the same minimum acceptable throughput, i.e., $\mathbb{E}(B_i)/c_i$ is equal for all $i \in \mathcal{C}$. In this case, we characterize the equilibrium of the game and the value of ρ_F for general service times.

Proposition 3.4.4. *If $\mathbb{E}(B_i)/c_i = k < 1$ for all $i \in \mathcal{C}$, then*

- *the unique equilibrium of the game is the PS solution $(\epsilon, \ldots, \epsilon)$ for $\rho \leq 1 - k$, and*
- *the game is not feasible for $\rho > 1 - k$.*

Proof. If all users had the same weights (so the equilibrium were PS), we would have that $\mathbb{E}(B_i)/c_i = 1 - \rho$, for all i. Since $\mathbb{E}(B_i)/c_i = k < 1$, we conclude that if $\rho \leq 1 - k$

then $(\epsilon, \ldots, \epsilon)$ is the unique equilibrium. When $\rho = 1 - k$ we have $c_i = \mathbb{E}(B_i)/(1-\rho)$, $\forall i$, that is, the vector $(c_1, \ldots, c_R)$ is achievable and as soon as ρ increases further the game becomes infeasible. □

We thus have that for identical minimum acceptable throughput the values of ρ_F and ρ_E coincide are is equal to $1 - k$. We also conclude from this result that in case of identical minimum acceptable throughput when $\rho \leq 1 - k$ the equilibrium is efficient since it is unique.

3.5 Heavy-traffic Approximation

In the previous section, we analyzed the equilibrium of the game (OPT-M). Given the complexity of the queueing model we consider, we are not able to solve this problem in closed-form for the general case. Hence, in this section, we develop an approximation based on the game (OPT-HT) which is valid in heavy-traffic. In Section 3.5.1, we study the existence of a solution of this game and then we analyze the efficiency of the obtained equilibrium in Section 3.5.2. Finally, we explain how the solution of the game (OPT-HT) can be used to give an approximation of the game (OPT-M).

3.5.1 Existence of Heavy-Traffic Equilibrium

In this section we focus on the existence of the heavy-traffic-equilibrium. We first define an equilibrium for the game (OPT-HT). A vector $\mathbf{g}^{NE}$ is a Nash equilibrium for (OPT-HT) if

$$g_i^{NE} = \operatorname{argmin} \left\{ g_i \geq \epsilon \ : \ \overline{T}_i(g_i, \mathbf{g}_{-i}^{NE}; 1) \leq \tilde{c}_i \right\},$$

for all $i \in \mathcal{C}$, where $\mathbf{g}_{-i}^{NE} = \left(g_1^{NE}, \ldots, g_{i-1}^{NE}, g_{i+1}^{NE}, \ldots, g_R^{NE}\right)$ and $\overline{T}_i(\mathbf{g}; 1)$ is the mean response time of class-i jobs in heavy-traffic which is given by (2.8).

We now define achievability and feasibility in heavy-traffic. A vector of performance $\mathbf{t}$ is said to be achievable in heavy-traffic if there exists a vector of weights $\mathbf{g} > 0$ for which $\overline{T}_i(\mathbf{g}; 1) = t_i$, for all $i \in \mathcal{C}$. On the other hand, we say that the game (OPT-M) is feasible if there exists a vector $\mathbf{g}$ such that $\overline{T}_i(\mathbf{g}; 1) \leq \tilde{c}_i$, for all $i \in \mathcal{C}$.

We concentrate on the expression given in (2.8) and we note that the mean response time in heavy-traffic of class-i jobs is decreasing with g_i and increasing with g_j, for all $j \neq i$. We also remark that Corollary R.2.1 follows since the mean response time of the DPS queue satisfies the latter property, that is $\overline{T}_i(\mathbf{g}; \rho)$ is decreasing with g_i and increasing with g_j, for all $j \neq i$. Hence, we can use the same reasoning as the one given in this corollary to state the following the result:

Corollary 3.5.1. *If the game* (OPT-HT) *is feasible, then*

- *there exists a Nash equilibrium for* (OPT-HT) *and*

- *the dynamics of best-response converge to a Nash Equilibrium if the starting point is feasible.*

Hence, from this properties, we can conclude that, for the heavy-traffic case as well, the feasibility of the game characterizes the existence of the equilibrium.

Prior to present our results on the feasibility of the game (OPT-HT), let us characterize the achievability in heavy-traffic. We denote by $\mathcal{T}^{HT}$ the set of all the performance vectors that are achievable in heavy-traffic.

The following proposition characterizes the achievability of a vector of mean response times in heavy-traffic:

Proposition 3.5.1. *A vector of performances* $\mathbf{t} \in \mathcal{T}^{HT}$ *if and only if*

$$\sum_{k=1}^{R} \lambda_k \frac{\mathbb{E}(B_k^2)}{\mathbb{E}(B_k)} t_k = \sum_{j=1}^{R} \lambda_j \mathbb{E}\left(B_j^2\right). \tag{3.10}$$

Proof. See Appendix 3.A.5. □

We now give the sufficient and necessary condition for the game (OPT-HT) to be feasible.

Proposition 3.5.2. *The game* (OPT-HT) *is feasible if and only if*

$$\sum_i \lambda_i \mathbb{E}\left(B_i^2\right) \left(\frac{\tilde{c}_i}{\mathbb{E}(B_i)} - 1 \right) \geq 0.$$

Proof. See Appendix 3.A.6. □

It is important to note that a sufficient condition for the game to be feasible is that all classes be *fair* in heavy-traffic. Note that $\overline{T}_i(\mathbf{g}^{PS}; 1) = \mathbb{E}(B_i)$, and thus

$$\overline{T}_i(\mathbf{g}^{PS}; 1) \leq \tilde{c}_i, \ \forall i \iff \mathbb{E}(B_i) \leq \tilde{c}_i, \ \forall i.$$

Using this property, if all the users are fair, then it follows that the left part of the inequality of Proposition 3.5.2 is positive, which means that the problem is feasible.

It is also important to see that the feasibility and achievability results given in this section are valid for general service times and arbitrary number of classes.

3.5.2 Efficiency of Heavy-Traffic Equilibrium

In this section, we assume that the game (OPT-HT) is feasible and we study the efficiency of the equilibrium of this game. We recall that it is assumed that the classes are ordered in decreasing order of $\frac{\mathbb{E}(B_i)}{\tilde{c}_i}$.

We note that if $\mathbf{c}$ is achievable in heavy-traffic, i.e., $\overline{T}_i(\mathbf{g};1) = \tilde{c}_i$ for all $i \in \mathcal{C}$, the vector $\mathbf{g}$ is an equilibrium. Furthermore, we observe that the weight vector $\theta\,\mathbf{g}$ satisfies $\overline{T}_i(\theta\,\mathbf{g};1) = \tilde{c}_i$ for all $i \in \mathcal{C}$ and is thus an equilibrium for any value of θ. Hence, we have that in the case where $\mathbf{c} \in \mathcal{T}^{HT}$ the are infinite equilibria.

The following theorem provides a complete characterisation of the unique Nash equilibrium when $\mathbf{c} \notin \mathcal{T}^{HT}$.

Theorem 3.5.1. *If the game is feasible and $\mathbf{c} \notin \mathcal{T}^{HT}$, the unique Nash equilibrium is*

$$\begin{aligned} g_i^{NE} &= \epsilon \frac{\tilde{t_m}/\mathbb{E}(B_m)}{\tilde{c}_i/\mathbb{E}(B_i)}, \text{ for all } i < m, \\ g_i^{NE} &= \epsilon, \text{ for all } i \geq m, \end{aligned}$$

where $m \in \mathcal{C}$ is the minimum value such that there exists a value $\tilde{t_m} \leq \tilde{c}_m$ verifying

$$\frac{\tilde{t_m}}{\mathbb{E}(B_m)} = \frac{\sum_{k=1}^{R} \lambda_k \mathbb{E}\left(B_k^2\right) - \sum_{k=1}^{m-1} \lambda_k \frac{\mathbb{E}\left(B_k^2\right)}{\mathbb{E}(B_k)} \tilde{c}_k}{\sum_{k=m}^{R} \lambda_k \mathbb{E}\left(B_k^2\right)}. \tag{3.11}$$

Proof. See Appendix 3.A.7. □

We now focus on the value of m. In the equilibrium, the following condition is satisfied:

- $\overline{T}_i(\mathbf{g};1) = \tilde{c}_i$, for all $i < m$.
- $g_i^{NE} = \epsilon$, for all $i \geq m$.

In the particular case where all classes are fair, i.e., $E(B_i) \leq \tilde{c}_i$ for all $i \in \mathcal{C}$, we notice that $m = 1$ and thus the equilibrium is $\mathbf{g^{NE}} = (\epsilon, \ldots, \epsilon)$. Besides, when $m = R$, there is only one class paying the minimum pice and the rest of the mean response time of the rest of the classes meet their deadlines.

The following corollary shows that the price paid by classes at the Nash equilibrium decreases as the ratio $\mathbb{E}(B_k)/\tilde{c}_k$ decreases.

Corollary 3.5.2. *If the game is feasible and $\mathbf{c} \notin \mathcal{T}^{HT}$, let $\mathbf{g^{NE}} = (g_1^{NE}, \ldots, g_R^{NE})$ be the vector of weights at equilibrium. We have*

$$g_1^{NE} \geq g_2^{NE} \geq \cdots \geq g_{R-1}^{NE} \geq g_R^{NE} = \epsilon$$

Proof. It follows from Theorem 3.5.1 and our assumption on the ordering of the classes. □

It is interesting to observe that the ordering of classes at equilibrium do not depend on the arrival or second moment of the distributions. Instead, the key parameter is the ratio $\mathbb{E}(B_k)/\tilde{c}_k$, which can be interpreted as the throughput of a class k. Thus, classes will deviate from the minimum weight in decreasing order with respect to the throughput they expect to obtain from the system.

We can also define the social optimum of the system in heavy-traffic:

$$\begin{aligned} \min_{(g_1,\dots,g_R)} \quad & \sum_{i=1}^{R} \rho_i g_i \\ \text{subject to} \quad & \overline{T_i}(\mathbf{g};1) \leq \tilde{c}_i, \text{for all } i = 1,\dots,R, \\ \text{and} \quad & g_i \geq \epsilon, \text{for all } i = 1,\dots,R. \end{aligned} \tag{SOC-HT}$$

The efficiency of the equilibrium is measured with the notion of Price of Anarchy which is defined as the ratio between the maximum payment of the users in the equilibria and the payment of the users in the social optimum.

We now stablish the value of the efficiency of the game (OPT-HT).

Corollary 3.5.3. *For the game* (OPT-HT)*, we have that*

- $\mathbf{c} \notin \mathcal{T}^{HT}$ *we have that* $PoA = \infty$*,*
- *otherwise, we have that the equilibrium is efficient since* $PoA = 1$*.*

Proof. We first observe that, using the same reasoning as in the previous section, that a social optimum is a Nash equilibrium in heavy-traffic.

We firs analyze the case when $\mathbf{c} \in \mathcal{T}^{HT}$. In this instance, we know that there is an infinite number of equilibria. Thus, we can conclude that the value of the PoA is infinity.

On the other hand, when $\mathbf{c} \notin \mathcal{T}^{HT}$, we have shown in Theorem 3.5.1 that the equilibrium is unique. Hence, the value of the PoA in the case is one. □

3.5.3 Approximation of the Original Game

We explain how the results of the game in heavy-traffic can be used to obtain insights into the solution of games (OPT-P) and (OPT-M).

As explained in Section 2.1, provided that ρ is sufficiently large for the approximation $\overline{T}_i(\mathbf{g};\rho) = \frac{\overline{T}_i(\mathbf{g};1)}{1-\rho}$ to be valid, the results established for game (OPT-HT) can be applied to approximate the solution of (OPT-P) by setting

$$\tilde{c}_i = -(1-\rho)d_i/\log\alpha_i.$$

Furthermore, we can also approximate the solution of (OPT-M) by setting

$$\tilde{c}_i = (1-\rho)c_i.$$

We will focus on (OPT-M). For this case we analyze this approximation to obtain the (approximated) equilibrium with an arbitrary number of classes and general service time distributions.

Existence of the Equilibrium

We now focus on the existence of the approximated equilibrium. Hence, we study the feasibility of the approximation. Assuming exponential service times, the characterization of the feasibility of (OPT-M) is given in Theorem 3.4.1. It consists on a set of inequalities that the vector of deadlines $\mathbf{c}$ must satisfy.

For general service times, we can characterize the (approximate) feasibility. From Proposition 3.5.2 and using that $\tilde{c}_i = (1-\rho)c_i$, it follows directly that the necessary and sufficient condition for the (approximate) feasibility of (OPT-M) is

$$\sum_i \lambda_i \mathbb{E}\left(B_i^2\right)\left(\frac{c_i}{\mathbb{E}(B_i)/(1-\rho)} - 1\right) \geq 0. \tag{3.12}$$

We observe that if all users are fair, then the game is feasible.

Using (3.12), we can approximate the value of ρ_F, as defined in Section (OPT-M), for general service times by

$$\rho_F = \frac{\sum_{i=1}^R \lambda_i \mathbb{E}\left(B_i^2\right)\left(\frac{c_i}{\mathbb{E}(B_i)} - 1\right)}{\sum_{i=1}^R \lambda_i \frac{\mathbb{E}\left(B_i^2\right)}{\mathbb{E}(B_i)} c_i}.$$

Hence, if the total load is higher than this value, the (approximated) game is not feasible.

Characterization and Efficiency

We aim to extend the result of Theorem 3.5.1 to the case $\rho < 1$. Hence, we use that $\tilde{c}_i = c_i(1-\rho)$ and $\tilde{t}_i = t_i(1-\rho)$ abd we obtain that the Nash-Equilibrium of (OPT-M)

can be approximated by

$$g_i^{NE} = \epsilon \frac{t_m/\mathbb{E}(B_m)}{c_i/\mathbb{E}(B_i)}, \text{ for all } i < m,$$

and

$$g_i^{NE} = \epsilon, \text{ for all } i \geq m,$$

where $m = 1, \ldots, R$ is the minimum value such that there exists a value $t_m \leq c_m$ verifying

$$\frac{t_m}{\mathbb{E}(B_m)} = \frac{\sum_{k=1}^{R} \frac{\lambda_k \mathbb{E}(B_k^2)}{(1-\rho)} - \sum_{k=1}^{m-1} \lambda_k \frac{\mathbb{E}(B_k^2)}{\mathbb{E}(B_k)} c_k}{\sum_{k=m}^{R} \lambda_k \mathbb{E}\left(B_k^2\right)}. \tag{3.13}$$

Note that if class 1 is fair, then all users are fair. In this case, the right-hand side of (3.13) is upper-bounded by $(1-\rho)^{-1}$, implying that

$$c_1 \geq \frac{\mathbb{E}(B_1)}{1-\rho} \geq t_1,$$

so that $m = 1$. Thus, if class 1 is fair, the approximate equilibrium corresponds to the PS solution $g_i^{NE} = \epsilon$ for all i, which is clearly the exact equilibrium.

Let us compare the above approximate characterization of the Nash equilibrium with the exact result given in Proposition 3.4.2 in the case of two users and exponential service time distributions. We observe that, in both cases, class 2 pays the minimum price. Besides, if class 1 is fair, then the corresponding equilibrium in both cases coincide and it is equal to the weights in the PS queue. We can state thus, that the approximation is exact when class 1 is fair. If class 1 is not fair, the equilibrium in both cases have the same form, i.e., $\mathbf{g^{NE}} = (g_1^{NE}, \epsilon)$, with $g_1^{NE} > \epsilon$. In Section 3.6 we analyze the accuracy of this approximation.

Finally, we want to measure the efficiency of the approximated equilibrium. Hence, we use the notions of Price of Stability and Price Anarchy, as defined in (3.4) and (3.5). Then, we observe that the approximated equilibrium is unique and this means that the PoA is one. Thus, we can claim that the approximated game is also efficient.

3.6 Numerical Experiments

In this section, we numerically study the most important properties of the results of this work. We first present several numerical experiments to compare the equilibrium of the game (OPT-M) (which we call the original problem) with that of the heavy-traffic approximation (OPT-HT). We then show that the dynamics of the best-response converge to the Nash Equilibrium of (OPT-M) from any starting point.

3.6.1 Validation of the Approximation

We analyze numerically the accuracy of the approximated equilibrium. Our main observation from the experiments that we conducted is that while in certain cases the error in weights can be substantial, the proposed heavy-traffic approximation is good at predicting the set of classes that pay a higher price than minimum price at the equilibrium, and the mean response times of the classes paying the minimum price.

The accuracy of the approximation is measured with the percentage relative error. The expression of the percentage relative error for the weight of class i is given by

$$\left| \frac{g_i^{SYS} - g_i^{HT}}{g_i^{SYS}} \right| \times 100,$$

where g_i^{SYS} (resp., g_i^{HT}) is its equilibrium weight for the original problem (resp. HT approximation).

Without loss of generality, the minimum weight ϵ is set to 1 in all the following experiments.

Exponential service time distribution

First, we present the results for exponentially distributed service times. In the first set of experiments, there are two players with deadlines $c_1 = 5$ and $c_2 = 6$, and the mean service times $\mu_1 = 2$ and $\mu_2 = 3$. Note that $c_1\mu_1 = 10 < c_2\mu_2 = 18$. We now vary the total system load starting from 0.8 until the system becomes unfeasible while maintaining $\rho_1 = 0.3\rho$ and $\rho_2 = 0.7\rho$. For each value of load, the equilibrium is computed using the best-response algorithm. In order to compute the best-response of a class for the original problem, the mean response time is computed from the system of equations presented in [40]. In the top subfigure of Figure 3.1, we plot the equilibrium weights for both the original problem and the HT approximation as a function of the total system load. The percentage relative error between the two is shown in the bottom subfigure of the same figure. Both problems become unfeasible for $\rho > 0.93$, so the data is restricted to $\rho \leq 0.93$. When the load of the system is between 0.9 and 0.93 we observe in Figure 3.1 (above) that the equilibrium of the heavy-traffic result approximates very well the equilibrium of the original problem. In particular, the heavy-traffic approximation follows the same increasing trend of the equilibrium weight of class 1 as that of the original problem. The error of class 1 users is small, while there is no error for the users of class 2. We see in Figure 3.1 (below) that the maximum percentage relative error is 9%.

In the second set of experiments, we present a scenario where the approximation becomes accurate when ρ is close to 1. We scale the deadlines by $(1-\rho)^{-1}$, that is, the deadline of user i, $c_i = \frac{\tilde{c}_i}{(1-\rho)}$ for some fixed $\tilde{c}_i$. This reflects that class i is aware that the performance worsens as ρ increases, and is willing to adjust its deadline correspondingly.

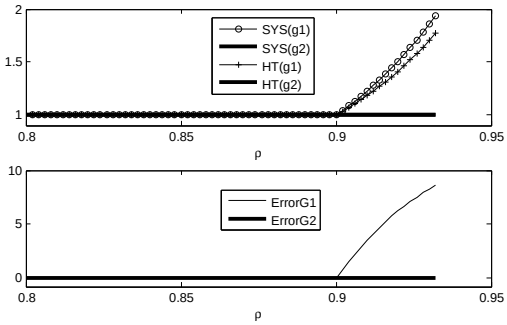

Figure 3.1: Comparison of equilibrium weights (above) and the corresponding percentage relative error (below) as a function of the total system load. $R = 2$ and exponential service time distribution.

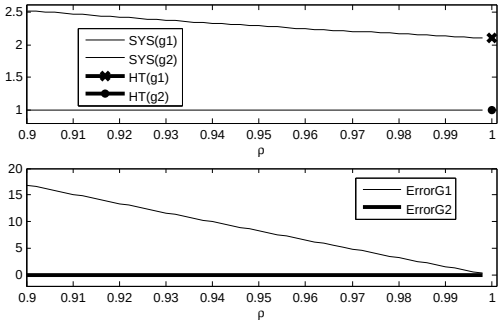

Figure 3.2: Comparision of equilibrium weights (above) and the corresponding percentage relative error (below) as a function of the total load, and the deadlines of the two classes are scaled by $(1-\rho)^{-1}$. $R = 2$ and exponential service time distribution.

When the deadlines are scaled with $(1-\rho)^{-1}$, the constraint on the mean response time of player i for the original problem becomes $\overline{T}_i(\mathbf{g};\rho) \leq \frac{\tilde{c}_i}{1-\rho}$, and that for the heavy-traffic approximation becomes $\overline{T}_i(\mathbf{g};1) \leq \tilde{c}_i$. Note that the latter constraint does not change with ρ. We set the parameters to : $\mu_1 = 2$ and $\mu_2 = 3$, $\rho_1 = 0.3\rho$, and $\rho_2 = 0.7\rho$, with the scaled deadlines being $\tilde{c}_1 = 0.3$ and $\tilde{c}_2 = 0.7$. In Figure 3.2, we present the accuracy of the heavy-traffic approximation as $\rho \to 1$. We observe that the error in the weight of class 1 reduces as the load tends to 1 which means that in heavy-traffic.

In the next set of experiments, we look at a four-player game with exponential service times. In Figure 3.3 the users have similar value of throughput, i.e., similar $c\mu$, and in Figure 3.4 they are more heterogeneous. The parameters of the users of each case are listed below each figure. In both figures, the equilibrium weights are plotted in the top subfigure, the corresponding error is plotted in the middle subfigure, and in the bottom subfigure we plot the error in the mean response times of the classes. The trend in the four-player plots is similar to that of the two-player example in which the deadlines are not scaled, i.e., the payment of all the classes is ϵ if $\rho \leq \rho_E$, at least one class pays more than ϵ if $\rho_E \leq \rho \leq \rho_F$ and if $\rho \geq \rho_F$ the problem is not feasible, where ρ_E and ρ_F are as defined in Section 3.4.3. We observe that the error in the weights is acceptable when the users are homogeneous (see middle subfigure of Figure 3.3) and the error in the weights can increase when the disparity of the users increases (see middle subfigure of Figure 3.4). A similar observation on the negative impact of heterogeneity on the error was also made in [93]. However, we conclude that, in both instances, the approximation captures correctly the set of users that pay more than ϵ and the prediction in the mean response times is acceptable.

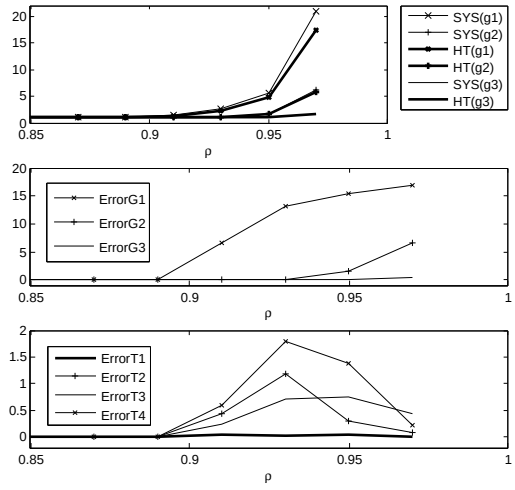

Figure 3.3: Comparison of equilibrium weights (above), the percentage relative error of the weights (middle) and the percentage relative error of the time (bellow) as a function of the total system load. $R = 4$ and exponential service time distribution. $\mathbf{c} = [10, 15, 25, 45]$, $\boldsymbol{\mu} = [1, 2, 4, 9]$.

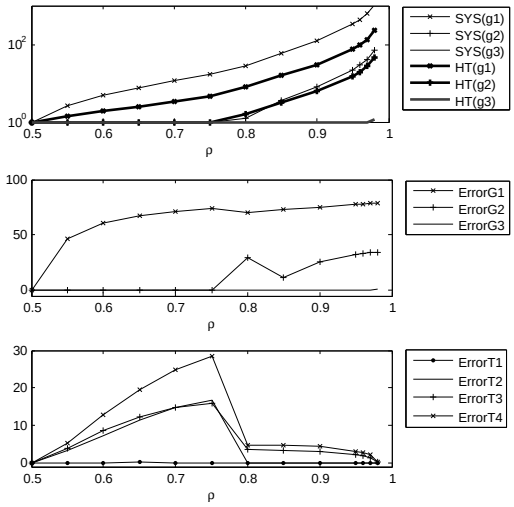

Figure 3.4: Comparison of equilibrium weights (above), the percentage relative error of the weights (middle) and the percentage relative error of the time (bellow) as a function of the total system load. $R = 4$ and exponential service time distribution. $\mathbf{c} = [5/3, 5/4, 10, 100]$, $\boldsymbol{\mu} = [1, 2, 8, 12]$.

Hyper-exponential service requirements

Finally, in this section, we compare the approximation for a two-player game with hyper-exponentially distributed service times. While there is no explicit expression for mean response time in DPS with service time distributions other than the exponential distribution, for the hyper-exponential distribution, a simple trick can be used to compute the mean response times using those of the exponential distribution. For example, consider a two-class DPS queue with hyper-exponential distribution of two phases each. The service rates of the phases are (μ_1, μ_2) for class 1 and (μ_3, μ_4) for class 2, and the arrival rates to these phases are (λ_1, λ_2) for class 1 and (λ_3, λ_4) for class 2. In order to compute the mean response time in this queue when the weights are $\mathbf{g} = (g_1, g_2)$, one first computes the mean response time in a four-class DPS queue with exponential distribution and weights $\mathbf{g} = (g_1, g_1, g_2, g_2)$. The arrival rate of class i in this queue is λ_i, and the rates of the exponential distribution of class i is taken to be μ_i. The mean respone time of class i in the DPS queue with hyper-exponential distribution is then

$$\overline{T}_1^{HEXP}(\mathbf{g}; \rho) = \frac{\lambda_1}{\lambda_1 + \lambda_2}\overline{T}_1(\mathbf{g}; \rho) + \frac{\lambda_2}{\lambda_1 + \lambda_2}\overline{T}_2(\mathbf{g}; \rho),$$

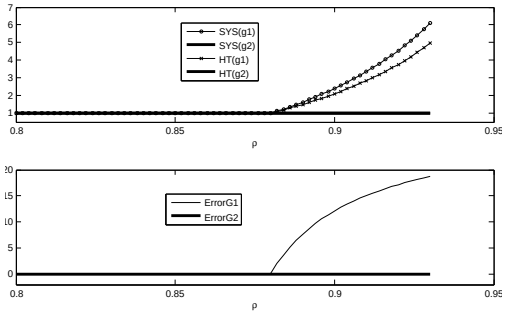

Figure 3.5: Comparison of equilibrium weights (above) and the corresponding percentage relative error (below) as a function of the total system load. $R = 2$ and hyperexponential service time requirements.

and

$$\overline{T}_2^{HEXP}(\mathbf{g};\rho) = \frac{\lambda_3}{\lambda_3+\lambda_4}\overline{T}_3(\mathbf{g};\rho) + \frac{\lambda_4}{\lambda_3+\lambda_4}\overline{T}_4(\mathbf{g};\rho).$$

Using the above trick, the equilibrium weights were computed for the two-player DPS game with parameters: $\mu_1 = 1$, $\mu_2 = 3$, $\mu_3 = 5$, $\mu_4 = 7$, and deadlines $c_1 = 5$ and $c_2 = 7$. The fraction of the load of class 1 was $(\rho_1, \rho_2) = (\frac{\rho}{6}, \frac{\rho}{3})$, and for class 2 it was $(\rho_3, \rho_4) = (\frac{\rho}{4}, \frac{\rho}{4})$. In Figure 3.5 we depict variation of the weights and the relative error when the total load of the system changes. Finally, we observe that the error on the equilibrium is similar to that of the exponentially distributed service times.

3.6.2 Convergence to the Nash Equilibrium

In this section, we analyze the convergence to the Nash equilibrium of the game (OPT-M). In particular, we focus on the dynamics of the users under the best-response algorithm. We consider exponential service times and three classes of users with the following parameters: the load of each class is $(\rho_1, \rho_2, \rho_3) = (0.1, 0.5, 0.2)$, the mean job sizes are given by $(\mu_1, \mu_2, \mu_3) = (1, 2, 3)$ and the deadlines are $(c_1, c_2, c_3) = (2, 2.5, 100)$. As before, we fix the value of ϵ to 1. We are interested in observing the dynamics of the best-response for different not feasible starting points. In the left column of Figure 3.6 the best-response starts from the point $\mathbf{g} = (1, 1, 1)$, in the middle column from $\mathbf{g} = (3, 4, 5)$ and in the right column from $\mathbf{g} = (1, 15, 15)$. In the top subfigure of each column we depict the evolution of the weights over time and in the bottom subfigure the evolution of the mean response times over time. The x-axis of all the figures is in the logarithmic scale for a more clear illustration of the dynamics of the best-response algorithm. We observe that in all the instances the best-response algorithm convergences in at most 200 iterations to the point $(13.4, 2.5, 1)$ which is the Nash equilibrium. We leave the proof of the convergence for future work.

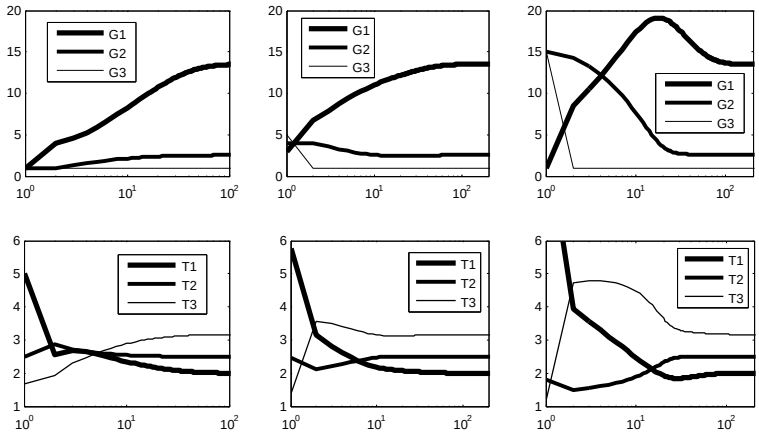

Figure 3.6: The evolution of the weights (up) and the mean response times (down) with the Best Response Algorithm for three different starting points: $\mathbf{g} = (1, 1, 1)$ (left column), $\mathbf{g} = (3, 4, 5)$ (middle column) and $\mathbf{g} = (1, 15, 15)$ (right column). X-axis in logarithmic scale.

3.7 Conclusions

We presented a priced model that studies the strategic behaviour of users sharing the capacity of a processor that operates with relative priorities. Each user chooses a price which corresponds to a priority level and receives a share of the capacity that increases with its payment. The objective of a user is to choose its priority level so as to minimize its own payment, while guaranteeing that its jobs are served before its deadline. We showed that the obtained equilibrium is efficient for two players and general service times. Besides, we fully characterized the solution of this game when the number of players is two and the service time distribution is exponential. We also defined a game in the heavy-traffic regime which we solved for the general instance and we use it as an approximation of the original problem. We proved that the equilibrium of the heavy-traffic game and the approximation we provide are efficient.

We performed several numerical experiments to study the accuracy of the approximated equilibrium. On the one hand, we observed the approximation is accurate when the minimum acceptable throughput of the users is similar. On the other hand, if the heterogeneity of the throughput expectation of the users increases, we concluded that the accuracy of the approximation can diminish. However, we derived that, in all the instances, the heavy-traffic approximation captures the correct structure of the equilibrium and gives us a negligible error in the mean response time prediction.

3.A Appendix of Chapter 3

3.A.1 Proof of Theorem 3.4.1

As in Definition 3.3.1, we let $\mathcal{T}$ be the set of achievable vectors. Define the set

$$\mathcal{U} = \left\{ \mathbf{c} \in \mathbb{R}_+^K : \sum_{i \in r} \rho_i c_i \geq W_r, \ \forall r \in \mathcal{R} \right\}.$$

Before giving a formal proof of Theorem 3.4.1, we briefly explain the main arguments behind the proof. It is easy to see from (3.7)-(3.8) and Definition 3.3.2 that if $\mathbf{c}$ is a feasible vector, then $\mathbf{c} \in \mathcal{U}$. However, the converse is less clear. In order to show that each element $\mathbf{c}$ of $\mathcal{U}$ is a feasible vector, the idea is to construct from $\mathbf{c}$ a vector $\mathbf{t} \in \mathcal{U}$ such that $\mathbf{t} \preceq \mathbf{c}$ and for which $W_{\mathcal{C}} \leq \sum_{i \in \mathcal{C}} \rho_i t_i$ holds as an equality (whereas the inequality can be strict for $\mathbf{c}$). This vector $\mathbf{t}$ is obtained as the limit of a strictly decreasing sequence $\left\{\mathbf{c}^{(n)}\right\}_{n \geq 0}$ which, starting from $\mathbf{c}^{(0)} = \mathbf{c}$, converges in a finite number of steps. The key argument to generate this sequence is that, unless $\mathbf{c}^{(n)} \in \mathcal{T}$, there always exists at least one component of $\mathbf{c}^{(n)}$ that appears only in inequalities. By decreasing this component, we can obtain a vector $\mathbf{c}^{(n+1)} \in \mathcal{U}$ such that $\mathbf{c}^{(n+1)} \prec \mathbf{c}^{(n)}$ and $0 \leq \sum_{i \in \mathcal{C}} \rho_i c_i^{(n+1)} - W_{\mathcal{C}} < \sum_{i \in \mathcal{C}} \rho_i c_i^{(n)} - W_{\mathcal{C}}$, which implies the convergence to an achievable vector. We shall first prove that if $\mathbf{c}$ is not an achievable vector, then there is at least one of its components which is involved only in inequalities. Our first step in this direction is stated in Lemma 3.A.1.

Lemma 3.A.1. *If $r \subseteq s$ then $W_r \leq W_s$.*

Proof. From (3.6),

$$W_s = \frac{1}{1 - \boldsymbol{\rho}_s} \sum_{i \in s} \frac{\rho_i}{\mu_i} \ \geq \frac{1}{1 - \boldsymbol{\rho}_s} \sum_{i \in r} \frac{\rho_i}{\mu_i} \geq \frac{1 - \boldsymbol{\rho}_r}{1 - \boldsymbol{\rho}_s} W_r \ \geq W_r.$$

□

For $\mathbf{c} \in \mathcal{U}$, let us define the sets:

- $\mathcal{S}^{=} = \left\{ r : \sum_{i \in r} \rho_i c_i = W_r \right\}$ and
- $\mathcal{S}^{>} = \left\{ r : \sum_{i \in r} \rho_i c_i > W_r \right\}$.

We have omitted the dependence of the sets on $\mathbf{c}$. The second result we need is the following.

Lemma 3.A.2. *If $r_1, r_2 \in \mathcal{S}^{=}$, then $r_1 \cup r_2 \in \mathcal{S}^{=}$.*

Proof. Let $s = r_1 \cup r_2$ and $v = r_1 \cap r_2$. In order to prove the desired result, we shall show that if $r_1, r_2 \in \mathcal{S}^{=}$ then $W_s \geq \sum_{i \in s} \rho_i c_i$. Since $\mathbf{c} \in \mathcal{U}$, we know that $W_s \leq \sum_{i \in s} \rho_i c_i$. Therefore, the only possible outcome is $W_s = \sum_{i \in s} \rho_i c_i$. From (3.6),

$$\begin{aligned}
W_s &= \frac{1}{1-\boldsymbol{\rho}_s} \sum_{i \in s} \frac{\rho_i}{\mu_i} \\
&= \frac{1}{1-\boldsymbol{\rho}_s} \left(\sum_{i \in r_1} \frac{\rho_i}{\mu_i} + \sum_{i \in r_2} \frac{\rho_i}{\mu_i} - \sum_{i \in v} \frac{\rho_i}{\mu_i} \right) \\
&= \frac{1}{1-\boldsymbol{\rho}_s} \left(\left(1-\boldsymbol{\rho}_{r_1}\right) W_{r_1} + \left(1-\boldsymbol{\rho}_{r_2}\right) W_{r_2} - \left(1-\boldsymbol{\rho}_v\right) W_v \right) \\
&= W_{r_1} + W_{r_2} + \frac{1}{1-\boldsymbol{\rho}_s} \left(\left(\boldsymbol{\rho}_s - \boldsymbol{\rho}_{r_1}\right) W_{r_1} + \left(\boldsymbol{\rho}_s - \boldsymbol{\rho}_{r_2}\right) W_{r_2} - \left(1-\boldsymbol{\rho}_v\right) W_v \right) \\
&= \sum_{i \in r_1} \rho_i c_i + \sum_{i \in r_2} \rho_i c_i + \frac{1}{1-\boldsymbol{\rho}_s} \left(\left(\boldsymbol{\rho}_s - \boldsymbol{\rho}_{r_1}\right) W_{r_1} + \left(\boldsymbol{\rho}_s - \boldsymbol{\rho}_{r_2}\right) W_{r_2} - \left(1-\boldsymbol{\rho}_v\right) W_v \right) \\
&= \sum_{i \in s} \rho_i c_i + \sum_{i \in v} \rho_i c_i + \frac{1}{1-\boldsymbol{\rho}_s} \left(\left(\boldsymbol{\rho}_s - \boldsymbol{\rho}_{r_1}\right) W_{r_1} + \left(\boldsymbol{\rho}_s - \boldsymbol{\rho}_{r_2}\right) W_{r_2} - \left(1-\boldsymbol{\rho}_v\right) W_v \right) \\
&\geq \sum_{i \in s} \rho_i c_i + W_v + \frac{1}{1-\boldsymbol{\rho}_s} \left(\left(\boldsymbol{\rho}_s - \boldsymbol{\rho}_{r_1}\right) W_{r_1} + \left(\boldsymbol{\rho}_s - \boldsymbol{\rho}_{r_2}\right) W_{r_2} - \left(1-\boldsymbol{\rho}_v\right) W_v \right) \\
&\geq \sum_{i \in s} \rho_i c_i + \frac{1}{1-\boldsymbol{\rho}_s} \left(\left(\boldsymbol{\rho}_s - \boldsymbol{\rho}_{r_1}\right) W_{r_1} + \left(\boldsymbol{\rho}_s - \boldsymbol{\rho}_{r_2}\right) W_{r_2} - \left(\boldsymbol{\rho}_s - \boldsymbol{\rho}_v\right) W_v \right).
\end{aligned}$$

In order to complete the proof it is sufficient to show that the second term on the RHS is non-negative, which will then imply that $W_s \geq \sum_{i \in s} \rho_i c_i$. Since $v = r_1 \cap r_2$, from Lemma 3.A.1, it follows that $W_{r_1} \geq W_v$ and $W_{r_2} \geq W_v$. Thus,

$$\left(\boldsymbol{\rho}_s - \boldsymbol{\rho}_{r_1}\right) W_{r_1} + \left(\boldsymbol{\rho}_s - \boldsymbol{\rho}_{r_2}\right) W_{r_2} \geq \left(\boldsymbol{\rho}_s - \boldsymbol{\rho}_{r_1} + \boldsymbol{\rho}_s - \boldsymbol{\rho}_{r_2}\right) W_v = \left(\boldsymbol{\rho}_s - \boldsymbol{\rho}_v\right) W_v,$$

where the last inequality follows from the fact that $\boldsymbol{\rho}_{r_1} + \boldsymbol{\rho}_{r_2} = \boldsymbol{\rho}_s + \boldsymbol{\rho}_v$. □

Corollary 3.A.1. *The set $\mathcal{S}^{=}$ is closed under finite unions.*

We are now in position to prove Theorem 3.4.1.

Proof of Theorem 3.4.1. If $\mathbf{c}$ is feasible then it is easy to see that $\mathbf{c} \in \mathcal{U}$. We now prove that if $\mathbf{c} \in \mathcal{U}$, then $\mathbf{c}$ is feasible. Towards this end, for every $\mathbf{c}$, we shall construct a finite sequence of vectors $\mathbf{c} = \mathbf{c}^{(0)} \succ \mathbf{c}^{(1)} \succ \ldots \succ \mathbf{c}^{(n)}$, with $n \leq R$, $\mathbf{c}^{(i)} \in \mathcal{U}, \forall i$ and $\mathbf{c}^{(n)} \in \mathcal{T}$. Also, n will depend upon $\mathbf{c}$. The vector $\mathbf{c}^{(n)}$ is then an achievable vector which makes $\mathbf{c}$ feasible.

Consider the vector $\mathbf{c}^{(n)}$ obtained at step n. Define the corresponding sets $\mathcal{S}_n^{=}$ and $\mathcal{S}_n^{>}$ which contain the indices of the equalities and the strict inequalities that define $\mathbf{c}^{(n)}$. Also, define $\mathcal{E}_n = \bigcup_{r \in \mathcal{S}_n^{=}} r$,

the set of classes that appear in at least one equality. We shall show that the sequence of $\mathcal{E}_n$ associated to the componentwise decreasing vectors will eventually contain $\mathcal{C}$, and this will happen in a finite number of steps.

If $\mathcal{E}_n = \mathcal{C}$, it follows from Corollary 3.A.1 that $\mathcal{C} \in \mathcal{S}_n^=$, and that $\mathbf{c}^{(n)}$ is achievable. Otherwise, take some $i \in \mathcal{C} \setminus \mathcal{E}_n$, that is, a class which appears only in inequalities.

Define

$$c_i^{(n+1)} = \max_{s \,:\, i \in s} \frac{W_s - \sum_{j \in s, j \neq i} \rho_j c_j^{(n)}}{\rho_i}$$
$$c_j^{(n+1)} = c_j^{(n)}, \ \forall j \neq i.$$

Note that $c_i^{(n+1)} \geq W_{\{i\}}/\rho_i > 0$, and that $c_i^{(n)} > c_i^{(n+1)}$. Therefore $\mathbf{c}^{(n)} \succ \mathbf{c}^{(n+1)}$.

With this definition class i will appear in at least one equality, and this class will be added to $\mathcal{E}_n$. Therefore, $\mathcal{E}_n \subset \mathcal{E}_{n+1}$, and $\mathcal{S}_n^= \subset \mathcal{S}_{n+1}^=$. Since there are R classes, after at most R steps all the classes will appear in at least one equality, that is, there is an $n \leq R$ such that $\mathcal{E}_n = \mathcal{C}$. From Corollary 3.A.1, it follows that $\mathcal{C} \in \mathcal{S}_n^=$, and $\mathbf{c}^{(n)}$ is an achievable vector such that $\mathbf{c}^{(n)} \preceq \mathbf{c}$. □

3.A.2 Proof of Proposition 3.4.1

If $\mathbf{c}$ is achievable, there exists a weight vector $\mathbf{g}$ such that $\overline{T}_i(\mathbf{g}; \rho) = c_i$ for all $i \in \mathcal{C}$. This weight vector is an equilibrium since no class can decrease its weight and still satisfy its constraint. To conclude the proof, it is enough to observe that the weight vector $\theta\,\mathbf{g}$ is such that $\overline{T}_i(\theta\,\mathbf{g}; \rho) = c_i$ for all $i \in \mathcal{C}$ and is thus an equilibrium for any value of $\theta \geq \min\left(\frac{\epsilon}{g_1}, \ldots, \frac{\epsilon}{g_R}\right)$.

We now focus on the case where If $\mathbf{c}$ is not achievable. Assume that there exist two equilibria $\mathbf{g}$ and $\mathbf{h} \neq \mathbf{g}$. If $h_1 = g_1$, then we can assume without loss of generality that $h_2 < g_2$. This implies that $g_2 > \epsilon$, and thus, according to (3.2), that $\overline{T}_2(\mathbf{g}; \rho) = c_2$. Since $\overline{T}_2(\mathbf{g}; \rho)$ is strictly decreasing in g_2, it yields $\overline{T}_2((h_1, h_2); \rho) = \overline{T}_2((g_1, h_2); \rho) > c_2$. Hence, $\mathbf{h}$ is not a feasible point for class 2 and thus cannot be an equilibrium. This is a contradiction, and therefore we cannot have two different equilibria $\mathbf{g}$ and $\mathbf{h}$ such that $h_1 = g_1$.

Assume therefore that $h_1 < g_1$. This implies that $g_1 > \epsilon$, and thus, from (3.2), that $\overline{T}_1(\mathbf{g}; \rho) = c_1$. Since $\overline{T}_1(\mathbf{g}; \rho)$ is strictly decreasing in g_1, $h_1 < g_1$ implies that $\overline{T}_1(\mathbf{g}; \rho) = c_1 < \overline{T}_1((h_1, g_2); \rho)$. However, for $\mathbf{h}$ to be an equilibrium, we need to have $\overline{T}_1((h_1, h_2); \rho) \leq c_1 < \overline{T}_1((h_1, g_2); \rho)$. Since $\overline{T}_1(\mathbf{g}; \rho)$ is increasing in g_2, it yields $h_2 < g_2$, which in turn implies that $g_2 > \epsilon$. The equilibrium $\mathbf{g}$ is therefore such that $g_1 > \epsilon$ and $g_2 > \epsilon$. However, since we have assumed that $\mathbf{c}$ is not achievable, we know that there exists $i \in \{1, 2\}$ such that $\overline{T}_i(\mathbf{g}^{NE}; \rho) < c_i$. According to (3.2), this implies that $g_i = \epsilon$.

This is a contradiction. We thus conclude that we cannot have two different equilibria.

3.A.3 Proof of Proposition 3.4.2

According to the order of the classes, if class 1 is fair, then $c_2\,\mu_2 \geq c_1\,\mu_1 \geq (1-\rho)^{-1}$. Therefore the Processor Sharing weights satisfy both time constraints. The point $\mathbf{g}^{NE} = (\epsilon, \epsilon)$ is clearly the unique Nash equilibrium since both classes have the minimum weight possible and the time constraints are satisfied.

If class 1 is not fair, i.e., $c_1\,\mu_1 < (1-\rho)^{-1}$, then the feasibility of the game implies that $(1-\rho)^{-1} \leq c_2\,\mu_2$. In this case, the equilibrium is achieved in $\mathbf{g} = (g_1, \epsilon)$, where g_1 is such that $\overline{T}_1(\mathbf{g};\rho) = c_1$ and $\overline{T}_2(\mathbf{g};\rho) \leq c_2$. Indeed g_1 is the minimum weight satisfying class-1 time constraint and ϵ is the minimum weight possible for class 2 whose time constraint is satisfied.

From (2.3), it results that

$$\overline{T}_1(\mathbf{g};\rho) = c_1 \iff \frac{g_2}{g_1} = \frac{-\mu_1\rho_2 - \mu_1(1-\rho_1)\left[\mu_1 c_1(1-\rho) - 1\right]}{-\mu_1\rho_2 + \mu_2(1-\rho_2)\left[\mu_1 c_1(1-\rho) - 1\right]},$$

which yields the desired result since $g_2 = \epsilon$.

3.A.4 Proof of Proposition 3.4.3

We first note from (3.7) and (3.9) that for any weight vector $\mathbf{g}$ it holds that

$$\rho_1\overline{T}_1(\mathbf{g};\rho) + \rho_2\overline{T}_2(\mathbf{g};\rho) \leq \rho_1 c_1 + \rho_2 c_2. \tag{3.14}$$

Let $\mathbf{g}^0 = (g_1^0, g_2^0)$ be the starting point of the Best-Response algorithm. If this point satisfies that $\overline{T}_i(\mathbf{g}^0;\rho) \leq c_i$ for $i = 1, 2$, then, as we said in Section 3.3.2, best-response convergences to the equilibrium. Otherwise, (3.14) implies that we have either $\overline{T}_1(\mathbf{g}^0;\rho) > c_1$ or $\overline{T}_2(\mathbf{g}^0;\rho) > c_2$, but not both.

Assume that $\overline{T}_1(\mathbf{g}^0;\rho) > c_1$. Then, the best response of class 1 is to increase its weight to a value g_1^1 such that at point $\mathbf{g}^1 = (g_1^1, g_2^0)$ its constraint $\overline{T}_1(\mathbf{g}^1;\rho) \leq c_1$ is satisfied as an equality. At this point, we have from (3.14) that $\rho_1\overline{T}_1(\mathbf{g}^1;\rho) + \rho_2\overline{T}_2(\mathbf{g}^1;\rho) = \rho_1 c_1 + \rho_2\overline{T}_2(\mathbf{g}^1;\rho) \leq \rho_1 c_1 + \rho_2 c_2$ and thus that $\overline{T}_2(\mathbf{g}^1;\rho) \leq c_2$. We conclude that the weight vector $\mathbf{g}^1$ is feasible. Hence, using Proposition R.2.1, we can claim that the best-response algorithm converges to the equilibrium.

3.A.5 Proof of Proposition 3.5.1

It can be easily proven that if a vector of performance $\mathbf{t}$ is achievable in heavy-traffic then it satisfies (3.10). For the other implication, we show that a vector $\mathbf{t} \in \mathbb{R}_+^R$ satisfying (3.10) is achievable in heavy-traffic, i.e., there exists a vector of weights $\mathbf{g}$ such that

$\overline{T}_i(\mathbf{g};1) = t_i$ for all $i \in \mathcal{C}$. Let $\mathbf{g}$ be a weight vector such that $\frac{g_i}{g_j} = \frac{t_j/\mathbb{E}(B_j)}{t_i/\mathbb{E}(B_i)}$ for all $i \neq j$. With (2.8), we have

$$\overline{T}_i(\mathbf{g};1) = \mathbb{E}(B_i)\frac{\sum_k \lambda_k \mathbb{E}\left(B_k^2\right)}{\sum_k \lambda_k \mathbb{E}\left(B_k^2\right)\frac{g_i}{g_k}} = \mathbb{E}(B_i)\frac{\sum_k \lambda_k \mathbb{E}\left(B_k^2\right)}{\sum_k \lambda_k \mathbb{E}\left(B_k^2\right)\frac{t_k/\mathbb{E}(B_k)}{t_i/\mathbb{E}(B_i)}} = t_i\frac{\sum_k \lambda_k \mathbb{E}\left(B_k^2\right)}{\sum_k \lambda_k \frac{\mathbb{E}\left(B_k^2\right)}{\mathbb{E}(B_k)}t_k} = t_i,$$

for all $i \in \mathcal{C}$, where the last inequality follows from (3.10). We thus conclude that the vector $\mathbf{t}$ is achievable.

3.A.6 Proof of Proposition 3.5.2

If the problem is feasible in heavy-traffic there exists an achievable vector in heavy-traffic $\mathbf{t} = (t_1, \ldots, t_R)$ such that $t_i \leq \tilde{c}_i$, for all i. Then, since $t_i \leq \tilde{c}_i$ for all i, it follows from Proposition 3.5.1 that $\sum_i \lambda_i \frac{\mathbb{E}\left(B_i^2\right)}{\mathbb{E}(B_i)}\tilde{c}_i \geq \sum_k \lambda_k \mathbb{E}\left(B_k^2\right)$.

We now focus on the other implication of the proposition. Given a vector of deadlines $\tilde{\mathbf{c}} = (\tilde{c}_1, \ldots, \tilde{c}_R)$ such that $\sum_k \lambda_k \frac{\mathbb{E}\left(B_k^2\right)}{\mathbb{E}(B_k)}\tilde{c}_k \geq \sum_k \lambda_k \mathbb{E}\left(B_k^2\right)$, we show that there exists a vector of performances $\mathbf{t}$ achievable in heavy-traffic. Let $\mathbf{t} = (t_1, \ldots, t_R)$ be such that

$$t_i = \tilde{c}_i \frac{\sum_k \lambda_k \mathbb{E}\left(B_k^2\right)}{\sum_k \lambda_k \frac{\mathbb{E}\left(B_k^2\right)}{\mathbb{E}(B_k)}\tilde{c}_k},$$

for all i. We observe that t_i is positive for all i and from $\sum_k \lambda_k \frac{\mathbb{E}\left(B_k^2\right)}{\mathbb{E}(B_k)}\tilde{c}_k \geq \sum_k \lambda_k \mathbb{E}\left(B_k^2\right)$ we derive that $t_i \leq \tilde{c}_i$ for all i. Moreover

$$\sum_k \lambda_k \frac{\mathbb{E}\left(B_k^2\right)}{\mathbb{E}(B_k)}t_k = \sum_k \lambda_k \frac{\mathbb{E}\left(B_k^2\right)}{\mathbb{E}(B_k)}\tilde{c}_i \frac{\sum_i \lambda_i \mathbb{E}\left(B_i^2\right)}{\sum_i \lambda_i \frac{\mathbb{E}\left(B_i^2\right)}{\mathbb{E}(B_i)}\tilde{c}_i} = \sum_i \lambda_i \mathbb{E}\left(B_i^2\right),$$

and we thus conclude with Proposition 3.5.1 that the vector $\mathbf{t}$ is achievable.

3.A.7 Proof of Theorem 3.5.1

Let us first introduce some results that will be used to prove Theorem 3.5.1. Let $\mathbf{g}^m$ be a vector of the form

$$\mathbf{g}^m = (g_1^m, g_2^m, \ldots, g_{m-1}^m, \epsilon, \ldots, \epsilon), \tag{3.15}$$

where $g_i^m > \epsilon$, if $i < m$. We now show the following property of the vector $\mathbf{g}^m$.

Lemma 3.A.3. *If $\overline{T}_m(\mathbf{g}^m;1) \leq \tilde{c_m}$, then, for all $j > m$, $\overline{T}_j(\mathbf{g}^m;1) \leq \tilde{c}_j$.*

Proof. From (2.8) and $\overline{T}_m(\mathbf{g}^m;1) \leq \tilde{c_m}$, we obtain for all $j > m$, $\frac{\sum_k \lambda_k \mathbb{E}\left(B_k^2\right)}{\sum_k \lambda_k \mathbb{E}\left(B_k^2\right)/g_k} \leq \tilde{c_m} g_m^m/\mathbb{E}(B_m) = \tilde{c_m}\ \epsilon/\mathbb{E}(B_m) \leq \tilde{c_j}\ \epsilon/\mathbb{E}(B_j)$, where the last inequality holds since the

ordering of the classes we assume. We now notice that the result follows directly from (2.8) since $\frac{\sum_k \lambda_k \mathbb{E}(B_k^2)}{\sum_k \lambda_k \mathbb{E}(B_k^2)/g_k} \leq \tilde{c_j}\ \epsilon/\mathbb{E}(B_j) \iff \overline{T}_j(\mathbf{g};1) \leq \tilde{c_j}$. □

We are now in position to prove the result of Theorem 3.5.1.

Proof of Theorem 3.5.1. Let m be the minimum value such that $\overline{T}_m(\mathbf{g}^m;1) \leq \tilde{c_m}$, where $\mathbf{g}^m$ is as defined in (3.15). According to Lemma 3.A.3, we have that $\overline{T}_k(\mathbf{g}^m;1) \leq \tilde{c_k}$, for $k \geq m$. On the other hand, we choose g_k such that $\overline{T}_k(\mathbf{g}^m;1) = \tilde{c_k}$ for all $k < m$. It then results that $\mathbf{g}^m$ is the equilibrium since in case any of the first $m-1$ coordinates of $\mathbf{g}^m$ diminishes its weight its time constraint is not satisfied and the rest of the coordinates of $\mathbf{g}^m$ are ϵ.

We now characterize the first $m-1$ components of the equilibrium. From (2.8), it follow that $\frac{g_i^m}{g_j^m} = \frac{\overline{T}_j(\mathbf{g}^m;1)/\mathbb{E}(B_j)}{\overline{T}_i(\mathbf{g}^m;1)/\mathbb{E}(B_i)}$ for all $i \neq j$. Since $\overline{T}_i(\mathbf{g}^m;1) = \tilde{c_i}$ for all $i < m$, we can state that for all $i < m$

$$\frac{g_i^m}{g_m^m} = \frac{\tilde{t_m}/\mathbb{E}(B_m)}{\tilde{c_i}/\mathbb{E}(B_i)} \iff g_i^m = \epsilon\ \frac{\tilde{t_m}/\mathbb{E}(B_m)}{\tilde{c_i}/\mathbb{E}(B_i)}.$$

Finally, we prove that $\overline{T}_m(\mathbf{g}^m;1) = \tilde{t_m} \leq \tilde{c_m}$ is equivalent to (3.11). Using (2.8), we obtain

$$\begin{aligned}
\tilde{c_m} \geq \tilde{t_m} &= \mathbb{E}(B_m)\frac{\sum_{k=1}^R \lambda_k \mathbb{E}\left(B_k^2\right)}{\sum_{k=1}^R \lambda_k \mathbb{E}\left(B_k^2\right)\frac{g_m^m}{g_k^m}} \\
&= \mathbb{E}(B_m)\frac{\sum_{k=1}^R \lambda_k \mathbb{E}\left(B_k^2\right)}{\sum_{k=1}^{m-1} \lambda_k \mathbb{E}\left(B_k^2\right)\frac{\tilde{c_k}/\mathbb{E}(B_k)}{\tilde{t_m}/\mathbb{E}(B_m)} + \sum_{k=m}^R \lambda_k \mathbb{E}\left(B_k^2\right)}.
\end{aligned}$$

And rearranging both sides of the equation we derive the expression (3.11)

$$\frac{\tilde{t_m}}{\mathbb{E}(B_m)} = \frac{\sum_{k=1}^R \lambda_k \mathbb{E}\left(B_k^2\right) - \sum_{k=1}^{m-1} \lambda_k \frac{\mathbb{E}\left(B_k^2\right)}{\mathbb{E}(B_k)} c_k}{\sum_{k=m}^R \lambda_k \mathbb{E}\left(B_k^2\right)}.$$

We now show this equilibrium is unique proving that if the equilibrium is $\mathbf{g^m}$, then $\mathbf{g^{m+i}}$ is not the equilibrium, for $i = 1, \ldots, R-m$. We thus consider that there exists a value m satisfying

$$\frac{\tilde{c_m}}{\mathbb{E}(B_m)} \geq \frac{t_m}{\mathbb{E}(B_m)} = \frac{\sum_{k=1}^R \lambda_k \mathbb{E}\left(B_k^2\right) - \sum_{k=1}^{m-1} \lambda_k \frac{\mathbb{E}\left(B_k^2\right)}{\mathbb{E}(B_k)} \tilde{c_k}}{\sum_{k=m}^R \lambda_k \mathbb{E}\left(B_k^2\right)},$$

which is equivalent to

$$\frac{t_m}{\mathbb{E}(B_m)}\sum_{k=m}^{R}\lambda_k\mathbb{E}(B_k^2)=\sum_{k=1}^{R}\lambda_k\mathbb{E}\left(B_k^2\right)-\sum_{k=1}^{m-1}\lambda_k\frac{\mathbb{E}\left(B_k^2\right)}{\mathbb{E}(B_k)}\tilde{c_k}. \tag{3.16}$$

We will see that for any $i=1,\ldots,R-m$, $\mathbf{g^{m+i}}$ that satisfies (3.11) is not the equilibrium.

We suppose that there exist a value $i=1,\ldots,R-m$ such that

$$\frac{\tilde{c_{m+i}}}{\mathbb{E}(B_{m+i})}\geq\frac{\overline{t_{m+i}}}{\mathbb{E}(B_{m+i})}=\frac{\sum_{k=1}^{R}\lambda_k\mathbb{E}\left(B_k^2\right)-\sum_{k=1}^{m+i-1}\lambda_k\frac{\mathbb{E}\left(B_k^2\right)}{\mathbb{E}(B_k)}\tilde{c_k}}{\sum_{k=m+i}^{R}\lambda_k\mathbb{E}\left(B_k^2\right)}, \tag{3.17}$$

is verified.

It thus follows that

$$\frac{\tilde{c_{m+i}}}{\mathbb{E}(B_{m+i})}\geq\frac{\overline{t_{m+i}}}{\mathbb{E}(B_{m+i})}=\frac{\sum_{k=1}^{R}\lambda_k\mathbb{E}\left(B_k^2\right)-\sum_{k=1}^{m-1}\lambda_k\frac{\mathbb{E}\left(B_k^2\right)}{\mathbb{E}(B_k)}\tilde{c_k}-\sum_{k=m}^{m+i-1}\lambda_k\frac{\mathbb{E}\left(B_k^2\right)}{\mathbb{E}(B_k)}\tilde{c_k}}{\sum_{k=m+i}^{R}\lambda_k\mathbb{E}\left(B_k^2\right)}.$$

Taking into account the equality of (3.16) and that $\frac{\tilde{t_m}}{\mathbb{E}(B_m)}\leq\frac{\tilde{c_m}}{\mathbb{E}(B_m)}\leq\frac{\tilde{c_k}}{\mathbb{E}(B_k)}$ for all $k>m$, we derive

$$\begin{aligned}\frac{\overline{t_{m+i}}}{\mathbb{E}(B_{m+i})} &= \frac{\frac{t_m}{\mathbb{E}(B_m)}\sum_{k=m}^{R}\lambda_k\mathbb{E}(B_k^2)-\sum_{k=m}^{m+i-1}\lambda_k\frac{\mathbb{E}\left(B_k^2\right)}{\mathbb{E}(B_k)}\tilde{c_k}}{\sum_{k=m+i}^{R}\lambda_k\mathbb{E}\left(B_k^2\right)}\\ &\leq \frac{\frac{\tilde{c_m}}{\mathbb{E}(B_m)}\sum_{k=m+i}^{R}\lambda_k\mathbb{E}(B_k^2)}{\sum_{k=m+i}^{R}\lambda_k\mathbb{E}\left(B_k^2\right)}=\frac{\tilde{c}_m}{\mathbb{E}(B_m)}.\end{aligned}$$

From the relation $\frac{g_k}{g_j}=\frac{\overline{T}_j(\mathbf{g};\rho)/\mathbb{E}(B_j)}{\overline{T}_k(\mathbf{g};\rho)/\mathbb{E}(B_k)}$ and using that $\overline{T}_m(\mathbf{g}^{m+i};\rho)=\tilde{c_m}$ and $\overline{T}_{m+i}(\mathbf{g}^{m+i};\rho)=\overline{t_{m+i}}$ if $\mathbf{g}^{m+i}$ is an equilibrium, we obtain that

$$\frac{\overline{t_{m+i}}}{\mathbb{E}(B_{m+i})}\leq\frac{\tilde{c}_m}{\mathbb{E}(B_m)}\iff g_{m+i}^{m+i}\geq g_m^{m+i}$$

which is not possible since $g_{m+i}^{m+i}=\epsilon$ and $g_m^{m+i}>\epsilon$ if $\mathbf{g}^{m+i}$ is an equilibrium. □

4

Non-cooperative Load Balancing

We analyze the performance of non-cooperative decentralized routing in server farms. In this system, each dispatcher receives a portion of the total traffic and routes it to the servers so as to minimize the mean response times of its jobs. For this setting, a non-cooperative game can be formulated. We aim to compare the performance in the equilibrium of this game with the performance of a system with a unique dispatcher.

This chapter is organized as follows. We introduce the problem in Section 4.1 and we present the related work in Section 4.2. In Section 4.3 we describe the model. In Section 4.4 we investigate the worst case traffic conditions. In Section 4.5, we give more precise results for server farms with two classes of servers. We study the *inefficiency* of a server farm with an infinite number of dispatchers in Section 4.6. Finally, the main conclusions of this work are presented in Section 4.7. Some of the proofs of this model are given in Appendix 4.A.

4.1 Efficiency of Non-Cooperative Load Balancing

In Chapter 3, we analyzed a resource-sharing game in a single server. In this chapter, we analyze the resource-sharing in server farms, which are multi-server systems with a variety of applications, such as cluster computing, web hosting or scientific simulation. A fundamental problem to optimize the performance of a server farm is how to balance the load over the servers[1].

We first present a a centralized architecture for load-balancing in server farms. We

[1] We shall use the terms load balancing and routing interchangeably.

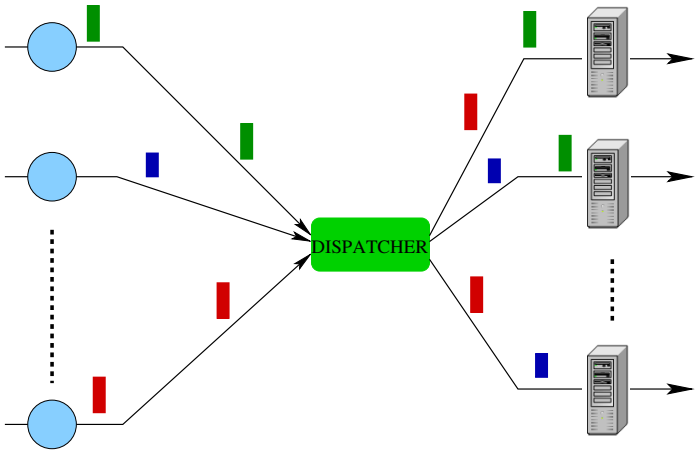

Figure 4.1: centralized system for a server farm.

represent in Figure 4.1 the centralized architecture we consider. In this figure, the users are the blue circles that are on the left and want to send data to the servers, that are situated on the right. We observe that there is only one dispatcher between the set of clients and the set of servers. Besides, the dispatcher receives all the data of the clients and routes the traffic to the servers so as to optimize a certain performance objective, such as the mean processing time of jobs for instance. The difficulty of this optimization technique increase with the number of servers and, modern data centers have thousands of processors. For instance, Akamai Technologies, in March 2012, operated with 105,000 servers [2] and it is estimated that Google has more than 900,000 servers [1]. Furthermore, the size of server-farms is still increasing and thus we can say that centralized architectures are not implementable in current server farms. Decentralized architectures have emerged as a solution to this problem.

In Figure 4.2, we present the decentralized architecture. We observe that in this setting, there is more than one dispatcher between the clients and the servers, and each dispatcher receives a portion of the traffic.

In the literature, several approaches have been suggested to implement decentralized routing. Approaches based on distributed optimization techniques have been presented in [16, 66].

We consider a different approach that is based on autonomous, selfish agents [78]. In these systems, each dispatcher is independent and optimizes the performance of the traffic it receives. For this system, a non-cooperative game can be formulated and thus it can be analyzed using game-theoretical techniques. Rational agents choose a strategy under these circumstances which is called a Nash equilibrium.

We remark that the decentralized approach we have just described is more scalable that the centralized architecture, since each dispatcher controls a portion of the incoming traffic. Furthermore, non-cooperative systems have a wide range of advantages

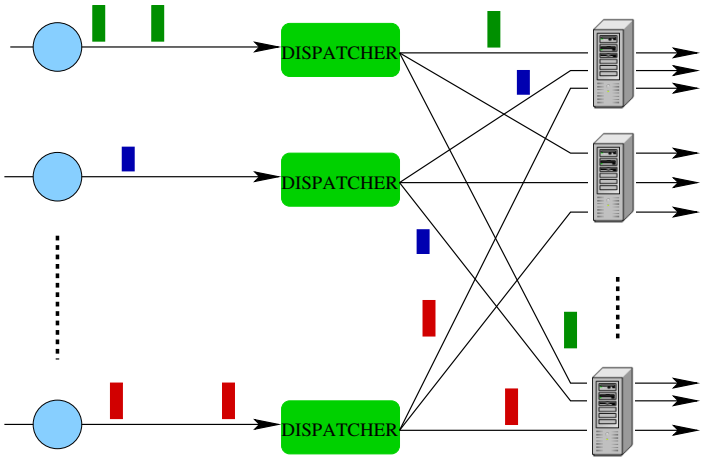

Figure 4.2: Decentralized system for a server farm.

with respect to centralized ones. For example, the selfish routing does not need any coordination among the dispatchers. This means that this setting is simple to implement and that it is robust to failures. However, it is shown in the literature that the loss of performance given by the selfish behaviour of the dispatcher can be high. The inefficiency of selfish routing is measured in the literature with the notion of Price of Anarchy (PoA).

We analyze the inefficiency of non-cooperative load balancing and we show that the PoA is a very pessimistic measure to quantify the loss in performance caused by selfish routing. Besides, we show that the PoA can achieve a large value but this value is observed only in instances that hardly occur in practice. For example, in [52], the worst-case architecture has one server whose capacity is much larger (tending to infinity) compared to that of the other servers. It is doubtful that such asymmetries will occur in data-centers where processors are more than likely to have similar characteristics.

In a data-center, the speed of the servers cannot be modified in general, whereas the incoming traffic cannot be controlled by the service provider and changes over time. Thus, for this kind of applications, it makes more sense to analyze the performance of selfish routing with respect to the centralized system for a fixed data-center architecture, i.e., fixed values of the speeds of the servers. Hence, we define the inefficiency of a given data center architecture as the worst-case, among all possible traffic distributions, of the ratio between the performance in the Nash equilibrium and the performance of the centralized architecture. We observe that the values of the inefficiency are between 1 and infinity.

Let us now explain the difference between the PoA and the inefficiency. The inefficiency is the ratio between the performances of the decentralized and centralized settings, in the worst case traffic conditions when the capacities of the servers are fixed, whereas the PoA is the same ratio, but taken for the worst possible combination of traf-

fic conditions and server speeds. Hence, by calculating the worst possible inefficiency, one retrieves the PoA.

We now explain the main contributions of this work. For an arbitrary capacities of the servers, we analyze the traffic conditions (or load) associated with the inefficiency. We show that the inefficiency is in general not achieved in heavy-traffic or close to saturation conditions. Even more, we show that the performance of the decentralized and of the centralized routing coincides in the heavy-traffic regime, i.e., when the load of the system approaches to one. We consider that this result is interesting since, in queueing theory, we obtain a bad performance in heavy traffic. For example, in a M/M/1 queue, the mean response time tends to infinity when the load goes to one. We also provide examples for which the inefficiency is obtained for fairly low values of the utilization rate.

We then observe that for each problem instance with arbitrary server capacities we can construct a scenario with two server speeds whose inefficiency is worse. We thus conjecture that the case of two classes of servers is the worst, and that our conclusions on the efficiency of non-cooperative load balancing extend beyond this case.

In the case of two server classes, we show that the inefficiency is obtained when selfish routing uses only one class of servers and is marginally using the second class of servers. This scenario was used in [52] and [14] to obtain a lower bound on the PoA for their models. Further, we obtain a closed-form formula for the inefficiency which in particular depends only on the ratio of the number of servers in each class and on the ratio of the capacities of each class (but not on the total nor on their capacities). When the number of servers is large, we also show that the PoA is equal to $\frac{K}{2\sqrt{K-1}}$, where K is the number of dispatchers. We then show that the inefficiency is very close to 1 in most cases of two server classes, and that it approaches the known upper bound (given by the PoA) in a very specific setting, namely, when there is only one fast server that is infinitely faster than the slower ones.

We also investigate the case where the number of dispatchers is infinite. For an arbitrary server capacities, we show that the performance of the decentralized and the centralized settings are not equal in the heavy-traffic regime, in contrast with the case of finite number of dispatchers. For the case of two servers classes, we give an expression of the inefficiency that depends only on the ratio of the number of servers in each class and on the ratio of the capacities of each class. We show that, for a server farm with two different speeds, the PoA equals the number of servers, which coincides with the result presented in [52], and that it is achieved only when there is one fast server that is infinitely faster than the slower ones. In all the other configurations, we observe that the inefficiency is very close to one.

Even though most of the results are given for a server farm with two different speeds, we believe that our work opens a new avenue in the study of the PoA and we hope that future research will be done not only on the PoA, but also on the inefficiency.

4.2 Related Work

A fundamental problem in networking is to determine how the traffic must be routed. It is well known, see [72, 77, 67], that users that act according to their own interests can lead to non optimal results. This phenomenon given by uncoordinated users has been widely studied in transportation theory and communication networks, see for example [69, 81, 38] and it is known in the literature as selfish routing problem or non-cooperative routing game. In this work, we analyze the efficiency of the equilibrium obtained in an atomic splittable routing game.

There are many results regarding the optimality of load balancing models in server farms. If the state (number of customers) of all the servers is available by the routing agent, the optimality is given by the Join the Shortest Queue policy. The study of the Join the Shortest Queue is know to be more a difficult task. In fact, only approximations have been proposed in the literature for the general case, see for example [37, 45]. The Power of Two model is shown to give an exponential improvement when the number of servers tends to infinity. The analysis of the Power of Two discipline in a general setting remains open and we refer to [75] for a survey on this topic. In [47, 27] the authors propose and analyze the use of the Size Interval task Assignment policy. If the service requirement of incoming jobs is known, the size interval task assignment has been proven to be optimal in [36].

We investigate the non-cooperative routing in server farms when the number of customers in the servers is unknown by the players, i.e., the load balancing in unobservable queues. In the last years researchers have extensively studied load balancing models with unobservable queues. In [12] the authors consider that, for each incoming job, the previous routing decision is taken into account to perform the load balancing task. Interesting load balancing models in the non-atomic case are also analyzed in [11, 22, 9]. In [11] the authors analyze the equilibrium in the oligopolistic price competition model achieved by the interaction between users and provider. The authors in [22] study the inefficiency of the equilibrium obtained in the non-atomic load balancing games when the scheduling of the servers are PS and Shortest Remaining Processing Time. Finally, in [9], the authors consider the Bernoulli splitting policy and PS queues for the non-atomic load balancing games. The equilibrium for the case of infinite dispatchers we give in Section 4.6 coincides with the one given in [9], however they only analyze the PoA and not the inefficiency.

The efficiency of the equilibrium of the load balancing model with non observable queues has been studied in the presence of non-linear delay functions, see, for example, [52], [15], [14], [86] and [22]. We now summarize the previous upper-bounds on the Price of Anarchy load balancing games. In [52] the authors show that the PoA of the non atomic case is less than the number of servers. On the contrary, the authors in [14] focus on the atomic case and they prove that the PoA is less than the square root of the number of dispatchers. According to this result, the value of the PoA is very large when

the number of servers and the number of dispatchers is high. The fact that the Nash equilibrium can be very inefficient has paved the way to a lot of research on mechanism design that aims at coming up with Nash equilibria that are efficient with respect to the centralized setting [61, 60, 80, 23, 78]. In this work we show that even though the value of the PoA can be high, the difference between the equilibrium performance and the centralized performance is small in most instances.

4.3 Problem Formulation

We consider a non-cooperative routing game with K dispatchers and S Processor-Sharing servers. Denote $\mathcal{C} = \{1, \ldots, K\}$ to be the set of dispatchers and $\mathcal{S} = \{1, \ldots, S\}$ to be the set of servers. Jobs received by dispatcher i are said to be jobs of stream i.

Server $j \in \mathcal{S}$ has capacity r_j. It is assumed that servers are numbered in the order of decreasing capacity, i.e., if $m \leq n$, then $r_m \geq r_n$. Let $\mathbf{r} = (r_j)_{j \in \mathcal{S}}$ denote the vector of server capacities and let $\overline{R} = \sum_{n \in \mathcal{S}} r_n$ denote the total capacity of the system.

Jobs of stream $i \in \mathcal{C}$ arrive to the system according to a Poisson process and have generally distributed service-times. We do not specify the arrival rate and the characteristics of the service-time distribution due to the fact that in an $M/G/1 - PS$ queue the mean number of jobs depends on the arrival process and service-time distribution only through the traffic intensity, i.e., the product of the arrival rate and the mean service-time. Let λ_i be the traffic intensity of stream i. It is assumed that $\lambda_i \leq \lambda_j$ for $i \leq j$. Moreover, it will also be assumed that the vector $\boldsymbol{\lambda}$ of traffic intensities belongs to the following set:

$$\Lambda(\bar{\lambda}) = \left\{ \boldsymbol{\lambda} \in \mathbb{R}^K \ : \ \sum_{i \in \mathcal{C}} \lambda_i = \bar{\lambda} \right\},$$

where $\bar{\lambda}$ denotes the total incoming traffic intensity. It is assumed that $\bar{\lambda} < \overline{R}$, which is the necessary and sufficient condition to guarantee the stability of the system. We denote by $\rho = \frac{\bar{\lambda}}{\overline{R}}$ the total traffic of the system.

We will sometimes be interested in what happens when $\bar{\lambda} \to \overline{R}$, a regime which we will refer to as heavy-traffic ($\rho \to 1$).

Let $\mathbf{x}_i = (x_{i,j})_{j \in \mathcal{S}}$ denote the routing strategy of dispatcher i, with $x_{i,j}$ being the amount of traffic it sends towards server j. Dispatcher i seeks to find a routing strategy that minimizes the mean sojourn times of its jobs, which, by Little's law, is equivalent to minimizing the mean number of jobs in the system as seen by this stream. This optimization problem can be formulated as follows:

$$\text{minimize } T_i(\mathbf{x}) = \sum_{j\in\mathcal{S}} \frac{x_{i,j}}{r_j - y_j} \tag{ROUTE-i}$$

$$\text{subject to } \sum_{j\in\mathcal{S}} x_{i,j} = \lambda_i, \quad i = 1,\dots,K, \tag{4.1}$$

$$\text{and } \ 0 \le x_{i,j} \le r_j, \quad \forall j \in \mathcal{S}, \tag{4.2}$$

where $y_j = \sum_{k\in\mathcal{C}} x_{k,j}$ is the traffic offered to server j. Note that the optimization problem solved by dispatcher i depends on the routing decisions of the other dispatchers since $y_j = x_{i,j} + \sum_{k\neq i} x_{k,j}$. We let $\mathcal{X}_i$ denote the set of feasible routing strategies for dispatcher i, i.e., the set of routing strategies satisfying constraints (4.1)-(4.2). A vector $\mathbf{x} = (\mathbf{x}_i)_{i\in\mathcal{C}}$ belonging to the product strategy space $\mathcal{X} = \bigotimes_{i\in\mathcal{C}} \mathcal{X}_i$ is called a strategy profile.

A Nash equilibrium of the routing game is a strategy profile from which no dispatcher finds it beneficial to deviate unilaterally. Hence, $\mathbf{x} \in \mathcal{X}$ is a Nash Equilibrium Point (NEP) if $\mathbf{x}_i$ is an optimal solution of problem (ROUTE-i) for all dispatcher $i \in \mathcal{C}$.

Let $\mathbf{x}$ be a NEP for the system with K dispatchers. The global performance of the system can be assessed using the global cost

$$D_K(\boldsymbol{\lambda}, \mathbf{r}) = \sum_{i\in\mathcal{C}} T_i(\mathbf{x}) = \sum_{j\in\mathcal{S}} \frac{y_j}{r_j - y_j},$$

where the offered traffic y_j are those at the NEP. The above cost represents the mean number of jobs in the system. Note that when there is a single dispatcher, we have a single dispatcher with $\lambda_1 = \bar{\lambda}$. The global cost can therefore be written as $D_1(\bar{\lambda}, \mathbf{r})$ in this case.

We shall use the ratio between the performance obtained by the Nash equilibrium and the global optimal solution as a metric in order to assess the *inefficiency* of a decentralized scheme with K dispatchers and S servers. We define the *inefficiency* as the performance ratio under the worst possible traffic conditions, namely:

$$\textit{inefficiency } I_K^S(\mathbf{r}) = \sup_{\boldsymbol{\lambda}\in\Lambda(\bar{\lambda}),\bar{\lambda}<\overline{R}} \frac{D_K(\boldsymbol{\lambda}, \mathbf{r})}{D_1(\bar{\lambda}, \mathbf{r})}. \tag{4.3}$$

The rationale for this definition is that in practice the system administrator controls neither the total incoming traffic nor how it is split between the dispatchers, whereas the number of servers and their capacities are fixed. Therefore it makes sense to consider the worst traffic conditions for the *inefficiency* of selfish routing, provided the system is stable.

The PoA for this system as defined in [14] can be retrieved by looking at the worst

inefficiency, i.e.,

$$PoA(K,S) = \sup_{\mathbf{r}} I_K^S(\mathbf{r}). \tag{4.4}$$

4.4 Worst Case Traffic Conditions

In this section, we show that for a sufficiently large load, the ratio $\frac{D_K(\boldsymbol{\lambda},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})}$ decreases with $\bar{\lambda}$. This result implies that the *inefficiency* of a data-center is not achieved in the heavy-traffic regime. Moreover, we also prove that when the system is in heavy-traffic, i.e., when $\bar{\lambda} \to \overline{R}$, the performance of both settings is the same.

The main difficulty in determining the behaviour of the *inefficiency* stems from the fact that for most cases there are no easy-to-compute explicit expressions for the NEP. A first simplification results from the following theorem which was proved in one of our previous works [14]. It states that, among all traffic vectors with total traffic intensity $\bar{\lambda}$, the global cost $D_K(\boldsymbol{\lambda},\mathbf{r})$ achieves its maximum when all dispatchers control the same fraction of the total traffic. Formally,

Theorem 4.4.1 ([14])**.**

$$D_K(\boldsymbol{\lambda},\mathbf{r}) \leq D_K(\frac{\bar{\lambda}}{K}\,\mathbf{e},\mathbf{r}). \quad \forall \boldsymbol{\lambda} \in \Lambda(\bar{\lambda}).$$

where $\mathbf{e}$ *is the all-ones vector.*

Thus, we have identified the traffic vector in the set $\Lambda(\bar{\lambda})$ which has the worst-ratio of global cost at the NEP to the global optimal cost. It follows from the above result that

Corollary 4.4.1.

$$I_K^S(\mathbf{r}) = \sup_{\bar{\lambda}<\overline{R}} \frac{D_K(\frac{\bar{\lambda}}{K}\,\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})}.$$

Routing games in which players have exactly the same strategy set are known as *symmetric* games. These games belong to the class of *potential games* [65], that is, they have the property that there exists a function, called the *potential* such that the NEP can be obtained as the solution of an optimization problem with the *potential* as the objective. This property considerably simplifies the computation of the NEP. Another important consequence of the above results is that the *inefficiency* depends only the total traffic intensity and not on individual traffic flows to each of the dispatchers.

Another consequence of Theorem 4.4.1 is that the inefficiency of decentralized routing increases with the number of dispatchers, that is,

Lemma 4.4.1.

$$I_K^S(\mathbf{r}) \leq I_{K+1}^S(\mathbf{r}), \quad \forall K \geq 1.$$

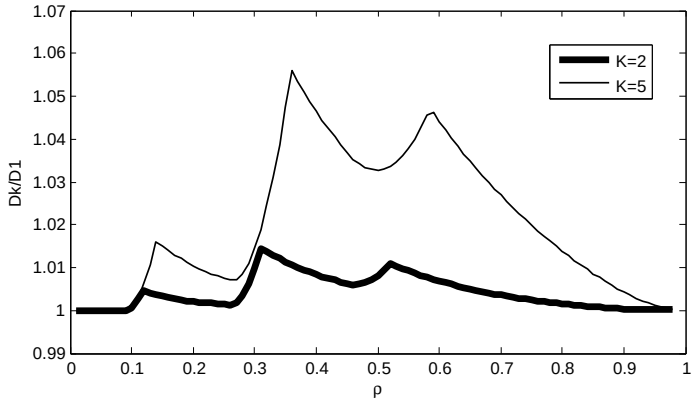

Figure 4.3: Evolution of the ratio $D_K(\frac{\bar{\lambda}}{K}\,\mathbf{e},\mathbf{r})/D_1(\bar{\lambda},\mathbf{r})$ for $K=2$ and $K=5$ as the load in the system ranges from 0% to 100%.

Proof. We have for all $\bar{\lambda} < \overline{R}$,

$$D_K(\frac{\bar{\lambda}}{K}\,\mathbf{e},\mathbf{r}) = D_{K+1}(\left(\frac{\bar{\lambda}}{K}\,\mathbf{e},0\right),\mathbf{r}) \leq D_{K+1}(\frac{\bar{\lambda}}{K+1}\,\mathbf{e},\mathbf{r}),$$

where the last inequality follows from Theorem 4.4.1. It yields

$$\sup_{\bar{\lambda}<\overline{R}} \frac{D_K(\frac{\bar{\lambda}}{K}\,\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})} \leq \sup_{\bar{\lambda}<\overline{R}} \frac{D_{K+1}(\frac{\bar{\lambda}}{K+1}\,\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})},$$

i.e., $I_K(\mathbf{r}) \leq I_{K+1}(\mathbf{r})$. □

Before going further, let us take a look at the ratio $\frac{D_K(\frac{\bar{\lambda}}{K}\,\mathbf{e},\mathbf{r})}{D_1(\lambda,\mathbf{r})}$ as a function of the load $\rho = \bar{\lambda}/\overline{R}$, as is shown in Figure 4.3 for two and five dispatchers. The data-center characteristics are the following: 200 servers of speed 6, 100 servers of speed 3, 300 servers of speed 2, and 200 servers of speed 1. It can be observed that as the load increases the ratio goes through peaks and valleys, and finally it moves towards 1 as the load approaches 1. In the numerical experiments, we noted that the peaks corresponded to the total traffic intensity when selfish routing started to use one more class of servers. Moreover, just after these peaks the number of servers used by selfish routing and the centralized one was the same. A similar behaviour is observed on different sets of experiments.

In general, it is not easy to make formal the above observation, that is to say, there are no simple expressions for the value of loads which corresponds to the peaks and the valleys. However, in heavy-traffic, it helps to observe that both selfish and centralized routing will be using the same number of servers. Then, in order to show that heavy-traffic conditions are not inefficient, it is sufficient to show that the ratio decreases with load when both settings use the same number of servers.

Proposition 4.4.1. *If the total traffic intensity $\bar{\lambda}$ is such that centralized and the decentralized settings use the same number of servers (more than one), then the ratio $D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})/D_1(\bar{\lambda}, \mathbf{r})$ is decreasing with $\bar{\lambda}$.*

Proof. See Appendix 4.A.2. □

In the above result we exclude the case of one server so as to obtain a stronger result. If both the settings use just one server, then the ratio remains 1, which is non-increasing. For a sufficiently high load all the servers will be used by both settings in order to guarantee the stability of the system. It then follows that in a server farm with an arbitrary number of servers and with arbitrary server capacities, heavy-traffic regime is not inefficient. In fact, we can prove a stronger result which states that the *inefficiency* of the heavy-traffic regime is close to 1, that is, in heavy-traffic both settings have similar performances. Formally,

Theorem 4.4.2. *For a fixed $K < \infty$,*

$$\lim_{\bar{\lambda}\to\overline{R}} \frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})}{D_1(\bar{\lambda}, \mathbf{r})} = 1.$$

Proof. See Appendix 4.A.3. □

It is important that the number of dispatchers be finite for the above result to hold. As we show in Section 4.6, if the number of dispatchers is infinite, as in the case of non-atomic games, the above limit may be a value larger than 1.

This result is important because it is widely believed that the maximum inefficiency of the decentralized routing scheme is obtained in the heavy-traffic regime. Theorem 4.4.2 shows that this belief is false. As can be observed in Figure 4.3, the worst case traffic conditions can occur at low or moderate utilization rates (in fact, the worst total traffic intensity can be arbitrary close to 0 if the server capacities are sufficiently close to each other). In heavy-traffic, even though the cost in both the settings will grow, the rate of growth is the same which results in a ratio close to 1.

The characterization of the exact traffic intensity which results in $I_K^S(\mathbf{r})$ proves to be a difficult task for arbitrary values of the capacities. In the following section we restrict ourselves to two server classes.

4.5 Inefficiency for Two Classes of Servers

Before considering in detail the two-classes case, let us first observe how the inefficiency depends on the configuration of servers. Assume that there are 5 dispatchers and 13 servers. Figure 4.4 presents the evolution of the ratio $\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})}$ according to the total

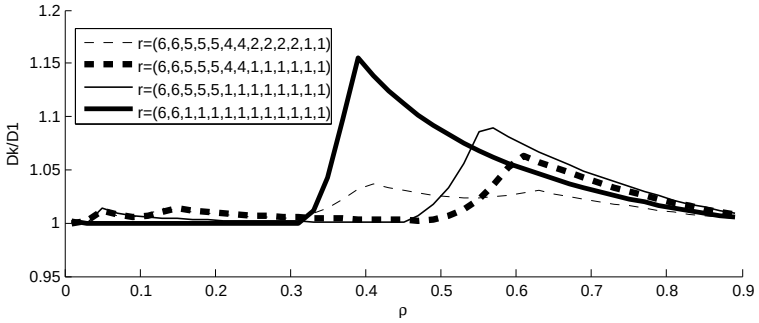

Figure 4.4: Evolution of the ratio $D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})/D_1(\bar{\lambda}, \mathbf{r})$ when $K = 5$ and $S = 13$ as the load in the system ranges from 0% to 90% for a server farm with different values of the capacities.

load of the system for several vectors $\mathbf{r}$ of server capacities. We observe that the highest inefficiency is obtained in the case of two classes of servers with extreme capacity values. From extensive numerical experimentations, we conjecture that, given the number of servers, for each problem instance with arbitrary server capacities we can construct a scenario with the same number of servers and two server speeds whose inefficiency is worse. This is formally stated in Conjecture 4.5.1.

Conjecture 4.5.1. *For a data-center of S servers with $r_1 > r_S$*

$$I_K^S(\mathbf{r}) \geq I_K^S(\mathbf{r}^*),$$

where $\mathbf{r} = (\overbrace{r_1, \ldots, r_1}^{m}, \overbrace{r_S, \ldots, r_S}^{S-m})$ *and* $\mathbf{r}^* = (\overbrace{r_1, \ldots, r_1}^{m}, \overbrace{r_{m+1}, \ldots, r_{S-1}, r_S}^{S-m})$, *with* $r_1 > r_{m+1} \geq \cdots \geq r_{S-1} \geq r_S$ *and* $m \geq 1$.

A direct consequence of the above conjecture is that if for two classes of servers the inefficiency is very close to one except in some pathological cases, this should also be true for more than two classes.

In the following, we thus consider a server farm with two classes of servers. Let S_1 be the number of "fast" servers of capacity r_1, and $S_2 = S - S_1$ be the number of "slow" servers, each one of capacity r_2, eith $r_1 > r_2$ [2]. The behaviour of the ratio $D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})/D_1(\bar{\lambda}, \mathbf{r})$ is illustrated in Figure 4.5 in the case of a server farm with $S_1 = 100$ fast servers of capacity $r_1 = 100$, and $S_2 = 300$ slow servers of capacity $r_2 = 10$. We plot the evolution of the ratio $D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})/D_1(\bar{\lambda}, \mathbf{r})$ as the load on the system ranges from 0% to 1 for $K = 2$, $K = 5$.

It was observed that for low loads both the settings used the fast servers. The ratio in this regime was 1. After a certain point, the centralized setting started to use the slow servers as well, and the ratio increased with the load until the point when the

[2] In the case $r_2 = r_1$, it is easy to see that the NEP is always an optimal routing solution, whatever the total traffic intensity.

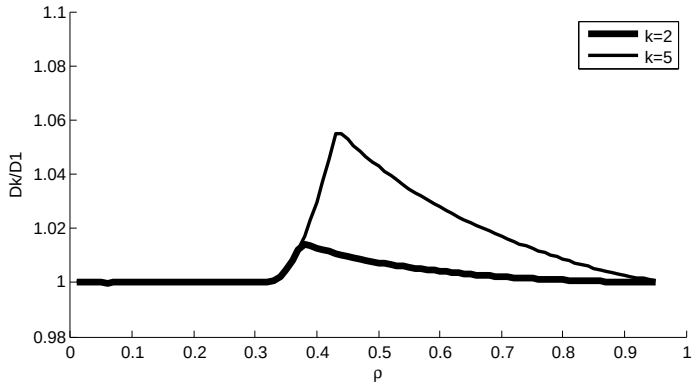

Figure 4.5: Evolution of the ratio $D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})/D_1(\bar{\lambda}, \mathbf{r})$ for $K = 2$ and $K = 5$ as the load in the system ranges from 0% to 100%.

decentralized setting also started to use the slow servers. From this point on, the ratio decreased with increase in the load.

We shall now characterize the point where the ratio starts to increase and where the peak occurs. Define

$$\bar{\lambda}^{OPT} = S_1\sqrt{r_1}(\sqrt{r_1} - \sqrt{r_2}),$$

and

$$\bar{\lambda}^{NE} = S_1 r_1 \left(1 - \frac{2}{\sqrt{(K-1)^2 + 4K\frac{r_1}{r_2}} - (K-1)}\right).$$

The following lemma gives the conditions on $\bar{\lambda}$ under which the centralized setting and the decentralized one use only the fast class of servers, or both classes.

Lemma 4.5.1. *For $K \geq 2$, $\bar{\lambda}^{OPT} < \bar{\lambda}^{NE}$, and*

1. *if $\bar{\lambda} \leq \bar{\lambda}^{OPT}$, both settings use only the "fast" servers,*
2. *if $\bar{\lambda}^{OPT} \leq \bar{\lambda} \leq \bar{\lambda}^{NE}$, the decentralized setting uses only the "fast" servers, while the centralized one uses all servers,*
3. *if $\bar{\lambda} > \bar{\lambda}^{NE}$, both settings use all servers.*

Proof. See Appendix 4.A.4. □

Since $\bar{\lambda}^{OPT} < \bar{\lambda}^{NE}$, a consequence of Lemma 4.5.1 is that the decentralized setting always uses a subset of the servers used by the centralized one. We immediately obtain expressions of the performance in the centralized and decentralized settings, as given in Corollary 4.5.1.

Corollary 4.5.1. *For the centralized setting, if* $\bar{\lambda} < \bar{\lambda}^{OPT}$*, then*

$$D_1(\bar{\lambda}, \mathbf{r}) = \bar{\lambda}/(r_1 - \frac{\bar{\lambda}}{S_1}),$$

otherwise

$$D_1(\bar{\lambda}, \mathbf{r}) = \left[\bar{\lambda}\sqrt{\frac{r_1}{r_2}} + S_1 y_1 \left(1 - \sqrt{\frac{r_1}{r_2}}\right)\right] \frac{1}{r_1 - y_1},$$

where $y_1 = \sqrt{r_1}\ \frac{\bar{\lambda} - S_2\sqrt{r_2}(\sqrt{r_2} - \sqrt{r_1})}{S_1\sqrt{r_1} + S_2\sqrt{r_2}}$*, and* $y_2 = (\bar{\lambda} - S_1 y_1)/S_2$ *are the loads on each fast server and on each slow server in the case* $\bar{\lambda} > \bar{\lambda}^{OPT}$*, respectively. Similarly, if* $\bar{\lambda} < \bar{\lambda}^{NE}$*, then*

$$D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r}) = \bar{\lambda}/(r_1 - \frac{\bar{\lambda}}{S_1}),$$

and

$$D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r}) = \frac{1}{2}\sum_{j=1}^{2} S_j \left[\sqrt{(K-1)^2 + 4Kr_j\gamma(K)} - (K+1)\right]$$

otherwise.

Proof. We first prove the results for the centralized setting. For $\bar{\lambda} < \bar{\lambda}^{OPT}$, we know from Lemma 4.5.1 that the centralized setting uses only the first group of servers, and we thus have

$$D_1(\bar{\lambda}, \mathbf{r}) = \sum_{j=1}^{S_1} \frac{\frac{\bar{\lambda}}{S_1}}{r_1 - \frac{\bar{\lambda}}{S_1}} = \frac{\bar{\lambda}}{r_1 - \frac{\bar{\lambda}}{S_1}}.$$

For $\bar{\lambda} \geq \bar{\lambda}^{OPT}$, we know from Lemma 4.5.1 that the centralized setting uses all the servers. According to the KKT conditions,

$$\frac{r_1}{(r_1 - y_1)^2} = \frac{r_2}{(r_2 - y_2)^2} \tag{4.5}$$

where $\bar{\lambda} = S_1 y_1 + S_2 y_2$. After some algebra, it yields

$$y_1 = \sqrt{r_1}\frac{\bar{\lambda} - S_2\sqrt{r_2}(\sqrt{r_2} - \sqrt{r_1})}{S_1\sqrt{r_1} + S_2\sqrt{r_2}}. \tag{4.6}$$

With the constraint of $S_1y_1 + S_2y_2 = \bar{\lambda}$ and (4.5), we obtain

$$\begin{aligned} D_1(\bar{\lambda}, \mathbf{r}) &= S_1\frac{y_1}{r_1 - y_1} + S_2\frac{y_2}{r_2 - y_2} \\ &= S_1\frac{y_1}{r_1 - y_1} + (\bar{\lambda} - S_1y_1)\frac{1}{r_2 - y_2} \\ &= S_1\frac{y_1}{r_1 - y_1} + (\bar{\lambda} - S_1y_1)\sqrt{\frac{r_1}{r_2}}\frac{1}{r_1 - y_1} \\ &= \left[\bar{\lambda}\sqrt{\frac{r_1}{r_2}} + S_1y_1\left(1 - \sqrt{\frac{r_1}{r_2}}\right)\right]\frac{1}{r_1 - y_1}, \end{aligned}$$

where y_1 is given in (4.6).

Let us now consider the decentralized setting. For $\bar{\lambda} < \bar{\lambda}^{NE}$, we know from Lemma 4.5.1 that the decentralized setting uses only the servers of the first class, and we thus have $D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r}) = \frac{\bar{\lambda}}{r_1 - \frac{\bar{\lambda}}{S_1}}$. For $\bar{\lambda} \geq \bar{\lambda}^{NE}$, all the servers are used and using (4.12) we obtain

$$\begin{aligned} D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r}) &= \sum_{j=1}^{S_1+S_2} \frac{y_j(K)}{r_j}\left(1 - \frac{y_j(K)}{r_j}\right)^{-1} \\ &= \frac{1}{2}\sum_{j=1}^{S_1+S_2}\left[\sqrt{(K-1)^2 + 4K\gamma(K)r_j} - (K+1)\right], \end{aligned}$$

and thus

$$D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r}) = \frac{1}{2}\sum_{j=1}^{2} S_j\left[\sqrt{(K-1)^2 + 4K\gamma(K)r_j} - (K+1)\right].$$

□

In Lemma 4.5.1, we identified three intervals, namely, $[0, \bar{\lambda}^{OPT})$, $[\bar{\lambda}^{OPT}, \bar{\lambda}^{NE})$, $[\bar{\lambda}^{NE}, \overline{R})$, each one corresponding to a different set of servers used by the two settings. In Proposition 4.5.1, we describe how the ratio $D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})/D_1(\bar{\lambda}, \mathbf{r})$ evolves in each of these three intervals.

Proposition 4.5.1. *The ratio* $D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})/D_1(\bar{\lambda}, \mathbf{r})$ *is*

(a) equal to 1 for $0 \leq \bar{\lambda} \leq \bar{\lambda}^{OPT}$,

(b) strictly increasing over the interval $\left(\bar{\lambda}^{OPT}, \bar{\lambda}^{NE}\right)$,

(c) and strictly decreasing over the interval $\left(\bar{\lambda}^{NE}, \overline{R}\right)$.

Proof. See Appendix 4.A.5. □

Moreover, the ratio $D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})/D_1(\bar{\lambda}, \mathbf{r})$ has the following property.

Lemma 4.5.2. *The ratio $D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})/D_1(\bar{\lambda},\mathbf{r})$ is a continuous function of $\bar{\lambda}$ over the interval $[0,\overline{R})$.*

Proof. First, we state that this function is continuous in $[0,\bar{\lambda}^{OPT})$, $(\bar{\lambda}^{OPT},\bar{\lambda}^{NE})$ and $(\bar{\lambda}^{NE},\overline{R})$ which follows from the definitions of $D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})$ and $D_1(\bar{\lambda},\mathbf{r})$ in Corollary 4.5.1. Now, we show that $\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})}$ is continuous in $\bar{\lambda}^{OPT}$ as follows:

$$\lim_{\bar{\lambda}\to\bar{\lambda}^{OPT-}}\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})}=\lim_{\bar{\lambda}\to\bar{\lambda}^{OPT+}}\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})}=\frac{D_K(\frac{\bar{\lambda}^{OPT}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda}^{OPT},\mathbf{r})}=1.$$

On the other hand, we also prove that it is continuous in $\bar{\lambda}^{NE}$:

$$\begin{aligned}\lim_{\bar{\lambda}\to\bar{\lambda}^{NE-}}\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})}=\lim_{\bar{\lambda}\to\bar{\lambda}^{NE+}}\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})}&=\frac{D_K(\frac{\bar{\lambda}^{NE}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda}^{NE},\mathbf{r})}\\&=\frac{\frac{1}{2}S_1[\sqrt{(K-1)^2+4Kr_1/r_2}-(K+1)]}{\frac{(S_1\sqrt{r_1}+S_2\sqrt{r_2})^2}{\frac{2S_1r_1}{\sqrt{(K-1)^2+4Kr_1/r_2}-(K-1)}+S_2r_2}-(S_1+S_2)}.\end{aligned}$$

□

We can now state the main result of this section.

Theorem 4.5.1. *The inefficiency is worst when the total arriving traffic intensity equals $\bar{\lambda}^{NE}$, namely,*

$$I_K^S(\mathbf{r})=\frac{D_K(\frac{\bar{\lambda}^{NE}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda}^{NE},\mathbf{r})}.$$

Proof. It was shown in Lemma 4.5.2 that $D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})/D_1(\bar{\lambda},\mathbf{r})$ is a continuous function of $\bar{\lambda}$ over the interval $[0,\overline{R})$. Proposition 4.5.1.(a) states that the ratio is minimum for $0\le\bar{\lambda}\le\bar{\lambda}^{OPT}$. For $\bar{\lambda}$ in $\left(\bar{\lambda}^{OPT},\bar{\lambda}^{NE}\right)$, we know from Proposition 4.5.1.(b) that this ratio is strictly increasing, which implies that

$$I_K^S(\mathbf{r})\ge D_K(\frac{\bar{\lambda}^{NE}}{K}\mathbf{e},\mathbf{r})/D_1(\bar{\lambda}^{NE},\mathbf{r}),$$

by continuity. Since, according to Proposition 4.5.1.(c), the ratio is decreasing over the interval $(\bar{\lambda}^{NE},\overline{R})$, we can conclude that its maximum value is obtained for $\bar{\lambda}=\bar{\lambda}^{NE}$. □

Theorem 4.5.1 fully characterizes the worst case traffic conditions for a server farm with two classes of servers. It states that the worst inefficiency of the decentralized setting is achieved when (a) each dispatcher controls the same amount of traffic and (b) the total traffic intensity is such that the decentralized setting only starts using the

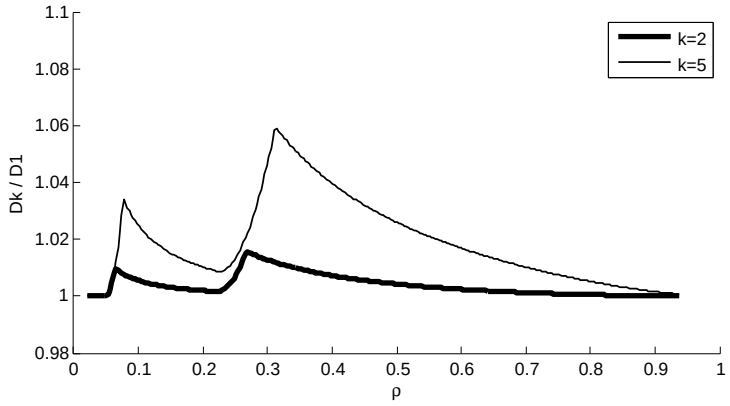

Figure 4.6: The evolution of the ratio $D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})/D_1(\bar{\lambda},\mathbf{r})$ for $K=2$ and $K=5$ with respect to ρ in a server farm with 3 server classes.

slow servers. The behaviour described by Proposition 4.5.1 can easily be observed in Figure 4.5.

For more than two classes of servers, we were unfortunately not able to prove the above results concerning the worst traffic conditions. Nevertheless, we conjecture that a similar behaviour happens also in this case. As another illustration of this behaviour, in Figure 4.6 we plot the ratio $D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})/D_1(\bar{\lambda},\mathbf{r})$ as a function of the load on the system, for a server farm with 3 server classes (and for $K=2$, $K=5$) with $S_1=100$ fast servers of capacity $r_1=30$, $S_2=200$ intermediate servers of capacity $r_2=20$ and $S_3=100$ slow servers of capacity $r_3=10$.

4.5.1 *Inefficiency* Analysis

We now give the expression for the *inefficiency* of selfish routing for data-centers with two classes of servers. Using Theorem 4.5.1 we assume the worst traffic conditions for the inefficiency of selfish routing, i.e., the symmetric game obtained for $\bar{\lambda}=\bar{\lambda}^{NE}$.

Proposition 4.5.2. *Let* $\beta=\frac{r_1}{r_2}>1$ *and* $\alpha=\frac{S_1}{S_2}>0$, *then*

$$I_K^S(\mathbf{r})=\frac{1}{2}\frac{\sqrt{(K-1)^2+4K\beta}-(K+1)}{\frac{(\frac{1}{\alpha}+\sqrt{\beta})^2}{\frac{1}{\alpha}+\frac{2\beta}{\sqrt{(K-1)^2+4K\beta}-(K-1)}}-(\frac{1}{\alpha}+1)}. \tag{4.7}$$

Proof. According to Theorem 4.5.1, we have $I_K^S(\mathbf{r})=D_K(\frac{\bar{\lambda}^{NE}}{K}\mathbf{e},\mathbf{r})/D_1(\bar{\lambda}^{NE},\mathbf{r})$. The proof is then obtained after some algebra by using the expressions for $D_K(\frac{\bar{\lambda}^{NE}}{K}\mathbf{e},\mathbf{r})$ and $D_1(\bar{\lambda}^{NE},\mathbf{r})$ given in Corollary 4.5.1, and the expression for $\bar{\lambda}^{NE}$ given in Lemma 4.5.1. □

The *inefficiency* $I_K^S(\mathbf{r})$ does not depend on the total number of servers S, but only on the ratio of server capacities and on the ratio of the numbers of servers of each type. In Figure 4.7 and 4.8, we plot the *inefficiency* $I_K(\mathbf{r})$ of the non-cooperative routing scheme with $K = 2$ and $K = 5$ dispatchers and $S = 1000$ servers as the parameters α and β change from $\frac{1}{S-1}$ to 2 and from 1 to 1000, respectively. It can be observed that even for unbalanced scenarios (α small and β large), the *inefficiency* is always fairly close to 1, indicating that, even in the worst case traffic conditions, the gap between the NEP and the optimal routing solution is not significant. With slight abuse of notation, let us denote the RHS of (4.7) by $I_K(\alpha, \beta)$.

Lemma 4.5.3. *The function $I_K(\alpha, \beta)$ is decreasing with α.*

Proof. First, we modify $I_K(\alpha, \beta)$ and we obtain

$$I_K(\alpha, \beta) = \frac{1}{2} \frac{(x-2)(\frac{1}{\alpha} + \frac{2\beta}{x})}{\frac{1}{\alpha}(2\sqrt{\beta} - 1 - \frac{2\beta}{x}) + \beta(1 - \frac{2}{x})},$$

where $x = \sqrt{(K-1)^2 + 4K\beta} - (K-1)$.

We now show that the derivative of $I_K(\alpha, \beta)$ with respect to α is negative using that the derivative of $\frac{1}{\alpha}$ with respect to α is negative:

$$\begin{aligned}
\frac{\partial I_K(\alpha, \beta)}{\partial \alpha} &= \frac{\frac{1}{2}\left(\frac{1}{\alpha}\right)' \left[\beta(1 - \frac{2}{x}) - \frac{2\beta}{x}(2\sqrt{\beta} - 1 - \frac{2\beta}{x})\right]}{\left[\frac{1}{\alpha}(2\sqrt{\beta} - 1 - \frac{2\beta}{x}) + \beta(1 - \frac{2}{x})\right]^2} \\
&= \frac{\frac{1}{2}\left(\frac{1}{\alpha}\right)' \left(1 - \frac{2\beta}{x}\right)^2}{\left[\frac{1}{\alpha}(2\sqrt{\beta} - 1 - \frac{2\beta}{x}) + \beta(1 - \frac{2}{x})\right]^2} < 0.
\end{aligned}$$

□

A consequence of the above result is that given the ratio of server speeds in a data-center, the *inefficiency* is largest when there is one fast server and all the other servers are slow. Selfish routing has the tendency to use the fast servers more than the slow ones. When there is just one fast server, its performance tends to be the worst as compared to that of the centralized routing which reduces its cost by sending traffic to the slower ones as well. Thus, in decentralized routing architectures, it is best to avoid server configurations with this particular kind of asymmetry.

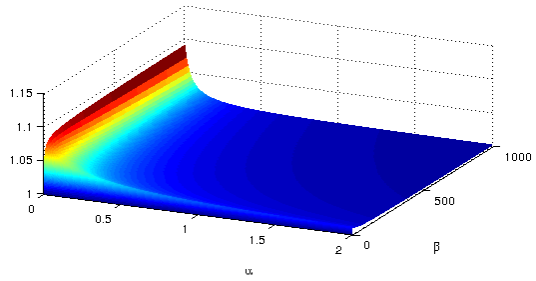

Figure 4.7: Evolution of the *inefficiency* as a function of α and β for $K = 2$ dispatchers and $S = 1000$ servers.

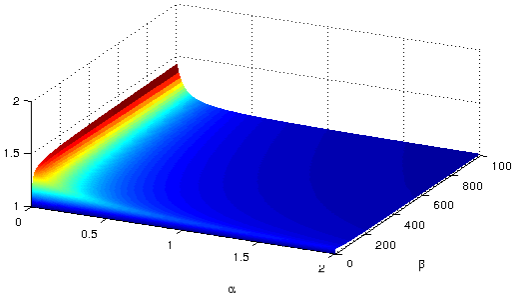

Figure 4.8: Evolution of the *inefficiency* as a function of α and β for $K = 5$ dispatchers and $S = 1000$ servers.

4.5.2 Price of Anarchy

The PoA is defined as the worst possible *inefficiency* when the server capacities are varied. Then, from (4.3), (4.4) and Proposition 4.5.2, it follows that

$$PoA(K, S) = \sup_{\alpha,\beta} I_K(\alpha, \beta).$$

From Lemma 4.5.3 and the fact that, for a fixed S, α can take values in $\{\frac{1}{S-1}, \frac{2}{S-2}, \ldots, S-1\}$, we can deduce that

$$PoA(K, S) = \sup_{\beta} I_K\left(\frac{1}{S-1}, \beta\right). \tag{4.8}$$

We are able to prove that $I_K\left(\frac{1}{S-1}, \beta\right)$ is increasing with β. This means that the PoA of a server farm with S servers and K dispatchers is achieved when $\alpha = \frac{1}{S-1}$ and β infinity, i.e., when there is only one fast server and it is infinitely faster than the slower ones. While there is no simple expression for the PoA in terms of K and S, we can nonetheless derive a certain number of properties from the preceding set of results.

Proposition 4.5.3. *The Price of Anarchy has the following properties.*

1. *For fixed K, $PoA(K, S)$ is increasing in S; and*
2. *for a fixed S, $PoA(K, S)$ is increasing in K.*

Proof. For fixed K and for every β, from Lemma 4.5.3 and (4.8),

$$I_K\left(\frac{1}{S-1}, \beta\right) \leq I_K\left(\frac{1}{S}, \beta\right) \leq \sup_{\beta} I_K\left(\frac{1}{S}, \beta\right) = PoA(K, S+1),$$

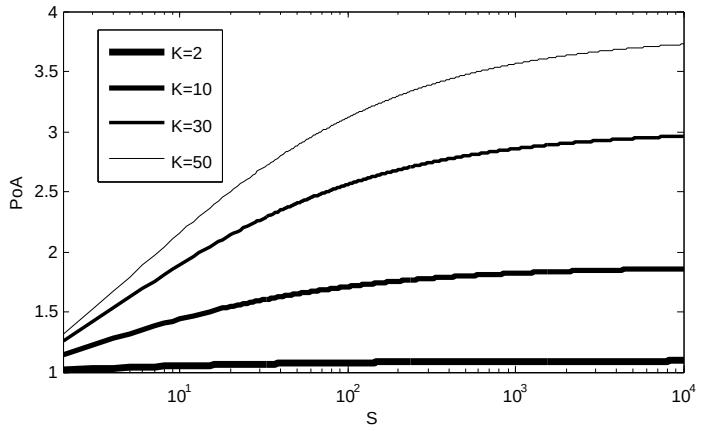

Figure 4.9: The Price of Anarchy as a function of the number of servers for different values of the number of dispatcher.

where the last equality follows from (4.8). Taking the supremum over β in the above inequality, we obtain, for a fixed K,

$$PoA(K,S) \leq PoA(K,S+1),$$

which proves the first property.

For a fixed S and β, from Lemma 4.4.1,

$$I_K\left(\frac{1}{S-1},\beta\right) \leq I_{K+1}\left(\frac{1}{S-1},\beta\right) \leq \sup_{\beta} I_{K+1}\left(\frac{1}{S-1},\beta\right) = PoA(K+1,S),$$

Again, taking the supremum over β in the above inequality, we obtain, for a fixed S,

$$PoA(K,S) \leq PoA(K+1,S),$$

which proves the second property. □

In Figure 4.9, the PoA is plotted as a function of S for different values of K. It is observed that this value remains modest even when the number of servers is $10,000$.

We now give an upper bound the PoA. For this, we first need the following result.

Lemma 4.5.4. *For a server farm with two server classes and K dispatchers,*

$$\lim_{S\to\infty} PoA(K,S) = \frac{K}{2\sqrt{K}-1}.$$

Proof. See Appendix 4.A.6. □

Proposition 4.5.4. *For a server farm with two server classes and K dispatchers, and for all K and S,*

$$PoA(K,S) \leq \min\left(\frac{K}{2\sqrt{K}-1}, S\right).$$

Proof. From Proposition 4.5.3, $PoA(K,S)$ is increasing with S. Combining this fact with Lemma 4.5.4, we can conclude that

$$PoA(K,S) \leq \frac{K}{2\sqrt{K}-1}.$$

Moreover, it was shown in [52] that, for the Wardrop case which is the limit of $K \to \infty$, $PoA(\infty,S) \leq S$. Thus,

$$PoA(K,S) \leq S.$$

We can deduce the desired result from the above two inequalities. □

In server farms with large number of servers, it follows from Lemma 4.5.4 that the PoA will be $\frac{K}{2\sqrt{K}-1}$. In [14], it was shown that this value was a lower bound on the PoA. The model in that paper had server dependent holding cost per unit time. The lower bound was obtained in an extreme case with negligible (tending to 0) holding cost on the fast servers and the decentralized setting marginally using the slow servers. Our present results show that the lower bound is indeed tight. Moreover, even in a less asymmetrical setting of equal holding costs per unit time, one can construct examples in which the PoA is attained.

The PoA obtained in the non-atomic case in [52] comes into play when there are few servers and a relatively large number of dispatcher. However, for data-centers the configuration is reversed: there are a few dispatchers and a large number of servers. In this case it is more appropriate to use the upper bound given in Lemma 4.5.4.

4.6 Inefficiency with an Infinite Number of Dispatchers

We know from Lemma 4.4.1 that the *inefficiency* increases with the number of dispatchers K. This motivates the analysis of the *inefficiency* when the number of dispatchers K grows to infinity. In this section, we show that in the heavy-traffic regime the *inefficiency* is not one, in contrast to the case of finite K. For the case of two classes of servers we give the expression of the *inefficiency*, we characterize the situation under which the PoA is achieved and show that it is equal to the number of servers.

We define *inefficiency* of a server farm with S servers and an infinite number of dispatchers as

$$I_{\infty}^{S}(\mathbf{r}) = \lim_{K\to\infty} I_{K}^{S}(\mathbf{r}). \tag{4.9}$$

4.6.1 Heavy-traffic Analysis

We study the performance of a data center with an arbitrary number of servers S when the system is in the heavy-traffic regime. We give the value of $\lim_{K\to\infty} \frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})}$ in the following proposition:

Proposition 4.6.1. *For a data-center with S servers, we have*

$$\lim_{\bar{\lambda}\to\overline{R}} \lim_{K\to\infty} \frac{D_K(\frac{\bar{\lambda}}{K}\,\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})} = \frac{S\ \overline{R}}{\left(\sum_{i=1}^{S}\sqrt{r_i}\right)^2}.$$

Proof. See Appendix 4.A.7. □

We observe that this result does not coincide with the one obtained for the Nash equilibrium for K large, as given in Theorem 4.4.2. This implies that the limits of the number of dispatchers and heavy-traffic do not interchange, i.e.,

$$\lim_{K\to\infty} \lim_{\bar{\lambda}\to\overline{R}} \frac{D_K(\frac{\bar{\lambda}}{K}\,\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})} \neq \lim_{\bar{\lambda}\to\overline{R}} \lim_{K\to\infty} \frac{D_K(\frac{\bar{\lambda}}{K}\,\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})}.$$

4.6.2 The case of Two Classes of Servers

We focus on the case of a data center with servers of two different speeds, r_1 and r_2 respectively, where $r_1 > r_2$. For the case of $K = \infty$, we conjecture that the worst inefficiency is obtained for two server-classes, i.e., the Conjecture 4.5.1 holds when the number of dispachers grows to infinity.

Let S_1 be the number of "fast" servers and $S_2 = S - S_1$ be the number of "slow" servers. We give the expression of the *inefficiency* for a data center with two classes of servers in terms only on the parameters $\alpha = \frac{S_1}{S_2}$ and $\beta = \frac{r_1}{r_2} > 1$.

Corollary 4.6.1.

$$I_{\infty}(\alpha,\beta) = \frac{(\beta-1)(1+\frac{1}{\alpha})}{(\sqrt{\beta}+\frac{1}{\alpha})^2 - (\frac{1}{\alpha}+1)^2}. \tag{4.10}$$

Proof. The result follows from Theorem 4.5.1 and (4.9), taking into account that

$$\sqrt{(K-1)^2+4Kx} - (K-1) = 2x,$$

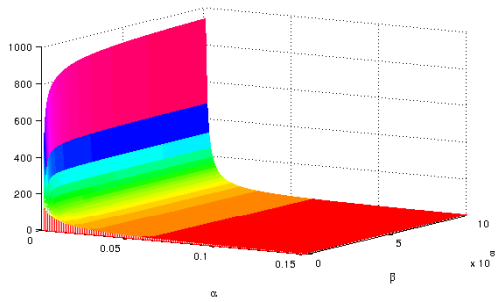

Figure 4.10: Evolution of the *inefficiency* as a function of α and β for $K = 10^6$ and $S = 1000$.

when $K \to \infty$, $\forall x \geq 1$. □

Using this expression, we can use the parameters $\alpha \in \{\frac{1}{S-1}, \frac{2}{S-2}, \ldots, \frac{S-2}{2}, S-1\}$ and β to characterize the inefficiency for a data center with two-server classes and infinite number of servers. Lemma 4.6.1 states the main properties of the inefficiency for an infinite number of dispatchers.

Lemma 4.6.1. *We have $I_\infty(\alpha, \beta)$ is decreasing with α, for all β, and $I_\infty(\alpha, \beta)$ is increasing with β, for all α.*

Proof. The result follows from Corollary 4.6.1. □

A direct consequence of Corollary 4.6.1 and Lemma 4.6.1 is that the Price of Anarchy of a data center with two classes of servers and infinite number of dispatchers is equal to the number of servers S.

Proposition 4.6.2. $PoA(\infty, S) = S$.

Proof. According to Lemma 4.6.1, the worst inefficiency is obtained when $\alpha = \frac{1}{S-1}$ and $\beta \to \infty$. Using the formula of Corollary 4.6.1 for this values of α and β, we get the desired result. □

We observe that this result coincides with the result given by [52].

We illustrate in Figure 4.10 the evolution of the inefficiency of a server farm of $S = 1000$ servers and $K = 10^6$ dispatchers when α changes from $\frac{1}{S-1}$ to 0.15 and β from 1 to 10^8. We observe that the inefficiency equals the PoA when $\alpha = \frac{1}{S-1}$ and $\beta = 10^8$, i.e., when there is only one fast server and it is 10^8 times faster than the slower ones. We also see in Figure 4.10 that the inefficiency stays very close to one in most cases, even if the worst inefficiency is 1000. Thus, we can conclude that although the inefficiency can be as bad as the number of servers when the number of dispatchers is infinity, the decentralized setting is almost always as efficient as the centralized setting.

4.6.3 Price of Anarchy

We have seen that the PoA equals the number of servers in case of an infinity number of dispatchers, while for K finite the upper-bound is given by the minimum of $\frac{K}{2\sqrt{K}-1}$ and S. We observe that, given the number of servers S, there exist a K^* such that

- if $K \leq K^*$, then $PoA \leq \frac{K}{2\sqrt{K}-1}$,
- if $K \geq K^*$, then $PoA \leq S$.

For a sufficiently large S, we can say that $S = \frac{K^*}{2\sqrt{K^*}-1} \approx 0.5\sqrt{K^*}$. Thus, we claim that for a sufficiently large S, then $K^* \approx 4S^2$ and this means that if the number of dispatchers is smaller than $4S^2$ the upper-bound on the PoA is given by $\frac{K}{2\sqrt{K}-1}$, and by the number of servers S otherwise.

4.7 Conclusions

Price of Anarchy is an oft-used worst-case measure of the inefficiency of non-cooperative decentralized architectures. In spite of its popularity, we have observed that the Price of Anarchy is an overly pessimistic measure that does not reflect the performance obtained in most instances of the load balancing game. For an arbitrary architecture in the system, we have seen that, contrary to a common belief, the inefficiency is in general not achieved in the heavy-traffic regime. Surprisingly, we have shown that inefficiency might be achieved at arbitrarily low load. For the case of two classes of servers we give an explicit expression of the inefficiency and we have shown that non-cooperative load balancing has close-to-optimal performances in most cases. We also show that the worst-case performances given by the Price of Anarchy occur only in a very specific setting, namely, when there is only one fast server and it is infinitely faster than the slower ones.

We believe that our study opens up a new complementary point of view on the PoA and we hope that in the future researchers will not only investigate the PoA, but also the inefficiency. As our work suggests, even if the PoA is very bad, the inefficiency might be low in most instances of the problem. We believe that this issue should be investigated for other models. As a future work, we aim to prove that that our conclusions are also true for more than two classes of servers.

4.A Appendix of Chapter 4

4.A.1 Some Known Results

The results in this section are taken from [14]. Since they are cited several times in the present work, we choose to present them here for its easy perusal. Let $W(K, z) =$

$\sum_{j\in\mathcal{S}} W_j(K,z)$, we define the function

$$W_j(K,z) = \mathbb{1}_{\{z\in[\frac{1}{r_j},\frac{1}{r_{j+1}})\}} \cdot \left(\sum_{s=1}^{j} \frac{2r_s}{\sqrt{(K-1)^2+4Kr_sz}-(K-1)} - \sum_{s=1}^{j} r_s + \bar{\lambda} \right). \quad (4.11)$$

The following proposition gives the solution of the symmetric game.

Proposition 4.A.1. *The subset of servers that are used at the NEP is* $\mathcal{S}^*(K) = \{1,2,\ldots,j^*(K)\}$, *where* $j^*(K)$ *is the greatest value of* j *such that* $W(K,1/r_{j+1}) \leq 0 < W(K,1/r_j)$. *The equilibrium flows are* $x_{i,j}(K) = y_j(K)/K$, $i \in \mathcal{C}, j \in \mathcal{S}^*(K)$, *where the offered traffic of server* j *is given by*

$$y_j(K) = r_j \frac{\sqrt{(K-1)^2+4K\gamma(K)r_j}-(K+1)}{\sqrt{(K-1)^2+4K\gamma(K)r_j}-(K-1)}, \quad (4.12)$$

with $\gamma(K)$ *the unique root of* $W(K,z)=0$ *in* $[\frac{1}{r_1},\infty)$.

4.A.2 Proof of Proposition 4.4.1

Before proving Proposition 4.4.1, we establish closed-form expressions for the value of the performance of the centralized and decentralized settings. Recall that we assume a server farm with S servers with decreasing values of the capacities, i.e, $r_i \leq r_j$, if $i > j$. The result is stated in the following lemma.

Lemma 4.A.1. *Let* n *be the number of servers that the centralized setting uses, for* $n = 1\ldots,S$, *then*

$$D_1(\bar{\lambda},\mathbf{r}) = \frac{\left(\sum_{j=1}^{n}\sqrt{r_j}\right)^2}{\sum_{j=1}^{n} r_j - \bar{\lambda}} - n.$$

Similarly, if the decentralized setting uses n *servers, we have*

$$D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r}) = \frac{1}{2}\sum_{j=1}^{n}\left[\sqrt{(K-1)^2+4Kr_j\gamma(K)}-(K+1)\right].$$

Proof. We first prove the results for the centralized setting. When the centralized setting uses n servers Proposition 4.A.1 states that in this case,

$$y_j(1) = r_j\left(1-\frac{1}{\sqrt{\gamma(1)}\sqrt{r_j}}\right), \quad j=1,\ldots,n,$$

that yields

$$\frac{y_j(1)}{r_j-y_j(1)} = \sqrt{\gamma(1)}\sqrt{r_j}-1, \text{ for } j=1,\ldots,n.$$

We thus obtain that

$$D_1(\bar{\lambda}, \mathbf{r}) = \sum_{j=1}^{n} \frac{y_j(1)}{r_j - y_j(1)} = \sqrt{\gamma(1)} \sum_{j=1}^{n} \sqrt{r_j} - n.$$

Since $\gamma(1)$ is the unique root of $W(K, \gamma(1)) = 0$ as defined in Appendix 4.A.1 and, according to Proposition 4.A.1, then $\gamma(1)$ is the solution of

$$\frac{1}{\sqrt{\gamma(1)}} \sum_{j=1}^{n} \sqrt{r_j} = \sum_{j=1}^{n} r_j - \bar{\lambda}. \tag{4.13}$$

Thus, it follows that $\sqrt{\gamma(1)} = \sum_{j=1}^{n} \sqrt{r_j} / (\sum_{j=1}^{n} r_j - \bar{\lambda})$ and

$$D_1(\bar{\lambda}, \mathbf{r}) = \frac{(\sum_{j=1}^{n} \sqrt{r_j})^2}{\sum_{j=1}^{n} r_j - \bar{\lambda}} - n.$$

Let us now consider the decentralized setting. If the number of servers used by the decentralized setting is n, then (4.12) gives that for $j = 1, \ldots, n$

$$1 - \frac{y_j(K)}{r_j} = \frac{2}{\sqrt{(K-1)^2 + 4K\gamma(K)r_j} - (K-1)}. \tag{4.14}$$

From (4.12) and (4.14), it yields the desired result

$$D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r}) = \sum_{j=1}^{n} \frac{y_j(K)}{r_j} \left(1 - \frac{y_j(K)}{r_j}\right)^{-1} = \frac{1}{2} \sum_{j=1}^{n} \left[\sqrt{(K-1)^2 + 4K\gamma(K)r_j} - (K+1)\right].$$

□

We first show in the following lemma an important property to prove Proposition 4.4.1.

Lemma 4.A.2. *Let $a_k = \sqrt{(K-1)^2 + 4K\gamma(K)r_k} + (K-1)$, then for all $i > j$, $\frac{a_j}{a_i}$ is increasing with $\bar{\lambda}$.*

Proof. First, we define $b_j = \sqrt{(K-1)^2 + 4K\gamma(K)r_j}$ and we see that $\frac{b_j}{b_i}$ is increasing with $\bar{\lambda}$ if $\frac{(K-1)^2+4K\gamma(K)r_j}{(K-1)^2+4K\gamma(K)r_i}$ is increasing with $\bar{\lambda}$ because $\frac{b_j}{b_i}$ is positive and thus:

$$\frac{\partial}{\partial \bar{\lambda}} \left(\frac{(K-1)^2 + 4K\gamma(K)r_j}{(K-1)^2 + 4K\gamma(K)r_i}\right) \geq 0 \iff 4K \frac{\partial \gamma(K)}{\partial \bar{\lambda}} (K-1)^2 (r_j - r_i) \geq 0$$

that it is true due to $r_j \geq r_i$ if $i > j$. Hence, we have proved that $\frac{(K-1)^2+4K\gamma(K)r_j}{(K-1)^2+4K\gamma(K)r_i}$ is increasing with $\bar{\lambda}$ and this implies that $\frac{b_j}{b_i}$ is increasing with $\bar{\lambda}$. We also observe that

$\frac{\partial b_j}{\partial \bar{\lambda}} \geq \frac{\partial b_i}{\partial \bar{\lambda}}$, if $i > j$:

$$\begin{aligned} \frac{\partial b_j}{\partial \bar{\lambda}} \geq \frac{\partial b_i}{\partial \bar{\lambda}} \quad &\iff \quad \frac{2K\frac{\partial \gamma(K)}{\partial \bar{\lambda}} r_j}{b_j} \geq \frac{2K\frac{\partial \gamma(K)}{\partial \bar{\lambda}} r_i}{b_i} \\ &\iff \quad \frac{1}{\sqrt{\frac{(K-1)^2}{r_j^2} + \frac{4K\gamma(K)}{r_j}}} \geq \frac{1}{\sqrt{\frac{(K-1)^2}{r_i^2} + \frac{4K\gamma(K)}{r_i}}}, \end{aligned}$$

and this inequality holds since $r_j \geq r_i$ when $i > j$.

As $\frac{\partial b_j}{\partial \bar{\lambda}} = \frac{\partial a_j}{\partial \bar{\lambda}}$ and $a_j = b_j + (K-1)$, for $j = 1, \ldots, n$, we are able to state that if $\frac{b_j}{b_i}$ is increasing, then $\frac{a_j}{a_i}$ is increasing with $\bar{\lambda}$:

$$\frac{\partial}{\partial \bar{\lambda}}\left(\frac{a_j}{a_i}\right) > 0 \iff \frac{\frac{\partial b_j}{\partial \bar{\lambda}} a_i - \frac{\partial b_i}{\partial \bar{\lambda}} a_j}{a_i^2} > 0 \iff \frac{\partial b_j}{\partial \bar{\lambda}} b_i - \frac{\partial b_i}{\partial \bar{\lambda}} b_j + (K-1)\left(\frac{\partial b_j}{\partial \bar{\lambda}} - \frac{\partial b_i}{\partial \bar{\lambda}}\right) > 0,$$

and we know the inequality is satisfied because $\frac{\partial\left(\frac{b_j}{b_i}\right)}{\partial \bar{\lambda}} > 0$ and $\frac{\partial b_j}{\partial \bar{\lambda}} > \frac{\partial b_i}{\partial \bar{\lambda}}$. □

We are now in situation to prove the result of Proposition 4.4.1.

Proof. We show that when both settings use n servers ($n = 1, \ldots, S$), then the ratio $\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})}{D_1(\bar{\lambda}, \mathbf{r})}$ is decreasing with $\bar{\lambda}$. We use the expressions of $D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})$ and $D_1(\bar{\lambda}, \mathbf{r})$ given in Lemma 4.A.1 and we modify the ratio $\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})}{D_1(\bar{\lambda}, \mathbf{r})}$ as follows:

$$\begin{aligned} \frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})}{D_1(\bar{\lambda}, \mathbf{r})} &= \frac{\frac{1}{2}\sum_{j=1}^{n}\left[\sqrt{(K-1)^2 + 4Kr_j\gamma(K)} - (K+1)\right]}{-n + \sqrt{\gamma(1)}\sum_{j=1}^{n}\sqrt{r_j}} \\ &= \frac{-n + \frac{1}{2}\sum_{j=1}^{n}\left[\sqrt{(K-1)^2 + 4Kr_j\gamma(K)} - (K-1)\right]}{-n + \sqrt{\gamma(1)}\sum_{j=1}^{n}\sqrt{r_j}} = \frac{f_1 + f_2}{f_1 + g_2}, \end{aligned}$$

where we define $f_1 = \frac{-n}{\sqrt{\gamma(1)}}$, $g_2 = \sum_{j=1}^{n}\sqrt{r_j}$ and

$$f_2 = \frac{1}{2\sqrt{\gamma(1)}}\sum_{j=1}^{n}\left[\sqrt{(K-1)^2 + 4Kr_j\gamma(K)} - (K-1)\right].$$

We want to prove that the derivative of $\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})}{D_1(\bar{\lambda}, \mathbf{r})}$ with respect to $\bar{\lambda}$ is negative. We have that:

$$\frac{\partial}{\partial \bar{\lambda}}\left(\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})}{D_1(\bar{\lambda}, \mathbf{r})}\right) < 0 \iff \frac{\partial f_1}{\partial \bar{\lambda}}(g_2 - f_2) + \frac{\partial f_2}{\partial \bar{\lambda}}\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})}{\sqrt{\gamma(1)}} < 0.$$

We observe that f_1 is increasing with $\bar{\lambda}$, because $\gamma(1)$ increases with $\bar{\lambda}$, and $D_1(\bar{\lambda}, \mathbf{r}) \leq D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})$ implies that $g_2 \leq f_2$. Therefore, if we show that f_2 is decreasing with $\bar{\lambda}$ and we can conclude that $\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})}$ is decreasing with $\bar{\lambda}$. From (4.13) and (4.11), if both settings use n servers then

$$\frac{1}{\sqrt{\gamma(1)}} = \frac{\sum_{j=1}^n r_j - \bar{\lambda}}{\sum_{j=1}^n \sqrt{r_j}} = \frac{1}{\sum_{j=1}^n \sqrt{r_j}} \sum_{s=1}^n \frac{2r_s}{\overline{a_s}},$$

where $\overline{a_s} = \sqrt{(K-1)^2 + 4K\gamma(K)r_s} - (K-1)$. We rewrite f_2 as follows:

$$f_2 = \frac{1}{\sum_{j=1}^n \sqrt{r_j}} \sum_{j=1}^n \overline{a_j} \sum_{s=1}^n \frac{r_s}{\overline{a_s}} = \frac{1}{\sum_{j=1}^n \sqrt{r_j}} \left(\sum_{j=1}^n r_j + \sum_{j=1}^n \sum_{i>j} \left[r_j \frac{\overline{a_i}}{\overline{a_j}} + r_i \frac{\overline{a_j}}{\overline{a_i}} \right] \right).$$

We define $a_s = \sqrt{(K-1)^2 + 4K\gamma(K)r_s} + (K-1)$ and we notice that if we multiply and divide $\overline{a_s}$ by a_s it yields $\overline{a_s} = \frac{4K\gamma(K)r_s}{a_s}$. So f_2 gets modified as follows with this property:

$$f_2 = \frac{1}{\sum_{j=1}^n \sqrt{r_j}} \left(\sum_{j=1}^n r_j + \sum_{j=1}^n \sum_{i>j} \left[r_j \frac{a_i}{a_j} + r_i \frac{a_j}{a_i} \right] \right).$$

Now, we show that $r_j/a_j^2 > r_i/a_i^2$ for all $i > j$ since $\frac{r_k}{a_k^2}$ is decreasing with k because we can write it as $\frac{r_k}{a_k^2} = \left[\left(\sqrt{\frac{(K-1)^2}{r_k} + 4K\gamma(K)} + \frac{K-1}{\sqrt{r_k}} \right)^{-1} \right]^2$ and r_k decreases with k. Finally, we see that f_2 is decreasing with $\bar{\lambda}$:

$$\frac{\partial f_2}{\partial \bar{\lambda}} = \frac{1}{\sum_{j=1}^n \sqrt{r_j}} \sum_{j=1}^n \sum_{i>j} \left[\left(\frac{\partial a_j}{\partial \bar{\lambda}} a_i - \frac{\partial a_i}{\partial \bar{\lambda}} a_j \right) \left(\frac{r_i}{a_i^2} - \frac{r_j}{a_j^2} \right) \right] < 0,$$

and we conclude that this is true because from Lemma 4.A.2 $\frac{a_j}{a_i}$ is increasing with $\bar{\lambda}$ if $i > j$ (so that $\frac{\partial a_j}{\partial \bar{\lambda}} a_i - \frac{\partial a_i}{\partial \bar{\lambda}} a_j > 0$) and we have observed that $r_j/a_j^2 > r_i/a_i^2$ when $i > j$. □

4.A.3 Proof of Theorem 4.4.2

Proof. First, we know that in heavy-traffic all the servers are used, so we consider that S servers are used in both settings. We also observe that in heavy-traffic $\gamma(K)$, as defined in Proposition 4.A.1, tends to ∞, and the following approximation is satisfied:

$$\sqrt{(K-1)^2 + 4K\gamma(K)r_j} - (K-1) \approx 2\sqrt{K\gamma(K)r_j}. \tag{4.15}$$

From (4.15) and the definition of $\gamma(K)$, we obtain $\sqrt{K\gamma(K)} \approx \frac{\sum_{j=1}^S \sqrt{r_j}}{\sum_{j=1}^S r_j - \bar{\lambda}}$. Now,

using this expression, (4.15) and Lemma 4.A.1, we show that $D_K(\frac{\bar\lambda}{K}\mathbf{e},\mathbf{r}) \approx D_1(\bar\lambda,\mathbf{r})$ in heavy-traffic:

$$\begin{aligned} D_K(\frac{\bar\lambda}{K}\mathbf{e},\mathbf{r}) &= \frac{1}{2}\sum_{j=1}^{S}\left[\sqrt{(K-1)^2+4K\gamma(K)r_j}-(K+1)\right] \\ &= -S+\frac{1}{2}\sum_{j=1}^{S}\left[\sqrt{(K-1)^2+4K\gamma(K)r_j}-(K-1)\right] \\ &\approx -S+\sqrt{K\gamma(K)}\sum_{j=1}^{S}\sqrt{r_j} = -S+\frac{(\sum_{j=1}^{S}\sqrt{r_j})^2}{\sum_{j=1}^{S}r_j-\bar\lambda} = D_1(\bar\lambda,\mathbf{r}). \end{aligned}$$

□

4.A.4 Proof of Lemma 4.5.1

Proof. Let us first prove that $\bar\lambda^{OPT} < \bar\lambda^{NE}$:

$$\begin{aligned} \bar\lambda^{OPT} < \bar\lambda^{NE} &\iff \sqrt{r_1r_2} > \frac{2r_1}{\sqrt{(K-1)^2+4Kr_1/r_2}-(K-1)} \\ &\iff \sqrt{r_1r_2}\sqrt{(K-1)^2+4Kr_1/r_2} > \sqrt{r_1}\,[2\sqrt{r_1}+(K-1)\sqrt{r_2}] \\ &\iff 4Kr_1 > 4r_1+4(K-1)\sqrt{r_1r_2} \\ &\iff r_1 > r_2. \end{aligned}$$

We now turn to the second part of the proof. According to Proposition 4.A.1, the centralized setting uses only the fast servers (S_1 servers of capacity r_1) for all values of $\bar\lambda$ such that $W_2(1,\frac{1}{r_2}) \le 0$. It yields

$$\bar\lambda \le (S_1r_1+S_2r_2)-\sqrt{r_2}(S_1\sqrt{r_1}-S_2\sqrt{r_2})$$

which is equivalent to $\bar\lambda \le \bar\lambda^{OPT}$.

Similarly, we know from Proposition 4.A.1, that the decentralized setting starts using the second group of servers if and only if

$$\begin{aligned} \bar\lambda &\ge S_1r_1+S_2r_2-\sum_{s=1}^{S_1+S_2}\frac{2r_s}{\sqrt{(K-1)^2+4Kr_s/r_2}-(K-1)} \\ &= S_1r_1+S_2r_2-\sum_{s=1}^{2}S_s\frac{2r_s}{\sqrt{(K-1)^2+4Kr_s/r_2}-(K-1)} \\ &= S_1r_1-S_1\frac{2r_1}{\sqrt{(K-1)^2+4Kr_1/r_2}-(K-1)}, \end{aligned}$$

which is equivalent to $\bar\lambda \ge \bar\lambda^{NE}$, as claimed. □

4.A.5 Proof of Proposition 4.5.1

We shall break-down the proof of the proposition in three parts according to the three intervals which define the behaviour of the ratio $D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})/D_1(\bar{\lambda},\mathbf{r})$. In the following two lemmata, we present that the ratio is increasing in one of the intervals and decreasing in the other.

Lemma 4.A.3. *The ratio* $\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})}$ *is strictly increasing over the interval* $\left(\bar{\lambda}^{OPT},\bar{\lambda}^{NE}\right)$.

Proof. In order to prove that $\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})}$ is an increasing function of $\bar{\lambda}$ for $\bar{\lambda}\in(\bar{\lambda}^{OPT},\bar{\lambda}^{NE})$ we shall prove that

$$\frac{\frac{\partial}{\partial\bar{\lambda}}\left(D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})\right)}{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})} > \frac{\frac{\partial}{\partial\bar{\lambda}}\left(D_1(\bar{\lambda},\mathbf{r})\right)}{D_1(\bar{\lambda},\mathbf{r})}.$$

Since $\bar{\lambda}<\bar{\lambda}^{NE}$, we have $D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})=\frac{\bar{\lambda}}{r_1-\frac{\bar{\lambda}}{S_1}}$ and

$$\frac{\partial D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{\partial\bar{\lambda}} = \frac{r_1}{(r_1-\frac{\bar{\lambda}}{S_1})^2} = \frac{r_1}{\bar{\lambda}(r_1-\frac{\bar{\lambda}}{S_1})}D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r}),$$

from which we deduce that

$$\frac{\frac{\partial}{\partial\bar{\lambda}}D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})} = \frac{r_1}{\bar{\lambda}(r_1-\frac{\bar{\lambda}}{S_1})} = \frac{r_1}{\bar{\lambda}^2}\,D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r}).$$

For the centralized setting, $\bar{\lambda}\geq\bar{\lambda}^{OPT}$ means that

$$D_1(\bar{\lambda},\mathbf{r}) = S_1\frac{y_1}{r_1-y_1}+S_2\frac{y_2}{r_2-y_2}.$$

According to (4.5), the derivative of $D_1(\frac{\bar{\lambda}}{K}e,\mathbf{r})$ gets modified as follows:

$$\frac{\partial D_1(\bar{\lambda},\mathbf{r})}{\partial\bar{\lambda}} = S_1\frac{r_1y_1'}{(r_1-y_1)^2}+S_2\frac{r_2y_2'}{(r_2-y_2)^2} = (S_1y_1'+S_2y_2')\frac{r_1}{(r_1-y_1)^2}.$$

The constraint $S_1y_1+S_2y_2=\bar{\lambda}$ implies that $S_1y_1'+S_2y_2'=1$ and hence

$$\frac{\partial D_1(\bar{\lambda},\mathbf{r})}{\partial\bar{\lambda}} = \frac{r_1}{(r_1-y_1)^2}.$$

It yields

$$\begin{aligned}\frac{\frac{\partial}{\partial\bar{\lambda}}D_1(\bar{\lambda},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})} &= \frac{r_1}{(r_1-y_1)^2}\frac{1}{\left[D_1(\bar{\lambda},\mathbf{r})\right]^2}D_1(\bar{\lambda},\mathbf{r}) \\ &= \frac{r_1}{\left[\bar{\lambda}\sqrt{\frac{r_1}{r_2}}+S_1y_1\left(1-\sqrt{\frac{r_1}{r_2}}\right)\right]^2}\ D_1(\bar{\lambda},\mathbf{r}).\end{aligned}$$

As a result, for $\bar{\lambda}\in(\bar{\lambda}^{OPT},\bar{\lambda}^{NE})$, $\frac{\frac{\partial}{\partial\bar{\lambda}}D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})} > \frac{\frac{\partial}{\partial\bar{\lambda}}D_1(\bar{\lambda},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})}$ is equivalent to

$$\frac{r_1}{\bar{\lambda}^2}D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r}) > \frac{r_1}{\left[\bar{\lambda}\sqrt{\frac{r_1}{r_2}}+S_1y_1\left(1-\sqrt{\frac{r_1}{r_2}}\right)\right]^2}\ D_1(\bar{\lambda},\mathbf{r}),$$

and

$$\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})} > \frac{\bar{\lambda}^2}{\left[\bar{\lambda}\sqrt{\frac{r_1}{r_2}}+S_1y_1\left(1-\sqrt{\frac{r_1}{r_2}}\right)\right]^2}.$$

Now, we assume that there exist $\bar{\lambda}\in(\bar{\lambda}^{OPT},\bar{\lambda}^{NE})$ such that $\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})}$ is not increasing with $\bar{\lambda}$.

Since $\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})} \geq 1$, it results

$$1 \leq \frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})} \leq \frac{\bar{\lambda}^2}{\left[\bar{\lambda}\sqrt{\frac{r_1}{r_2}}+S_1y_1\left(1-\sqrt{\frac{r_1}{r_2}}\right)\right]^2}.$$

Since we have that

$$\frac{\bar{\lambda}^2}{\left[\bar{\lambda}\sqrt{\frac{r_1}{r_2}}+S_1y_1\left(1-\sqrt{\frac{r_1}{r_2}}\right)\right]^2} \geq 1,$$

then

$$\frac{\bar{\lambda}}{\bar{\lambda}\sqrt{\frac{r_1}{r_2}}+S_1y_1\left(1-\sqrt{\frac{r_1}{r_2}}\right)} \geq 1,$$

because the numerator and the denominator are both positive. Thus, with (4.6), the above yields

$$\begin{aligned}1 &\leq \frac{\bar{\lambda}}{\bar{\lambda}\sqrt{\frac{r_1}{r_2}}+S_1\sqrt{r_1}\frac{\bar{\lambda}-S_2\sqrt{r_2}(\sqrt{r_2}-\sqrt{r_1})}{S_1\sqrt{r_1}+S_2\sqrt{r_2}}\left(1-\sqrt{\frac{r_1}{r_2}}\right)} \\ &\leq \frac{\bar{\lambda}(S_1\sqrt{r_1}+S_2\sqrt{r_2})}{\bar{\lambda}\sqrt{r_1}(S_1+S_2)-S_1S_2\sqrt{r_1}(\sqrt{r_2}-\sqrt{r_1})^2}.\end{aligned}$$

This is equivalent to

$$\bar{\lambda}\sqrt{r_1}(S_1+S_2) - S_1S_2\sqrt{r_1}(\sqrt{r_2}-\sqrt{r_1})^2 \leq \bar{\lambda}(S_1\sqrt{r_1}+S_2\sqrt{r_2}),$$

and after rearranging both sides of the expression, we arrive at the following condition

$$\bar{\lambda} \leq S_1\sqrt{r_1}(\sqrt{r_1}-\sqrt{r_2}) = \bar{\lambda}^{OPT},$$

that it is a contradiction since $\bar{\lambda} \in (\bar{\lambda}^{OPT}, \bar{\lambda}^{NEP})$.

□

Lemma 4.A.4. *The ratio* $\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})}$ *is strictly decreasing over the interval* $\left(\bar{\lambda}^{NE}, \overline{R}\right)$.

Proof. In the interval $\left(\bar{\lambda}^{NE}, \overline{R}\right)$ we know that all servers are used. Thus, according to Proposition 4.4.1, the ratio $\frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e},\mathbf{r})}{D_1(\bar{\lambda},\mathbf{r})}$ is decreasing as a function of $\bar{\lambda}$.

□

The proof of Proposition 4.5.1 results from Corollary 4.5.1, Lemma 4.A.3 and Lemma 4.A.4.

4.A.6 Proof of Lemma 4.5.4

Proof. From (4.8), we observe that

$$\lim_{S\to\infty} PoA(K,S) = \sup_{\beta} \lim_{S\to\infty} I_K\left(\frac{1}{S-1},\beta\right).$$

In order to compute the limit of the PoA, we shall first compute the limit of I_K and then we shall compute the supremum. Let $x = \sqrt{(K-1)^2+4K\beta} - (K-1)$. We rewrite (4.7) as

$$I_K\left(\frac{1}{S-1},\beta\right) = \frac{1}{2}\frac{x-2}{\frac{(S-1+\sqrt{\beta})^2}{S-1+\frac{2\beta}{x}} - S}.$$

For large S, we show that

$$\frac{(S-1+\sqrt{\beta})^2}{S-1+\frac{2\beta}{y}} - S \approx 2\sqrt{\beta} - 1 - \frac{2\beta}{x},$$

as follows:

$$\begin{aligned}\frac{(S-1+\sqrt{\beta})^2}{S-1+\frac{2\beta}{x}} &\approx \frac{(S-1)^2+2\sqrt{\beta}(S-1)+\beta}{(S-1)\left(1+\frac{2\beta}{(S-1)x}\right)} \\ &\approx \left(S-1+2\sqrt{\beta}+\frac{\beta}{(S-1)}\right)\cdot\left(1-\frac{2\beta}{(S-1)x}\right) \\ &\approx S-1+2\sqrt{\beta}-\frac{2\beta}{x}.\end{aligned}$$

Now that we have computed the limit of I_K as $S \to \infty$, we shall compute the supremum with respect to β. In order to do this, we shall show that the limit computed previously is an increasing function of β.

Denote $F_K(\beta) = \lim_{S\to\infty} I_K(1/(S-1),\beta)$. We shall show that it is an increasing function of β. Let $y = \sqrt{(K-1)^2+4K\beta}$. We write

$$F_K(\beta) = \frac{1}{2}\frac{y-(K+1)}{2\sqrt{\beta}-1-\frac{y+K-1}{2K}}.$$

Using that $\beta > 1$ we show that the derivative of $F_K(\beta)$ with respect to β is positive:

$$\frac{\partial F_K(\beta)}{\partial\beta} > 0 \iff y'\left[2\sqrt{\beta}-1-\frac{y+K-1}{2K}\right]-(y-(K+1))\left(\frac{1}{\sqrt{\beta}}-\frac{1}{2K}y'\right) > 0.$$

This expression can be simplified as follows:

$$2y'(\sqrt{\beta}-1)-(y-(K+1))\frac{1}{\sqrt{\beta}} > 0.$$

Since y' is the derivative of $y = \sqrt{(K-1)^2+4K\beta}$ with respect to β, we derive it and substituting in the above expression it yields:

$$\frac{4K(\sqrt{\beta}-1)}{y}-(y-(K+1))\frac{1}{\sqrt{\beta}} > 0.$$

Multiplying by $b\sqrt{\beta}$ and using that $y = \sqrt{(K-1)^2+4K\beta}$ we obtain

$$\begin{aligned}&4K\sqrt{\beta}(\sqrt{\beta}-1)-\left[(K-1)^2+4K\beta-(K+1)y\right] > 0 \\ \iff\ &4K\beta-4K\sqrt{\beta}-\left[(K+1)^2+4K(\beta-1)-(K+1)y\right] > 0.\end{aligned}$$

This expression can be simplified and it results:

$$-4K(\sqrt{\beta}-1)+(K+1)(y-(K+1)) > 0.$$

Now, we notice that $(y-(K+1) = \frac{4K(\beta-1)}{y+K+1}$ and thus

$$-4K(\sqrt{\beta}-1)+(K+1)\frac{4K(\beta-1)}{y+K+1} > 0.$$

Since $\beta - 1 = (\sqrt{\beta}-1)(\sqrt{\beta}+1)$, we get

$$-1+(K+1)\frac{(\sqrt{\beta}+1)}{y+K+1} > 0.$$

If we multiply this expression by $y+K+1$, it results $\sqrt{\beta}(K+1)-y > 0$. Now, we divide and multiply this expression by $\sqrt{\beta}(K+1)+y$ and we obtain

$$\frac{\beta(K+1)^2 - \left[(K+1)^2 + 4K(\beta-1)\right]}{\sqrt{\beta}(K+1)+y} > 0.$$

This expression can be simplified as follows:

$$(\beta-1)(K+1)^2 - 4K(\beta-1) > 0 \iff (K-1)^2(\beta-1) > 0,$$

and thus we have proved that $F_K(\beta)$ is a decreasing function of β.

□

4.A.7 Proof of Proposition 4.6.1

Proof. Let n be the number of servers used by the decentralized setting for an infinity number of dispatchers, with $n = 1, \ldots, S$. We observe that

$$\sqrt{(K-1)^2 + 4K\gamma(K)r_j} \to 2\gamma(\infty)r_j, \text{ when } K \to \infty.$$

We use this property in (4.11) to show that $\gamma(\infty) = \frac{n}{\sum_{j=1}^n r_j - \bar{\lambda}}$ and for the expression of $D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})$ given in Lemma 4.A.1, we obtain that

$$\lim_{K\to\infty} D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r}) = \frac{1}{2}\sum_{j=1}^n [2r_j\gamma(\infty) - 2] = \frac{n\sum_{j=1}^n r_j}{\sum_{j=1}^n r_j - \bar{\lambda}} - n.$$

Now, we evaluate $D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})/D_1(\bar{\lambda}, \mathbf{r})$ when $K \to \infty$ and it yields to

$$\lim_{K\to\infty} \frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})}{D_1(\bar{\lambda}, \mathbf{r})} = \frac{\frac{n\sum_{j=1}^n r_j}{\sum_{j=1}^n r_j - \bar{\lambda}} - n}{\frac{\left(\sum_{j=1}^n \sqrt{r_j}\right)^2}{\sum_{j=1}^n r_j - n} - n} = \frac{\bar{\lambda}}{\bar{\lambda} - \sum_{j=1}^n r_j + \frac{1}{n}\left(\sum_{j=1}^n \sqrt{r_j}\right)^2},$$

and we know that in heavy-traffic all the servers are used, i.e., when $\bar{\lambda} \to \overline{R}$, we have

$n = S$. Hence, it results

$$\lim_{\rho \to 1} \lim_{K \to \infty} \frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})}{D_1(\bar{\lambda}, \mathbf{r})} = \frac{S\ \overline{R}}{\left(\sum_{i=1}^{S} \sqrt{r_i}\right)^2}.$$

□

5

Path Discovery Algorithms

In this chapter, we study a complete graph where the values of the edges are initially unknown, but can be discovered querying an oracle. Given a performance metric that depends on the edge values, we seek to discover an optimal path between two nodes by means of uncovering the least possible amount of edge values. We aim to analyze the efficiency of algorithms that solve this problem.

This chapter is organized as follows. In Section 5.1 we give some motivation of this work and present the problem. We give the related work in Section 5.2 and in Section 5.3 we formulate the problem. We prove in Section 5.4 our lower bounds on the number of queries and in Section 5.5 we establish lower and upper bounds on the query ratio. In Section 5.6 we present the numerical experiments. Finally, in Section 5.7, some conclusions are drawn and future research directions are proposed. Some of the proofs of this work are given in Appendix 5.A.

5.1 Optimization in Networks with Unknown Edges Value

In this section, we introduce the problem that we analyze in this chapter. First, we explain the main motivation of this work in Section 5.1.1. Then, in Section 5.1.2 we present the Optimal Path Discovery problem.

5.1.1 Motivation

Consider a set of nodes located at various spots in the Internet, and imagine that a source node wants to deliver a message to a destination node with the best performance

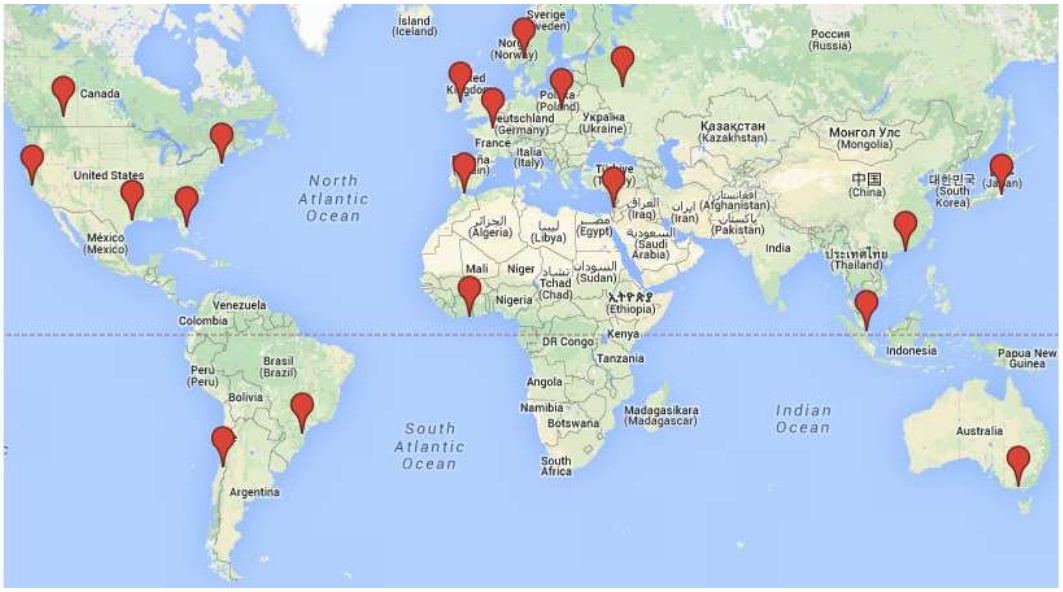

Figure 5.1: Location of the 19 nodes selected in the NLNog ring.

possible according to a certain metric. It may happen that the direct Internet path between the source and destination nodes has an unacceptable performance. In that case, provided that other nodes can act as relays for the message, it may be worth searching for an alternate path passing through one or more intermediate nodes. One can be even more ambitious and search for the best performance path in the complete graph formed by all nodes. However, monitoring the quality of the Internet paths between all pairs of nodes by sending probe packets can be excessively costly since the number of such paths has order n^2 if there are n nodes in a complete graph. Hence, it makes sense to minimize the monitoring effort required to discover the optimal path.

Prior to our study, we have performed measurements in the Internet that clearly show that path outages and performance degradations are routine events. To do so, we selected 19 nodes of the NLNog ring [3] and measured the latency and loss rates between all pairs of nodes every two minutes during one week. The localization of the 19 nodes is depicted in Figure 5.1.

The most important conclusions of these experiments are the following:

- For 65% of origin/destination pairs a path outage occurs at least once in the week. Moreover, 21% of these path outages lasted more than 4 minutes (and more than 14 minutes for 11% of them).

- The IP route is not the minimum latency path in 38% of the cases. There is always at least one origin/destination pair whose latency can be reduced by more than 76% by selecting an alternate path to the IP route.

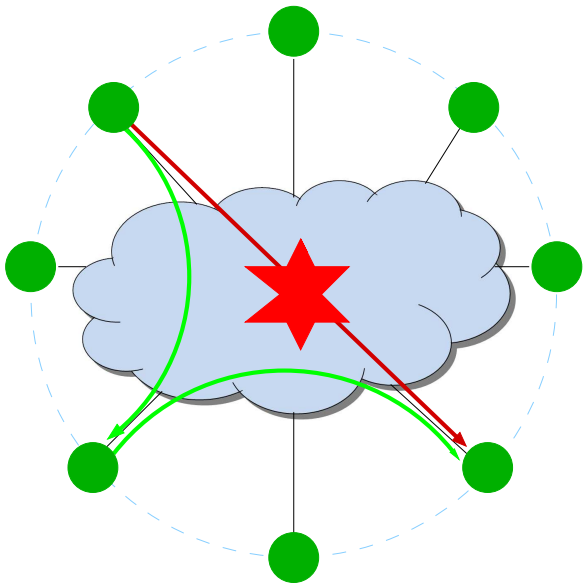

Figure 5.2: An alternative path is selected when the direct path fails.

- Similarly, more than 11% of the IP routes have a loss rate greater than 1%. By selecting alternate paths to those proposed by IP routing protocols, it would have been possible to have no loss at all.

According to these measurements, we can conclude that the performance of the paths selected by Internet routing protocols is not always optimal. Thus, instead of relying on the IP routing protocols, an alternative approach is to use an overlay network between the nodes. The basic idea is illustrated in Figure 5.2, where it is shown that, in case the IP route is unavailable or congested, an alternative path is selected. The selected path is usually one with optimal performance. To find such a path, we want to minimize the required monitoring effort since, as we already mentioned, monitoring the quality of the Internet paths between all pairs of nodes by sending probe packets can be very costly.

Another possible application of this work is the discovery of an optimal path in very large graphs. For such graphs, algorithms that need to store all the information of a graph before processing it cannot be implemented since data cannot fit it. Hence, it is important in large graphs to design algorithms that solve the desired problem minimizing the required information. In [39] the authors analyze the problem of large graphs and they suggest a relational approach for the shortest path case. We study analytically the performance of algorithms that solve the Optimal Path Discovery problem.

5.1.2 The Optimal Path Discovery problem

We consider a complete undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges, and the number of nodes is n. The real value of the edges are positive

and initially unknown, but can be asked. The performance of a path $P_{(s,t)}$ is given by its value $F(P_{(s,t)})$. Hence, for a given function F that defines the performance of paths, the objective is to discover a path P^* between s and t that optimizes the performance. In other words, we aim to find a path P^* from s to t such that $F(P^*) \leq F(P)$ for all $s-t$-paths P, in case of minimization problem, or a path P^* such that $F(P) \leq F(P^*)$ for all $s-t$-paths P in case of maximization problem. The issue is that the edge values are hidden. Hence, any algorithm that aims to solve the problem needs to discover the value of some edges to find an optimal path. The goal then is to find an optimal path by means of uncovering the least possible amount of edge values. We call to this problem the Optimal Path Discovery (OPD) problem.

To the best of our knowledge, the OPD problem has been studied in the literature only for particular cases. An example is [87] where the authors introduce the shortest path discovery problem. The shortest path discovery problem seeks to find the lower-cost path with the minimum number of queried edges. The conditions we require to F, in this chapter, are very general. Besides, depending on the objective function we consider and if the optimization goal is minimize or maximize, the OPD problem analyzes not only the shortest path but also other interesting problems such as obtaining the safest path or the widest path.

In this work we consider that the knowledge of the edges in the graph is homogeneous. We note that a general graph, not necessarily complete, can be considered as a complete graph where the knowledge of the edges is heterogeneous since the non-existent edges are initially known to be infinite. Hence, our assumption means that there is no previous knowledge of the topology of the graph and, thus, we need to concentrate on complete graphs.

We now present the main contributions of this work. We first prove that, for any instance with n nodes, any algorithm will need to query at least $n-1$ edges to find a solution of the problem. On the other hand, we observe that this lower bound is very optimistic and we also show that, for any algorithm, there exists a bad input such that the number of edges queried by the algorithm will be of the same order of magnitude than the total number of edges. The latter results suggest that the number of queries is not an appropriate measure to discriminate between algorithms for the OPD problem. We thus propose a new measure, the query ratio, to evaluate the performance of algorithms that solve the OPD problem. The query ratio of an algorithm is defined as the worst-case ratio (over all instances) of the number of queries made by an algorithm on an instance to the minimum number of queries required to find a feasible path for that instance. We prove that any algorithm has a query ratio of at least $1+\frac{4}{n}-\frac{8}{n^2}$ and propose an approximation algorithm that uncovers the same set of edges when it finds the optimal solution and whose query ratio is upper bounded by $2-\frac{1}{n-1}$.

We present in Section 5.1.1 the main motivation for the OPD problem. Prior to that, we show an example of the problem we analyze in this chapter.

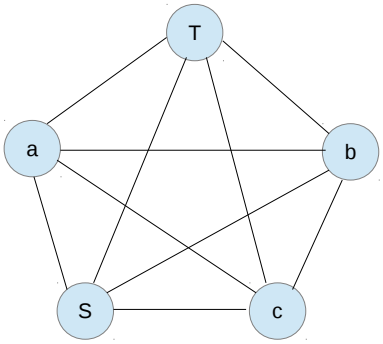

Figure 5.3: Initially unknown graph with 5 nodes.

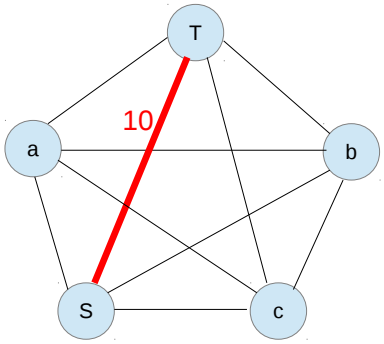

Figure 5.4: Algorithm queries the directed path from s to t in the first step.

Example of the OPD problem

We now give an example for a better understanding of the Optimal Path Discovery problem. In this example, we consider the shortest path discovery problem in a complete graph with 5 nodes. The values of the edges are initially unknown, but we know they are positive and that they can be queried. The objective is to discover the shortest path between s and t querying the minimum number of edges.

In Figure 5.3, we represent the initial state of the graph. For this graph, it is not possible to conclude which is the shortest path, since the value of all the edges is unknown. Hence, an algorithm that solves this problem queries the values of edges until it has queried enough edges to ensure that the shortest path between s and t is found. We emphasize that the only information available for the algorithm to know whether the shortest path is found or not, is the values of the previously queried edges and that the value of the unknown edges is positive.

Consider an algorithm that seeks to find the shortest path from s to t querying the minimum number of edges. First, the algorithm queries the direct path. We observe in Figure 5.4 that the value of the direct path is 10. Therefore, the algorithm needs to continue querying edges, since it does not know if there is any other path that is shorter. Hence, the algorithm continues querying edges until it can ensure that the shortest path is found.

We consider that the set of queried edges by algorithm to ensure that the shortest path is found is as given in Figure 5.5. We observe that in this situation it can be ensured that the shortest path is the direct path from s to t. In fact, any non-directed path from s to t is larger than 10, since it contains, at least, an edge whose values is higher than 10. Moreover, Figure 5.5 also illustrates that the number of edges that this algorithm has queried to find the shortest path is 6.

We know focus on the best algorithm that solves this instance. We observe in

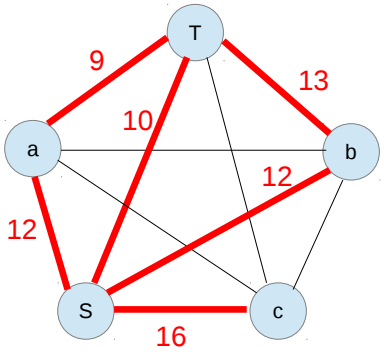

Figure 5.5: Algorithm stopped after 6 queries.

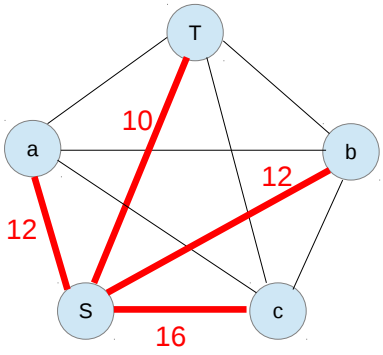

Figure 5.6: The minimum number of queries required is 4.

Figure 5.6 that, if we only query the edges (S,T), (S,a), (S,b) and (S,c), we can ensure that the shortest path from s to t is the direct path (S,T). We show this using the same argument as before, that is, any path with one or more hops must traverse these edges and, thus, we can conclude that it is not the shortest path. This result means that it is sufficient to query only 4 edges in this graph to find a shortest path.

We observe that the number of queried edges by the algorithm of Figure 5.5 is 6 while the the minimum number of queries required to solve the problem is 4. This means that the algorithm we have presented in Figure 5.5 is sub-optimal. Furthermore, for this graph, we have that the ratio of the number of queried edges by the algorithm and the minimum number of queries is 6/4. In this chapter, we measure the efficiency of algorithms that solve this problem using the notion of query ratio, which is defined as the worst possible case, among all the graphs of the same number of nodes, of the latter ratio.

5.2 Related Work

We introduce the OPD problem which is a general framework to find an optimal path according to a given metrics when the values of the edges are initially unknown. Hence, the goal is to find the optimal path querying the minimum number of edges. When the values of the edges are known, the optimal path finding has been studied only for particular cases of the objective function, see [59] for example for optimal algorithms in terms of time, space, and cost of solution path.

As we said in Section 5.1, the OPD problem has been only studied for particular cases. An example is given in [87] which introduces the shortest path discovery problem. In [87], the authors present a greedy algorithm that solves the shortest path discovery problem. The algorithm is greedy because it increments the search following the shortest

path known at each step. Such greedy criterion determines a family of algorithms. The authors show that for any algorithm not in this family, there exists an algorithm in the family that outperforms the one not in the family. In this work, they measure the performance of algorithms that solve this problem with the number of queries. In our work, we show that this measure, the number of queries, is inappropriate and we propose the query ratio as a measure of performance of algorithms that solve the OPD problem.

Algorithms such as Dijkstra's [30] or A^* [49] are classical tools that perform a single search that do no require previous information on the graph. However, they solve the OPD problem with a query ratio that can be as bad as $n/2$. To see this, consider an input in which the direct edge has value $1/2$, any other edge that is incident to the destination node has value 1, and the rest of the edges have value 0. Without loss of generality, we assume that the search algorithm starts its search in the source node. The algorithm will query all the edges incident to the source node. Then, it will pick any node different from the source and the destination, and will query all the edges incident to that node. The algorithm will repeat that process until it has queried all the edges incident to any node that is not the source nor the destination node. Hence, in total, such an algorithm will perform $n(n-1)/2$ queries. Nevertheless, the optimal certificate has size $(n-1)$.

Another related work is [7] where the authors present the use of agents to discover the edges of the graph or a shortest path from a source to a sink node. They give bounds on the number of agents required to discover directed and undirected graphs. The objective function of this problem is the number of agents used to discover the solution, which differs from the objective function of our problem which is to minimize the query ratio.

The authors of [70] study the shortest path problem when the graph is partially unknown in advance, but specified dynamically. These problems seek dynamic decision rules so that the total traversed distance has the best possible ratio to the shortest path. The authors describe optimal decision rules for two cases: layered graphs of width two, and two-optimal scenes with unit square obstacle. We observe that there exists a relation between the problem in [70] and our problem. However, the objective function are not the same since in the problem it is the ratio between distances, in our problem the objective function is the ratio between number of edges.

In the context of bioinformatics, the survey [19] presents a collection of results and methods in the area of combinatorial search, focusing on graph reconstruction using queries of different type. The authors aim to reconstruct a hidden graph from a class of graphs making as few queries as possible. The goal of such problem differs from the goal of our problem, since the OPD problem aims to find an optimal path. The survey [85] presents a compilation of results in the context of Shortest-Path Queries in Static Networks where the goal is to discover the shortest path of an unknown graph. The main difference of these problems with our problem is that Shortest-Path Queries in

Static Networks includes a preprocessing algorithm that may compute some information of the network to then be able to answer shortest-path or distance queries.

The algorithm we present in this chapter searches from the source and the sink node at the same time. This type of algorithms are known in the literature as bidirectional search algorithms and their study for the shortest path case has a long history, see [73, 64, 63]. For example, in [29] the authors suggest the Birectional Heuristic Front-to-Front algorithm i.e., the BHFFA2, which is know to be computationally expensive. The authors of [41, 28] present improved versions of the latter algorithm that are computationally less expensive.

5.3 Problem Formulation

In this section we formulate the OPD problem. First, we describe this problem and we give the main definitions in Section 5.3.1. In Section 5.3.2 we present our assumptions.

5.3.1 Model description

In the OPD problem, we seek to find the optimal path between two nodes. In order to discover this path, an algorithm has to uncover the unknown values of the edges. We use the abstraction of an oracle for that purpose. Let $\mathcal{O}$ be an oracle that can be accessed by an algorithm to *uncover* the value of an edge or a set of edges, i. e., the oracle is accessed by an algorithm by requesting the value of an edge or a set of edges. Hence, the oracle reveals to the algorithm the value of all the requested edges. For short, we say that an algorithm *uncovers an edge* when we refer to all this process.

We now give some definitions. Let $P_{(s,t)}$ be a path between the nodes s and t and $\delta_{(s,t)}$ be the value of the optimal path between s and t.

Definition 5.3.1. *The path $P_{(s,t)}$ is an* α-approximation *for the minimization case if it is satisfied that*

$$\frac{F(P_{(s,t)})}{\delta_{(s,t)}} \leq \alpha,$$

and, for the maximization case, if

$$\frac{\delta_{(s,t)}}{F(P_{(s,t)})} \leq \alpha.$$

When $\delta_{(s,t)}$ is totally unknown (no lower bound is known except for the fact that $\delta_{(s,t)}$ is positive and finite), there is no guarantee on the value of path $P_{(s,t)}$ with respect to that of an optimal path, and we say that $P_{(s,t)}$ is an ∞-*approximation*.

Definition 5.3.2. *A set of edges $\mathcal{C} \subseteq E$ is an* α-certificate *of G* if and only if

- *the value of the edges in $\mathcal{C}$ is known, whereas edges in $E \setminus \mathcal{C}$ have unknown values,*

- *$\mathcal{C}$ contains a path from s to t, the so called* proposed *path, and*
- *there are enough uncovered edges in $\mathcal{C}$ to guarantee that the proposed path is an α-approximation.*

We remark that every edge in the proposed path belongs to the α-certificate. Hence, its value is fully known. In all instances there exists at least one α-certificate for any α. Indeed, the set E that contains all the edges of the graph is an α-certificate for any α.

Definition 5.3.3. *An* instance *I of the OPD problem is formed by the following elements:*

- *a complete undirected graph $G = (V, E)$ with n nodes,*
- *two nodes s and $t \in V$,*
- *a targeted approximation factor $\alpha \geq 1$,*
- *a function $F : 2^E \to [0, \infty)$ that defines the value of any set of edges and*
- *an oracle $\mathcal{O}$ that can be accessed by an algorithm to* uncover *a set of edges.*

The size of an instance I is said to be the number of nodes of the fully connected graph G, and it is denoted by $|I|$.

Definition 5.3.4. *A feasible solution for the OPD problem is an α-certificate whose values are fully uncovered.*

Let us denote by $U(\mathcal{A}(I))$ the set of edges whose value has been uncovered by an algorithm $\mathcal{A}$ when solving the OPD problem on the instance I. In other words, the set $U(\mathcal{A}(I))$ is the α-certificate given by algorithm $\mathcal{A}$ as a solution for the instance I. We now define the number of queries of an algorithm that solves the OPD problem.

Definition 5.3.5. *The number of queries for instances of size n of an algorithm that solves the OPD problem is β_n if and only if*

$$|U(\mathcal{A}(I))| \leq \beta_n \, \forall \, I \text{ such that } |I| = n.$$

We note that there might be many α-certificates in an instance and an algorithm has to search for one. We are interested in the size of a smallest α-certificate of an instance, i.e., the $\alpha-$certificate with least number of edges. Let us denote by $|\mathcal{C}^{\alpha}_{\min}(I)|$ the size of a smallest α-certificate of instance I. Henceforth, we define the *query ratio* of an algorithm as follows.

Definition 5.3.6. *The* query ratio *of an algorithm $\mathcal{A}$ that proposes an α-approximation as a solution to solve the OPD problem is defined by the following ratio.*

$$\max_{I} \frac{|U(\mathcal{A}(I))|}{|\mathcal{C}^{\alpha}_{\min}(I)|}.$$

5.3.2 Our assumptions

As we said in Section 2.3, the value of a set of edges H is given by $F(H)$, where $F : 2^E \rightarrow [0, \infty)$ is a function that depends on the values of the edges of H.

We consider that the function F satisfies the following conditions:

- given $H \subset E$, it holds that,

$$F(H) = 0 \iff H = \{\emptyset\},$$

- given $H, H', H'' \subseteq E$ such that $H \cap H'' = \emptyset$ and $H \cap H' = \emptyset$, it holds that

$$\text{if } F(H') \leq F(H'') \Rightarrow F(H \cup H') \leq F(H \cup H''),$$

- given $H, H' \subseteq E$, where H' proper subset of H, it holds that

$$F(H') < F(H), \text{ for the minimization case and}$$

$$F(H') > F(H), \text{ for the maximization case.}$$

The last condition establishes the monotonicity property of the function F. From this condition, we can say that, for the minimization case, the function F, applied to any subset of edges of a path, is a lower-bound of the value of the of this path. On the other hand, for the maximization case, the function F, applied to any subset of edges of a path, is an upper-bound of the value of the of this path. In particular, all non-directed path $P_{(s,t)}$ satisfies that for all $e \in P_{(s,t)}$ $F(e) < F(P_{(s,t)})$ for the minimization case and $F(e) > F(P_{(s,t)})$ for the maximization case.

We also assume that F of an edge coincides with the value of this edge, i.e., $F(e) = f(e) > 0$, for all $e \in E$.

We now present two interesting problems for networking researchers that arise as particular cases of the function F we consider:

- in the case where $F = \sum_{e \in P_{(s,t)}} f(e)$, where $f(e)$ is the length of the edge $e \in E$, the value of a path represents its length. In this case, the OPD problem finds the shortest path between two nodes. This problem is known in the literature as the *Shortest Path Discovery Problem* [87]. It has applications for the discovery of minimum-latency paths in communication networks.

- in the case where F, applied to a path, represents the probability of successful travel of a packet in this path, which can be written as $\prod_{e \in P_{(s,t)}} p(e)$, where $p(e) \in [0, 1]$ is the probability that a packet is not lost in edge e, the OPD problem finds the path that maximizes the probability of packets successful arrival to its destination.

We also need to assume that our function F is such that an optimal path can be computed in polynomial time when the values of the edges are known.

For clarity of the presentation, in the following sections we focus on the case where the optimization goal is to minimize. However, all the techniques can be used to analyze maximization case and the results obtained for both cases coincide.

5.4 Number of Queries as Efficiency Measure

In this section, we concentrate on the number of queries of algorithms that solve the OPD problem. We first show that a set of edges must contain a cut-set of G in order to ensure to be a solution of the OPD problem. Then, we present an instance with n nodes such that any algorithm needs to uncover more than $n^2/4$ edges.

We first show that any α-certificate contains a cut set of G.

Lemma 5.4.1. *For any instance of the OPD problem with $\alpha \geq 1$, all α-certificate contain a cut-set in G such that the corresponding cut places s in one set of the partition and t in the other.*

Proof. First, we remark that an α-certificate contains a proposed path by definition. Therefore, the value $F(P_{(s,t)})$ of the proposed path $P_{(s,t)}$ is fully determined. Second, we remark that, in order to provide any finite approximation guarantee, the α-certificate needs to provide a bound for the value of a optimal path between s and t. Otherwise, no bound for the value of an optimal path between s and t (except for the fact that they are positive and finite).

Now, consider an instance I of optimal path discovery problem and an α-certificate $\mathcal{C}$. Let us assume that the α-certificate does not contain a cut-set that separates s and t. Hence, there exists a path $P^*_{(s,t)}$ between s and t so that $P^*_{(s,t)} \cap \mathcal{C} = \emptyset$. Since only edges in $\mathcal{C}$ have a known value, the value of $P^*_{(s,t)}$ is totally unknown for the α-certificate $\mathcal{C}$, implying that $\mathcal{C}$ cannot give a bound on an optimal path. Thus, the α-certificate $\mathcal{C}$ cannot guarantee any finite approximation for its proposed solution. Therefore, there is a contradiction because $\mathcal{C}$ is not an α-certificate. □

We now use the definition of any finite $\alpha \geq 1$ and α-approximation of Section 5.3 to characterize an α-approximation of the OPD problem.

Lemma 5.4.2. *For any instance of the OPD problem, let $\mathcal{C}$ be any set of edges that contains a path between s and t and a cut-set in G such that the corresponding cut places s in one part and t in the other. Hence, $\mathcal{C}$ is an α-certificate for some finite $\alpha \geq 1$.*

Proof. Consider a set of edges $\mathcal{C}$ as in the statement of the lemma. Let us denote by $P^{\mathcal{C}}_{(s,t)}$ the path in $\mathcal{C}$ between s and t. It holds that $P_{(s,t)} \cap \mathcal{C} \neq \emptyset$ for any path $P_{(s,t)} \in \mathcal{P}_{(s,t)}$.

Since $\min\{F(P_{(s,t)} \cap \mathcal{C}) : P_{(s,t)} \in \mathcal{P}_{(s,t)}\} \leq \delta^*_{(s,t)}$, it holds that

$$F(P^{\mathcal{C}}_{(s,t)}) \leq \frac{F(P^{\mathcal{C}}_{(s,t)})}{\min\{F(P_{(s,t)} \cap \mathcal{C}) : P_{(s,t)} \in \mathcal{P}_{(s,t)}\}} \delta^*_{(s,t)} = \alpha\ \delta^*_{(s,t)}.$$

This implies that $\mathcal{C}$ is an $\alpha-$certificate for $\alpha < \infty$. □

According to the result of Lemma 5.4.1 any *alpha*-approximation must contain a cut-set. This result enables us to present lower bounds for the number of queries β_n required so that an algorithm can guarantee a finite α-approximation.

Corollary 5.4.1. *For any algorithm that solves the OPD problem for a finite approximation $\alpha \geq 1$, it holds that $\beta_n \geq n - 1$.*

Proof. It is a direct consequence of Lemma 5.4.1 and the fact that the smallest cut-set in the complete graph has size $n - 1$. □

Nevertheless, the previous lower bound for β_n is optimistic since there exist cases in which any algorithm needs to uncover strictly more than $n-1$ edges in order to provide a finite approximation.

Lemma 5.4.3. *For any finite approximation $\alpha \geq 1$ and any integer $1 \leq p \leq n/2$, there exists an instance of the OPD problem so that any algorithm requires at least $p \cdot (n-p)$ uncovered edges in order to provide an α-approximation.*

Proof. We prove this lemma via the construction of an input that certifies the conditions stated in the lemma. The direct edge (s, t) has value 1. Let us split the set of nodes in one set of size p and one set of size $n - p$ that leaves s in a group and t in the other. The value of each edge with its end points in different sets (except for the direct edge) is $f(e) = \alpha \geq 1$. Besides, for each edge with its two end points in the same set we set $f(e) = \epsilon$, where ϵ is positive and small.

We now use that for any path P it holds that $f(e) < F(P)$ to prove that the only α-approximation is the direct path (s, t). To see this, we notice that the value of all the non-directed paths is strictly higher than α since they contain at least one edges whose value is α, whereas the value of the direct path is one. However, in order to guarantee such condition, any algorithm needs to uncover at least the cut-set of size $p \cdot (n-p)$. □

From the previous result, we can conclude that there exists an input such that the number of uncovered edges to provide an α-approximation is $n^2/4$.

Corollary 5.4.2. *For any algorithm $\mathcal{A}$ that solves the OPD problem for an instance I of size n such that $\alpha \geq 1$, there exists an input so that $\mathcal{A}$ requires at least $n^2/4$ uncovered edges in order to provide an α-approximation.*

According to Corollary 5.4.2, we consider meaningless the number of queries β_n of an algorithm. Indeed, for any algorithm and whatever the value of $\alpha \geq 1$, it is always possible to find a bad instance such that the number of edges uncovered by the algorithm will be of the same order of magnitude than the total number of edges. Therefore we change our aim. In the rest of this work, we focus on the study of the query ratio of algorithms that proposes α-approximations to solve the OPD problem. We believe that the query ratio is a fair measure to evaluate the performance of algorithms for these problems since it expresses how far is the number of queries asked by an algorithm with respect to the best possible any algorithm can perform in that instance.

5.5 Query Ratio Analysis

In this part of the thesis, we concentrate on the study of the query ratio of algorithms that solve the OPD problem. We first present a lower bound on the query ratio via the design of an adversary constructed for any algorithm. Then, we present an algorithm and give an upper-bound of its query ratio.

5.5.1 Lower Bound on the Query Ratio

We now present a lower-bound on the query-ratio of any algorithm that solves the OPD problem. We show that for any algorithm, there exists an input I such that the ratio $|U(\mathcal{A}(I))|/|\mathcal{C}^{\alpha}_{\min}(I)|$ is strictly higher than one, for any value of α.

Lemma 5.5.1. *For any algorithm $\mathcal{A}$ that solves the OPD problem with $\alpha \geq 1$, there exists a particular instance I of size n such that*

$$\frac{|U(\mathcal{A}(I))|}{|\mathcal{C}^{\alpha}_{\min}(I)|} \geq 1 + \frac{4}{n} - \frac{8}{n^2}.$$

Proof. See Appendix 5.A.1. □

A direct consequence of Lemma 5.5.1 is the following theorem.

Theorem 5.5.1. *For any algorithm $\mathcal{A}$ that solves the OPD problem with $\alpha \geq 1$, it holds that*

$$\max_{I} \frac{|U(\mathcal{A}(I))|}{|\mathcal{C}^{\alpha}_{\min}(I)|} \geq 1 + \frac{4}{n} - \frac{8}{n^2}.$$

We say that an algorithm solves optimally the OPD problem if its query ratio is one. Hence, according to the result of Theorem 5.5.1, there is no algorithm that solves the OPD problem optimally.

Algorithm 2 Algorithm for Optimal Path Discovery Problem

1: INITIALIZE sets $m_s = \{\emptyset\}$, $m_t = \{\emptyset\}$;
paths $PATH_{prop} = \emptyset$;
paths $PATH_{(s,u)} = \emptyset$, and $PATH_{(u,t)} = \emptyset \, \forall u \in V/\{s,t\}$;
variable $approx = \infty$;
variables $s^* = s$ and $t^* = t$;
2: **while** $approx > \alpha$ **do**
3: UPDATE $m_s := m_s \cup s^*$.
4: UPDATE $m_t := m_t \cup t^*$.
5: QUERY $\{(s^*, t^*)\}$.
6: QUERY $\{(s^*, u) : \forall u \in V/\{m_s \cup m_t\}\}$.
7: QUERY $\{(u, t^*) : \forall v \in V/\{m_s \cup m_t\}\}$.
8: COMPUTE the optimal path from s to t containing only uncovered edges.
9: UPDATE $PATH_{prop}$ to the optimal path from s to t containing only uncovered edges.
10: COMPUTE the optimal su-path containing only uncovered edges $\forall \, u \in V/\{m_s \cup m_t\}$.
11: UPDATE $PATH_{(s,u)}$ to the optimal su-path containing only uncovered edges $\forall \, u \in V/\{m_s \cup m_t\}$.
12: COMPUTE the optimal ut-path containing only uncovered edges $\forall \, u \in V/\{m_s \cup m_t\}$.
13: UPDATE $PATH_{(t,u)}$ to the optimal ut-path containing only uncovered edges $\forall \, u \in V/\{m_s \cup m_t\}$.
14: COMPUTE $s^* := \text{argmin}_{u \in V/\{m_s \cup m_t\}} F(PATH_{(s,u)})$.
15: COMPUTE $t^* := \text{argmin}_{u \in V/\{m_s \cup m_t\}} F(PATH_{(u,t)})$.
16: UPDATE $approx := \frac{F(PATH_{prop})}{F(PATH_{(s,s^*)} \cup PATH_{(t^*,t)})}$.
17: **end while**
18: RETURN $P_{(s,t)} := PATH_{prop}$.

5.5.2 Upper-bound of the Query Ratio

In this section, we present an algorithm that proposes an α-approximation as a solution of the OPD problem. We remark that in the proposed algorithm α is a parameter, and it can be set arbitrarily. Therefore, the proposed algorithm is a parametrized algorithm that guarantees the approximation factor that we arbitrarily decide to obtain. We compute the query ratio of such an algorithm proving that it never queries more than two times the size of the smallest certificate of an instance, when it proposes an optimal solution. We first analyze the algorithm that solves the OPD problem. A precise description of the algorithm is presented in Algorithm 2.

We now explain how the algorithm is executed. First, it initializes the values s^* and t^* to s and t respectively (see line 1 in Algorithm 2). The algorithm works in rounds. At each round, the algorithm advances one step further in a double search that starts from nodes s and t. First, Algorithm 2 adds s^* (resp. t^*) to the set m_s (resp. m_t).

Then, it queries the edge (s^*, t^*) as well as all the edges of the form (s^*, u) and (u, t^*) where u are all the nodes not belonging to m_s or m_t (see lines 5 to 7 in Algorithm 2). The algorithm then computes the optimal path between s and t that contains only uncovered edges which is denoted $PATH_{prop}$ (see lines 8 and 9 in Algorithm 2). Then, the algorithm picks the closest nodes to s and t among the nodes that have not been previously picked. The closest node to s (resp. t) is denoted by s^* (resp. t^*) (see lines 14 and 15 in Algorithm 2). Finally, the algorithm computes whether $PATH_{prop}$ is an α-approximation in line 16. If that is the case, Algorithm 2 stops and returns $PATH_{prop}$, otherwise, it iterates one more round.

The correctness of the algorithm follows directly from the following two facts. First, the algorithm proposes a fully uncovered path. Second, the proposed path is an α-approximation since in the last round it holds that $\frac{F(PATH_{prop})}{F(PATH_{(s,s^*)} \cup PATH_{(t^*,t)})} \leq \alpha$ and $F(PATH_{(s,s^*)} \cup PATH_{(t^*,t)})$ is a lower bound on $\delta^*_{(s,t)}$.

We now analyze the query ratio of Algorithm 2. To do so, we first compute the number of queries performed by the algorithm up to a generic round i.

Lemma 5.5.2. *For instance I with $\alpha \geq 1$, the number of queried edges by Algorithm 2 up to the i-th round is equal to $i(2n - 2i - 1)$.*

Proof. At each round, the algorithm queries $2|V/\{m_s \cup m_t\}| + 1$ edges according to lines 5 to 7 of Algorithm 2. At each round, the sizes of m_s and m_t increases in one element. Note that s^* and t^* are different at each round. Otherwise, if at some round $s^* = t^*$, in the previous round the algorithm would have stopped since the union of the optimal path from s to s^* and the optimal path from t^* to t would have been the optimal path from s to t. Hence, *approx* would have been equal to 1. Therefore, at the j-th round, the size of $V/\{m_s \cup m_t\}$ is equal to $(n - 2j)$.

Thus, the number of edges queried by the algorithm up to round i is the sum of the queries at all the previous rounds, i.e.,

$$\sum_{j=1}^{i} (2(n - 2j) + 1) = i(2n - 2i - 1).$$

□

On the other hand, if Algorithm 2 stops after i rounds, we are able to give a lower bound on the size of the smallest certificate of the instance. We give such a lower bound in the following lemma.

Lemma 5.5.3. *If Algorithm 2 stops after i rounds in an instance I, the size of the minimum 1-certificate of the instance is at least $i(n - i)$, i.e.,*

$$|\mathcal{C}^1_{\min}(I)| \geq i(n - i).$$

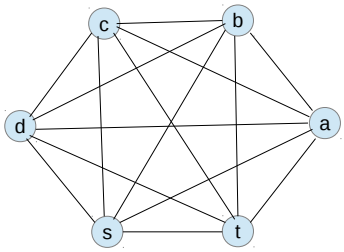

Figure 5.7: Initially unknown complete graph with 6 nodes.

Proof. See Appendix 5.A.2. □

We are now in position to present the query ratio of Algorithm 2.

Theorem 5.5.2. *Let $\mathcal{A}^{OPD}$ denote Algorithm 2. Therefore, for the query ratio of $\mathcal{A}^{OPD}$, it holds:*

$$\max_{I} \frac{|U(\mathcal{A}^{OPD}(I))|}{|\mathcal{C}^1_{\min}(I)|} \leq 2 - \frac{1}{n-1}.$$

Proof. From Lemmas 5.5.2 and 5.5.3, it holds:

$$\frac{|U(\mathcal{A}^{OPD}(I))|}{|\mathcal{C}^1_{\min}(I)|} \leq \frac{i(2n-2i-1)}{i(n-i)} = 2 - \frac{1}{n-i} \leq 2 - \frac{1}{n-1}.$$

□

We now present an example where Algorithm 2 finds the shortest path between two given nodes.

5.5.3 Example of Algorithm 2

We consider a complete graph with 6 nodes where the edges values are initially unknown, as shown in Figure 5.7. We focus on the shortest path case between the nodes s and t, i.e., the function F we consider is the sum, i.e., $F(H) = \sum_{e \in H} f(e)$ and we have that $\alpha = 1$.

Algorithm 2 initializes the sets m_s and m_t to s and t, respectively. In Figure 5.8, we represent the set of edges that Algorithm 2 queries in the first step. This set is formed by the following edges: (s,t), (s,a), (s,b), (s,c), (s,d), (a,t), (b,t), (c,t) and (d,t). Then, the algorithm sets the values of s^* and t^* to d and a, respectively, and the shortest path that the algorithm proposes is $PATH_{prop} = (s,t)$. Furthermore, after the

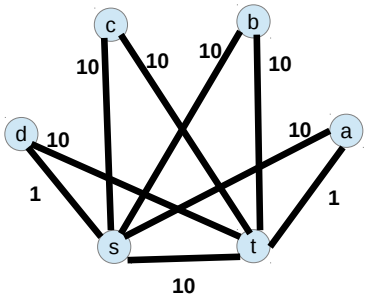

Figure 5.8: Set of edges queried by Algorithm 2 in the first step

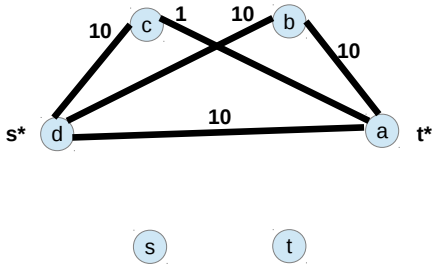

Figure 5.9: Set of edges queried by Algorithm 2 in the second step

first step, the value of the approximation factor is the following

$$\frac{F(PATH_{prop})}{F(PATH_{(s,s^*)} \cup PATH_{(t^*,t)})} = \frac{F(s,t)}{F((s,d) \cup (a,t))} = \frac{10}{1+1} = 5,$$

and it is higher than the desired α, which is one. Hence, the algorithm does not stop in this first step.

In the second step, node s^* is added to the set m_s and node t^* to the set m_t, which gives $m_s = \{s\} \cup \{s^*\}$ and $m_t = \{t\} \cup \{t^*\}$. As a consequence, we have that $V/m_s \cup m_t = \{b, c\}$. Thus, as it can be seen in Figure 5.9, the set of edges that are queried in the second step is formed by (s^*, b), (s^*, c), (b, t^*), (c, t^*) and (s^*, t^*), where $s^* = d$ and $t^* = a$.

We now show that in the second step the algorithm stops since it has found the shortest path. In Figure 5.10, we illustrate the edges that have been queried in the first and second steps by Algorithm 2. For a better visualization, the dashed lines represent the edges whose value is one and the continuous lines the edges with value equal to 10. The proposed path by the algorithm is again the direct path, i.e., $PATH_{prop} = (s, t)$. To see this, we observe that any other path from s to t with only uncovered edges traverses, at least, one edge whose value is 10, respectively. The algorithm updates the value of s^* and t^* to b and d respectively. For these values, it results that

$$F(PATH_{(s,s^*)} \cup PATH_{(t^*,t)}) = F(s,b) + F(c,t) = 10 + 10 = 20.$$

We notice that the value of (s, t) is 10, which results that the approximation ratio is less than 1, i.e.,

$$\frac{F(PATH_{prop})}{F(PATH_{(s,s^*)} \cup PATH_{(t^*,t)})} = \frac{10}{10+10} < 1.$$

Hence, since the approximation factor is less than one, the proposed path is the shortest path. We note that the algorithm has queried 15 edges to obtain the shortest

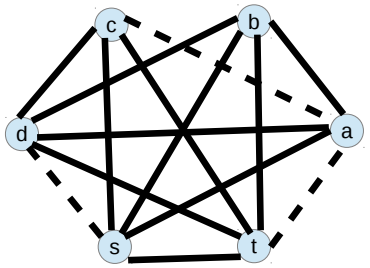

Figure 5.10: Set of queried edges by Algorithm 2. Value of dashed lines (resp. continuous lines) is one (resp. ten).

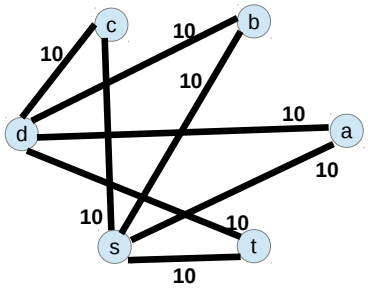

Figure 5.11: Minimum set of edges to be queried to find the shortest path from s to t.

path from s to t.

We now focus on the size of the minimum certificate of this instance. In Figure 5.11, we present the set of edges that must be queried to ensure that the shortest path between s and t is found. We observe that this set is formed by the following edges: (s,t), (s,b), (s,c), (a,d), (b,d) and (c,d). Besides, the size of the minimum certificate is 6. However, the number of queries required by the algorithm to find the shortest path is 16. Hence, for this instance, we have that

$$\frac{|U(\mathcal{A}(I))|}{|\mathcal{C}^1_{min}|} = \frac{14}{8} = 1.75,$$

which is less than the upper bound we derived in Theorem 5.5.2, as it can be observed below:

$$1.75 = \frac{14}{8} < 2 - \frac{1}{n-1} = 2 - \frac{1}{5} = 1.8.$$

5.5.4 Comparison with other algorithms

We first compare the performance of Algorithm 2 and the algorithm presented in [87] when it solves the OPD problem. Then, we analyze the query ratio of single search algorithms and we compare it the result obtained in Theorem 5.5.2.

We now describe a modification of the algorithm of [87] that solves the OPD problem. This algorithm works in rounds. In each round, it computes the optimal path between s and t with the current knowledge and the initial estimations. If all the edges of the optimal path have been previously queried, the algorithms stops and returns this path. Otherwise, it queries all the edges of this path and continues. Ties are broken giving priority to the paths with least number of edges.

The algorithm of [87], when it solves the shortest path discovery problem, requires some initial estimations of the values of the edges, that must be a less than the real value of the edges. We consider that all the edges are initially estimated to zero. In the

rest of the work, we denote by $\mathcal{A}_0^{Gre}$ this algorithm and by $\mathcal{A}^{OPD}$ Algorithm 2.

In the following lemma, we prove that the set of uncovered edges by Algorithm 2 and $\mathcal{A}_0^{Gre}$.

Lemma 5.5.4. *For any instance I with $\alpha = 1$, it holds that*

$$U(\mathcal{A}^{OPD}(I)) = U(\mathcal{A}_0^{Gre}(I)).$$

Proof. See Appendix 5.A.3. □

Using this result we can also give the query ratio of algorithm $\mathcal{A}_0^{Gre}$.

Theorem 5.5.3. *For the algorithm $\mathcal{A}_0^{Gre}$ we have that*

$$\max_I \frac{|U(\mathcal{A}_0^{Gre}(I))|}{|\mathcal{C}_{\min}^1(I)|} \leq 2 - \frac{1}{n-1}.$$

In Theorem 5.5.2 and Theorem 5.5.3 we prove an upper-bound of the query ratio of Algorithm 2 and algorithm $\mathcal{A}_0^{Gre}$. We observe that the query ratio of both algorithms when they find the optimal solution coincides.

We remark that the upper-bound given in Theorem 5.5.2 and Theorem 5.5.3 is tight and it is achieved when the size of the smallest certificate is $n-1$ and the algorithms uncover $2n-3$ edges.

We now compare the performance of Algorithm 2 with that of single search algorithms such as A^*. As we said in Section 5.2, the query ratio of the single search algorithms, such as A^*, is of order of n. We note that this value is much higher than the upper bound on the query ratio obtained for Algorithm 2 in Theorem 5.5.2.

5.6 Numerical Experiments

In this section, we experimentally analyze the algorithm presented in Section 5.5.2. We focus on the particular case of the shortest path discovery problem. As a consequence, we consider that F is the sum of the values of the edges of a path, i.e., $F(H) = \sum_{e \in H} f(e)$.

We first analyze the query ratio of Algorithm 2 when it finds the optimal solution in random graphs and then we compare it with the obtained upper-bound. First, we need to compute the size of the minimum certificate in a random graph. Hence, we now formulate this problem.

We denote by $f(e)$ the values of an edge $e \in E$ and $\delta^*_{(s,t)}$ is the shortest path between s and t. The minimum certificate of a graph is obtained as a solution of the following

optimization problem:

$$\begin{aligned} \min \quad & \sum_{e \in E} u(e) \\ \text{s.t.} \quad & \delta^*_{(,t)} \leq \sum_{e \in P_{(s,t)}} u(e) f(e), \forall P_{(s,t)} \in \mathcal{P}_{(s,t)} \\ & u(e) \in \{0,1\} \end{aligned}$$

According to this formulation, the minimum certificate is formed by the set of edges such that $u(e) = 1$. The latter means that the size of the minimum certificate is given by $|\mathcal{C}^{\alpha}_{min}| = \sum_{e \in V} u(e)$.

We consider 100 random graphs with 8 nodes where the values of the edges are uniformly distributed random numbers between zero and one, i.e., $f(e) \in (0,1)$. For these graphs, we execute Algorithm 2 and we calculate the size of the minimum certificate. In Table 5.1 we show the distribution of the values of the query ratio we obtained. For clarity, we only represent in Table 5.1 the values of the query ratio with frequency larger than 5%. We observe that the query ratio is 1 the 23% of the cases which means that our algorithm have solved the OPD problem optimally in 23 random graphs. On the other hand, the upper-bound given in Theorem 5.5.2 is attained 20 times which means that the bound obtained for our algorithm is tight. The mean of the query ratio over the 100 instances is 1.437.

Query Ratio	Frequency
1	23%
1.22	23%
1.2941	7%
1.6923	12%
1.8571	20%

Table 5.1: Distribution of the query ratio of the presented algorithm for 100 random graphs of 8 nodes

In the second set of experiments, we compare the performance of Algorithm 2 (Algorithm "algo" in Figure 5.12) with two modifications of the greedy algorithm presented in [87].

(a) The first algorithm at each step computes the shortest path according to the value of the uncovered edges and the value of the unknown edges set to a, and uncovers the edge of this path that is closest to the source. We refer to algorithm as deterministic and in Figure 5.12 is represented as "determ".

(b) The second algorithm at each step computes the shortest path according to the value of the uncovered edges and the value of the unknown edges set to 0, and

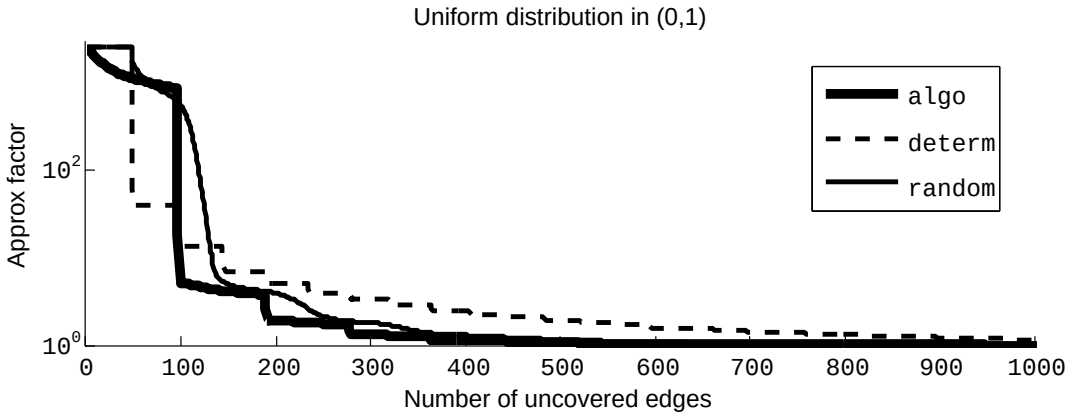

Figure 5.12: Approximation factor evolution comparison. Uniformly distributed edges and 50 nodes. Y axis in logarithmic scale.

uncovers an edge of this path picked uniformly at random. We refer to algorithm as random and in Figure 5.12 is represented as "rand".

We consider a random graph with 50 nodes where the values of the edges are uniformly distributed random numbers between zero and one. We execute Algorithm 2, the deterministic algorithm and the random algorithm for all the origin-destination pairs of this graph. In each execution of all the algorithms, we compute the evolution of the approximation factor with respect to the number of uncovered edges. We recall that the approximation factor is given by the ratio

$$\frac{F(PATH_{prop})}{F(PATH_{(s,s^*)} \cup PATH_{(t^*,t)})},$$

where $PATH_{prop}$ is the proposed path and $PATH_{(s,s^*)} \cup PATH_{(t^*,t)}$ is a lower bound on the shortest path. Hence, in Figure 5.12 we plot the average of the approximation factor over all the possible source-destination pairs to show the average performance of different algorithms. The y-axis of this figure is in the logarithmic scale.

In Figure 5.12 we show that the mean approximation factor of the deterministic algorithm decreases fast when the number of queries is 50 while the others decrease in a more constant fashion. However, this algorithm needs to uncover almost 1000 edges to achieve an approximation factor equal to one which is much higher than for the other algorithms. Moreover, we see that the average approximation factor of Algorithm 2 achieves the value 1 for a less number of uncovered edges comparing with the random algorithm. Furthermore, the average approximation factor of Algorithm 2 is smaller than the deterministic algorithm when the number of uncovered edges is higher than 100 and always less than the random algorithm.

5.7 Conclusions

In this chapter we presented the OPD problem. For a given function F that is applied to paths, algorithms that solve the OPD problem aim to find the path that optimizes its value minimizing the number of queries. We observed that interesting cases arise as particular cases of the OPD, for example the shortest path discovery problem or finding the path with maximum probability of successful arrival of packets to the destination. We first show that the number of queries does not measure correctly the performance of algorithms solving the OPD problem. However, we introduced the query ratio, a new measure which provides an important insight into the real quality of an algorithm solving these problems. That is because it compares the number of queries performed by the algorithm with the least amount of queries required to solve the problem. Therefore, we consider that the query ratio is the correct measure to take into account in the design of algorithms to solve the OPD problem. Finally, we have proposed an algorithm that searches from both ends and whose query ratio is upper bounded by 2, guaranteeing that the number of queries made by this algorithm is at most twice the minimum one for any instance.

5.A Appendix of Chapter 5

5.A.1 Proof of Lemma 5.5.1

In order to prove this result, we construct a bad instance I for each algorithm. The construction is made adversarially. We give a malicious adversary that acts as the oracle and, each time that an algorithm uncovers an edge, the adversary gives the value of that edge to the algorithm so that the performance of the algorithm is as bad as possible. The adversary is precisely described in Algorithm 3.

The adversary constructs an instance similar to the instance of lemma 5.4.3, but ad hoc to each algorithm. In the instance constructed by the adversary, the optimal path between s and t is always the direct edge (s, t). Furthermore, the direct path is the only α-approximation. On the other hand, the instance has a partition of the set of vertices with s in one set of the partition and t in the other set. The partition and the size of each sets of the partition depend on the algorithm. Edges that connect two nodes each in a different set of the partition have value α. While, edges that connect two nodes of the same set of the partition have value ϵ, where ϵ is positive and small.

Each algorithm can be seen as a sequence of edges to be uncovered. An algorithm poses a query to the oracle that is an edge to be uncovered. The oracle answers each query providing the value of that edge. In this particular construction, the adversary plays the role of the oracle. Edges to be uncovered by an algorithm can be grouped in four groups:

Algorithm 3 Adversary that construct a bad instance for any algorithm that solves the OPD problem, where $e_1, e_2 \ldots, e_l$ are the edges queried by the algorithm, i. e., e_i is the edge queried at the i-th step.

INITIALIZE sets $G_s = \{s\}$ and $G_t = \{t\}$;
functions $g(u) = 0$, $\forall u \in V/\{s,t\}$, $g(s) = s$, and $g(t) = t$.
for $i = 1, \ldots, l$ **do**
  if $e_i = (s,t)$ **then**
    REPLAY $f(e_i) = 1$.
  end if
  if $e_i = (s,u)$ such that $g(u) = 0$ **then**
    REPLAY $f(e_i) = \epsilon$ and update $g(u) := s$ **and** $G_s := G_s \cup \{u\}$.
  end if
  if $e_i = (s,u)$ such that $g(u) = t$ **then**
    REPLAY $f(e_i) = \alpha$.
  end if
  if $e_i = (u,t)$ such that $g(u) = 0$ **then**
    REPLAY $f(e_i) = \epsilon$ and update $g(u) := t$ **and** $G_t := G_t \cup \{u\}$.
  end if
  if $e_i = (u,t)$ such that $g(u) = s$ **then**
    REPLAY $f(e_i) = \alpha$.
  end if
  if $e_i = (u,v)$ such that $g(u) = g(v) = 0$ **then**
    REPLAY $f(e_i) = \epsilon$.
  end if
  if $e_i = (u,v)$ such that $g(u) = 0$ **and** $g(v) \in \{s,t\}$ **then**
    REPLAY $f(e_i) = \epsilon$.
    UPDATE $g(u) := g(v)$.
    UPDATE $g(u') := g(v) \forall u'$ such that there exists a path from u to u' composed by uncovered edges each with value equal to ϵ.
    UPDATE $G_{g(v)} := G_{g(v)} \cup \{u\} \cup \{u' : u' \rightarrow_\epsilon u\}$, where $u' \rightarrow_\epsilon u$ means that u and u' are connected by a path of uncovered edges each with value equal to ϵ.
  end if
  if $e_i = (u,v)$ such that $g(u) \in \{s,t\}$ **and** $g(v) \in \{s,t\}/g(u)$ **then**
    REPLAY $f(e_i) = \alpha$.
  end if
  if $e_i = (u,v)$ such that $g(u) \in \{s,t\}$ **and** $g(v) = g(u)$ **then**
    REPLAY $f(e_i) = \epsilon$.
  end if
end for

i) $\{(s,t)\}$,

ii) $\{(s,u) : u \in V/\{t\}\}$,

iii) $\{(u,t) : u \in V/\{s\}\}$ and,

iv) $\{(u,v) : u \wedge v \in V/\{s,t\}\}$.

The way in which the adversary determines the partition is described in the following lines. Nodes s and t are initially placed one in each set of the partition, let us denote these sets by G_s and G_t, respectively (see the initialization of function $g(s)$ and $g(t)$ in line 1 of Algorithm 3). If the algorithm queries the edge (s,t) to be uncovered, the adversary answers to the algorithm $f(s,t) = 1$ (see lines 3 and 4 in Algorithm 3). If the algorithm queries an edge of the second group to be uncovered and the node u has not been placed in any set of the partition, the adversary answers $f(s,u) = \epsilon$ and places u in the set G_s (see lines 6 and 7 in Algorithm 3). Otherwise, when u has been placed in a set of the partition (it can be only the set G_t), the adversary answers $f(s,u) = \alpha$ (see lines 9 and 10 in Algorithm 3). The adversary acts equivalently when the algorithm queries an edge of the third group. That is, if u has not been placed in any set of the partition, the adversary answers $f(u,t) = \epsilon$ and places u in the set G_t (see lines 12 and 13 in Algorithm 3). Otherwise, when u has been placed in a set of the partition (it can be only the set G_s), the adversary answers $f(u,t) = \alpha$ (see lines 15 and 16 in Algorithm 3). Finally, if the algorithm queries an edge of the fourth group, there exist four different cases: u and v have not been placed in any set of the partition, then the adversary answers $f(u,v) = \epsilon$ and none is placed in any set of the partition (see lines 18 and 19 in Algorithm 3). When one of them has been placed in a set of the partition, say u, and the other one has not, the adversary answers $f(u,v) = \epsilon$ and node v is placed in the same set than u. In this case, node v might have neighbours not yet assigned to a set of the partition as well. In that case, all the nodes in the same connected component than v are placed in the same set as nodes u and v (see lines 21 to 25 in Algorithm 3). Finally, if the two nodes have been placed in a set of the partition, the adversary answers $f(u,v) = \epsilon$ if the two nodes belong to the same set of the partition, or $f(u,v) = \alpha$ if they belong to different sets (see lines 27 to 31 in Algorithm 3).

We observe that the optimal path from s to t in any instance constructed by the adversary, regardless the algorithm, is the edge (s,t). Moreover, the only α-approximation is the direct path. On the other hand, for any algorithm, we obtain two sets G_s and G_t, according to the notation defined in Algorithm 3, that form a partition of V, i.e., $G_s \cap G_t = \emptyset$ and $G_s \cup G_t = V$. The sets G_s and G_t form a partition since every node is added either to the set G_s or to the set G_t. Moreover, since any algorithm needs to present an α-certificate and such certificate must contain a partition of V, therefore, every node is added to one set.

The smallest α-certificate consists in all the edges that connect the nodes of both sets of the partition G_s and G_t. Hence, it holds that $|\mathcal{C}^{\alpha}_{\min}(I)| = |G_s| \cdot |G_t|$. Moreover, we see that the number of uncovered edges by the algorithm is the sum of $|\mathcal{C}^{\alpha}_{\min}(I)|$ and the number of uncovered edges in each set of the partition. The graph formed by the uncovered edges in each set of the partition is connected, since by construction there exists a path from s (resp, t) to any node in G_s (resp, G_t). Therefore, the number of uncovered edges in each set of the partition is at least $|G_s| - 1$ and $|G_t| - 1$, respectively.

Thus, it holds that:

$$\begin{aligned}\frac{|U(\mathcal{A}(I))|}{|\mathcal{C}^{\alpha}_{\min}(I)|} &\geq \frac{|\mathcal{C}^{\alpha}_{\min}(I)| + |G_s| - 1 + |G_t| - 1}{|\mathcal{C}^{\alpha}_{\min}(I)|} \\ &= 1 + \frac{|G_s| + |G_t| - 2}{|G_s| \cdot |G_t|}.\end{aligned}$$

Since, it also holds that $|G_s| + |G_t| = n$, the fraction $\frac{|G_s|+|G_t|-2}{|G_s|\cdot|G_t|}$ yields its minimum value when $|G_s| = |G_t| = \frac{n}{2}$. Hence, we obtain

$$\frac{|U(\mathcal{A}(I))|}{|\mathcal{C}^{\alpha}_{\min}(I)|} \geq 1 + \frac{4}{n} - \frac{8}{n^2}.$$

5.A.2 Proof of Lemma 5.5.3

In this proof we use the following notation. We denote by $m_{s,j}$ (respectively, $m_{t,j}$) the set m_s (respectively m_t) at the end of the j-th round of the algorithm. We also denote by s^*_j (respectively t^*_j) the node that was added to m_s (respectively to m_t) at the j-th round of the algorithm. According to these definitions, we have that in round j,

$$m_{s,j} = m_{s,j-1} \cup \{s^*_j\},\ m_{t,j} = m_{t,j-1} \cup \{t^*_j\},$$

where $m_{s,0} = m_{t,0} = \{\emptyset\}$ and $s^*_1 = s$ and $t^*_1 = t$. We use $m_s = m_{s,i}$ and $m_t = m_{t,i}$ to denote the sets m_s and m_t after Algorithm 2 has finished. Moreover, for any two nodes u, v and a set of nodes $U \subseteq V$, we denote by $P^*_{(u,v)}|_U$ the optimal path between u and v in the graph induced by the set of nodes U.

First we show that the following inequality hold for all $1 \leq j \leq i$.

$$F(P^*_{(s,s^*_{j-1})}|_{m_{s,j-1}} \cup P^*_{(t^*_{j-1},t)}|_{m_{t,j-1}}) \leq F(P^*_{(s,s^*_j)}|_{m_{s,j}} \cup P^*_{(t^*_j,t)}|_{m_{t,j}}). \tag{5.1}$$

We first consider that the path $P^*_{(s,s^*_j)}$ visits s^*_{j-1} and $P^*_{(t^*_j,t)}$ visits t^*_{j-1}. In this case, we observe that

$$P^*_{(s,s^*_{j-1})}|_{m_{s,j-1}} \subset P^*_{(s,s^*_j)}|_{m_{s,j}},$$

which implies that

$$P^*_{(s,s^*_{j-1})}|_{m_{s,j-1}} \cup P^*_{(t^*_{j-1},t)}|_{m_{t,j-1}} \subset P^*_{(s,s^*_j)}|_{m_{s,j}} \cup P^*_{(t^*_j,t)}|_{m_{t,j}},$$

and it follows that

$$F(P^*_{(s,s^*_{j-1})}|_{m_{s,j-1}} \cup P^*_{(t^*_{j-1},t)}|_{m_{t,j-1}}) < F(P^*_{(s,s^*_j)}|_{m_{s,j}} \cup P^*_{(t^*_j,t)}|_{m_{t,j}}).$$

We now consider that the path $P^*_{(s,s^*_j)}|_{m_{s,j}}$ does not visit the node s^*_{j-1}, due to the

order in which Algorithm 2 chooses nodes s^*_{j-1} and s^*_j, it holds:

$$F(P^*_{(s,s^*_{j-1})}|_{m_{s,j-1}} \leq F(P^*_{(s,s^*_j)}|_{m_{s,j-1}-\{s^*_{j-1}\}}.$$

which implies that

$$F(P^*_{(s,s^*_{j-1})}|_{m_{s,j-1}} \cup P^*_{(t^*_{j-1},t)}|_{m_{t,j-1}}) \leq F(P^*_{(s,s^*_j)}|_{m_{s,j-1}-\{s^*_{j-1}\}\cup\{s^*_j\}} \cup P^*_{(t^*_{j-1},t)}|_{m_{t,j-1}}).$$

Otherwise, the algorithm would have picked s^*_j before s^*_{j-1}. Since the path $P^*_{(s,s^*_j)}|_{m_{s,j}}$ does not visit the node s^*_{j-1}, it also holds that

$$P^*_{(s,s^*_j)}|_{m_{s,j}} = P^*_{(s,s^*_j)}|_{m_{s,j-1}-\{s^*_{j-1}\}\cup\{s^*_j\}}.$$

Therefore, it holds:

$$\begin{aligned} F(P^*_{(s,s^*_{j-1})}|_{m_{s,j-1}} \cup P^*_{(t^*_{j-1},t)}|_{m_{t,j-1}}) &\leq F(P^*_{(s,s^*_j)}|_{m_{s,j-1}-\{s^*_{j-1}\}\cup\{s^*_j\}} \cup P^*_{(t^*_{j-1},t)}|_{m_{t,j-1}}) \\ &= F(P^*_{(s,s^*_j)}|_{m_{s,j}} \cup P^*_{(t^*_{j-1},t)}|_{m_{t,j-1}}). \end{aligned}$$

Now, if $P^*_{(t^*_j,t)}|_{m_{t,j}}$ visits t^*_{j-1} then with the same argument as before we state that

$$P^*_{(s,s^*_j)}|_{m_{s,j}} \cup P^*_{(t^*_{j-1},t)}|_{m_{t,j-1}} \subset P^*_{(s,s^*_j)}|_{m_{s,j}} \cup P^*_{(t^*_j,t)}|_{m_{t,j}},$$

and thus (5.1) holds since

$$F(P^*_{(s,s^*_j)}|_{m_{s,j}} \cup P^*_{(t^*_{j-1},t)}|_{m_{t,j-1}}) < F(P^*_{(s,s^*_j)}|_{m_{s,j}} \cup P^*_{(t^*_j,t)}|_{m_{t,j}}).$$

On the contrary, if $P^*_{(t^*_j,t)}|_{m_{t,j}}$ does not visit t^*_{j-1}, we use the previous reasoning to derive that

$$\begin{aligned} F(P^*_{(s,s^*_j)}|_{m_{s,j}} \cup P^*_{(t^*_{j-1},t)}|_{m_{t,j-1}}) &\leq F(P^*_{(s,s^*_j)}|_{m_{s,j}} \cup P^*_{(t^*_{j-1},t)}|_{m_{t-1,j}-\{t^*_{j-1}\}\cup\{t^*_j\}}) \\ &= F(P^*_{(s,s^*_j)}|_{m_{s,j}} \cup P^*_{(t^*_j,t)}|_{m_{t,j}}). \end{aligned}$$

In conclusion, (5.1) holds for all $1 \leq j \leq i$.

Second, we show that the following equation holds.

$$\frac{F(P_{(s,t)})}{F(P^*_{(s,s^*_i)}|_{m_s} \cup P^*_{(t^*_i,t)}|_{m_t})} > 1. \tag{5.2}$$

The optimal path proposed by the algorithm might be fully uncovered either at the very last round of the algorithm or it was uncovered in an earlier round. When the proposed path was uncovered in the last round of the algorithm, (5.2) holds because in this case the proposed path must visit s^*_i and t^*_i and thus $P_{(s,t)} = P^*_{(s,s^*_i)}|_{m_s} \cup P^*_{(s^*_i,t^*_i)} \cup$

$P^*_{(t_i^*,t)}|_{m_t}$, where $P^*_{(s_i^*,t_i^*)}$ is the optimal path from s_i^* to t_i^*. Hence $P^*_{(s,s_i^*)}|_{m_s} \cup P^*_{(t_i^*,t)}|_{m_t} \subseteq P_{(s,t)}$ which means that the ratio $F(P^*_{(s,s_i^*)}|_{m_s} \cup P^*_{(t_i^*,t)}|_{m_t}) < F(P_{(s,t)})$, i.e, the ratio of (5.2) is strictly higher than one.

In the case when the proposed path was uncovered in a round earlier than the last round of the algorithm, (5.2) holds because in the round $i-1$, the algorithm computes $\frac{F(P_{(s,t)})}{F(P^*_{(s,s_i^*)}|_{m_s} \cup P^*_{(t,t_i^*)}|_{m_t})}$ and the result has to be larger than 1 since the algorithm does not stop. Therefore it holds:

$$\frac{F(P_{(s,t)})}{F(P^*_{(s,s_i^*)}|_{m_s} \cup P^*_{(t_i^*,t)}|_{m_t})} > 1.$$

Now, using (5.1)-(5.2), we are able to show that, for all $1 \leq j \leq i$, it holds the following:

$$\frac{F(P_{(s,t)})}{F(P^*_{(s,s_j^*)}|_{m_{s,j}} \cup P^*_{(t_j^*,t)}|_{m_{t,j}})} > 1.$$

Therefore, for any $1 \leq j \leq i$ and any path that intersects $P^*_{(s,s_j^*)}|_{m_{s,j}}$ and $P^*_{(t_j^*),t}|_{m_{t,j}}$, any 1-certificate needs at least one edge not in $P^*_{(s,s_j^*)}|_{m_{s,j}} \cup P^*_{(t_j^*,t)}|_{m_{t,j}}$ to show that the proposed path is the optimal path. Now, for any pair of nodes s_j^*, t_j^*, consider the paths of the form $P^*_{(s,s_j^*)}|_{m_{s,j}} \cup P_{(s_j^*,t_j^*)}|_{V/\{m_{s,j} \cup m_{t,j}\}} \cup P^*_{(t_j^*,t)}|_{m_{t,j}}$, where $P_{(s_j^*,t_j^*)}|_{V/\{m_{s,j} \cup m_{t,j}\}}$ denotes a path between s_j^* and t_j^* in $V/\{m_{s,j} \cup m_{t,j}\}$. There exists at least $n-2j+1$ disjoint paths of the form previously described, one per each node in $V/\{m_{s,j} \cup m_{t,j}\}$ plus the path that connects directly s_j^* and t_j^*. Hence, at least $n-2j+1$ edges need to be present in any 1-certificate for each pair of nodes s_j^*, t_j^*.

Therefore, if we sum up all these edges, we obtain that any 1-certificate needs to contain at least the following amount of edges.

$$\sum_{j=1}^{i} n - 2j + 1 = in - i(i+1) + i = i(n-i).$$

5.A.3 Proof of Lemma 5.5.4

In this proof we use a similar notation than in the proof of Lemma 5.5.3, that is, we say that in round j Algorithm 2 uncovers the edge (s_j^*, t_j^*) and all the edges (s_j^*, u) and (u, t_j^*) for all u that do not belong to $m_{s,j} \cup m_{t,j}$.

The proof proceeds by an induction in the steps. Therefore, we define a common notion of step for both algorithms. We consider that at step j algorithm $\mathcal{A}_0^{Gre}$ does $n-2j+1$ rounds where, in each round, it computes the optimal path with the current information and uncovers all the edges of this path. A step for $\mathcal{A}^{OPD}$ is one round of Algorithm 2.

We want to show that the set of edges that both algorithms uncover when they

provide a 1-approximation is the same. To do that, we show that in each step j both algorithms uncover the same set of edges by an induction on the number of steps.

We now check that in the first step both algorithms uncover the same set of edges. Since all the edges are initially estimated to $a \geq 0$, the algorithm $\mathcal{A}_0^{Gre}$ finds that the optimal is the direct path (s,t). In the next $2(n-1)$ rounds, $\mathcal{A}_0^{Gre}$ finds that the optimal paths are all the two-hop paths. Hence, algorithm $\mathcal{A}_0^{Gre}$ uncovers the edge (s,t) and the edges (s,u) and (u,t) for all $u \in V/\{s,t\}$. On the other hand, Algorithm 2 also uncovers the mentioned set of edges in its first step.

We then suppose that until step j both algorithms have uncovered the same set of edges and they haven not stopped yet. We aim to prove that the set of uncovered edges after step j also coincide. At step j algorithm $\mathcal{A}_0^{Gre}$ does $n-2j+1$ rounds and in each round it computes the optimal path with the known information and uncovers all the edges of this path. We know that the optimal paths computed by $\mathcal{A}_0^{Gre}$ at step j must contain unknown edges since the algorithm has not stopped. Furthermore, at step j, the only edges that are unknown are those that connect nodes that do not belong to $m_{s,j-1} \cup m_{t,j-1}$. Hence, we have that

$$s_j^* = \underset{v \in V/\{m_{s,j-1} \cup m_{t,j-1}\}}{\operatorname{argmin}} F(P_{(s,v)}),$$

and

$$t_j^* = \underset{v \in V/\{m_{s,j-1} \cup m_{t,j-1}\}}{\operatorname{argmin}} F(P_{(v,t)}),$$

and the unknown edges are estimated to 0 which is the minimum possible value. This means that the optimal paths computed by $\mathcal{A}_0^{Gre}$ at step j are of the form $PATH_{(s,s_j^*)} \cup P_{(s_j^*,t_j^*)} \cup PATH_{(t_j^*,t)}$, where $P_{(s_j^*,t_j^*)}$ is either (s_j^*,t_j^*) or a path from s^* to t^* that traverses only one node that does not belong to $m_{s,j} \cup m_{t,j}$. Furthermore, in these paths the unique unknown edges are (s_j^*,t_j^*) and the edges (s_j^*,u) and (u,t_j^*) where u does not belong to $m_{s,j} \cup m_{t,j}$, which are the edges that uncovers Algorithm 2 at step j. Therefore, $\mathcal{A}_0^{Gre}$ uncovers the same set of edges that Algorithm 2 at step j.

6

Summary and Future Research

In this thesis, we analyzed the efficiency of telecommunication systems. Hence, we focused on the performance of various systems and we compared it with the optimal one. First, we focused in a single server queue model and we investigated the efficiency of a pricing technique of users that share a common resource. Then, we concentrated on multi-server systems and we explored the efficiency of the non-cooperative load balancing. Finally, we considered a network where the values of the edges are initially unknown. We looked at the efficiency of algorithms that find an optimal path between two given nodes.

Let us now present the main conclusions of this work, as well as the different directions our work can be taken further.

6.1 Single Server with Relative Priorities

We presented a non-cooperative game where users compete for the capacity of a resource. Each player chooses its payment, that is proportional of its priority to get service. The objective of each user is thus to minimize its payment while it is satisfied that its jobs are served before a given deadline. We assume that the capacity is shared according to the DPS model. In the DPS queue the rate allocated to each class depends on its relative priority. Due to the lack of explicit expressions of the response times in the DPS discipline, this game can be analyzed only in particular cases. For example, we gave the conditions of the existence of an equilibrium when the service requirements are exponential.

We also studied the efficiency of this game. We showed that the Price of Anarchy is one if the equilibrium is unique. Unfortunately, we are able to prove that there is a unique equilibrium when there are two players and the service time requirements are general. For the cases when this game cannot be solved in closed-form, we present an approximation based on a heavy-traffic result. In the simulations section, we measure the accuracy of the proposed approximation. We observed that the payments given by this approximation can provide a non negligible error if the users are heterogeneous. However, it captures correctly the structure of the solution and the performances are always accurate.

We identify some problems that this work leaves open. For example, we showed that the dynamics of the best-response algorithm converge to the Nash equilibrium in the following cases: *(i)* when the starting point is feasible, and *(ii)* when the number of players is two and the service time requirements is general. Hence, an interesting issue is considering the approximated game to show that the dynamics of the best-response algorithm converge to the Nash equilibrium for any point when the number of players is arbitrary.

We now present another possible extension of this work. In this game, we assumed that there is a minimum price to be paid by the users in order to get service. An interesting situation arises if we consider that there is also a maximum price that users are willing to pay. In this case, a player decreases its traffic if its price in the equilibrium exceeds the maximum price.

6.2 Non-cooperative Load Balancing

We compared the performance of two load balancing techniques in server farms. First, we studied a centralized architecture, where a single dispatcher receives all the incoming jobs. Then, we considered a decentralized architecture with finite number (strictly higher than one) of selfish dispatchers. We say that these dispatchers are selfish since each one receives a portion of the total traffic and seeks to minimize the mean response time of its jobs. Hence, for the decentralized case, a non-cooperative game can be defined.

The goal was to compare the performance of both architectures. This performance comparison have been previously done using the notion of Price of Anarchy, that is defined as the ratio of the performance in the worst equilibrium over the performance in the centralized architecture. We showed that, when the system load is very close to one, the performance of both systems is equal. This result is interesting since, in classical queueing theory, the mean response time tends to infinity when the load is almost one.

We investigated a server farm with servers of two different speeds. For a fixed values of the capacities, we focused on the ratio of the performance in the equilibrium and the

performance of the centralized architecture. We showed that this ratio is maximum when the decentralized setting starts using the slow servers. Furthermore, we determined that the maximum value of this ratio depends only on the ratio of the number of servers in each group and the ratio of the capacities. We also proved, for the case of a server with two different speeds, that the Price of Anarchy is achieved when there is one server that is infinitely faster than the rest of the servers. In the rest of the cases, we observed that the performance of both settings is similar.

We also investigated the non-atomic case, as a particular case when the number of dispatchers grows to infinity. For this instance, we proved that in heavy-traffic the performance of both settings is not equal. For a fixed values of the capacities, we proved the maximum of the ratio of the performance in the equilibrium and the performance of the centralized architecture is given when the decentralized setting starts using the slow servers. Besides, we presented that the value of the maximum can be written only in terms of the ratio of the number of servers in each group and the ratio of the values of the capacities. We gave the value of the of PoA for the non-atomic case, that is equal to the number of servers, which coincides with the result given by [52]. Otherwise, we saw that both settings have a similar performance.

The main limitation of this work is that most of the results are given for the case of a server farm of two different speeds. As future work, we have the inefficiency analysis of server farms for arbitrary values of the capacities. The study of the value of the PoA in the general case is also considered an interesting issue.

We assumed that the servers are PS queues, that is, all the the jobs are served simultaneously at equal rate. If we consider service differentiation in the servers, we need to define a different game for the decentralized architecture. In the atomic case, the jobs can choose their payment and the strategy of each dispatcher is the amount of traffic they send to each server. The equilibrium in this case is the strategy where no player (user or dispatcher) has incentive to change its strategy. In the non-atomic game, we can also define a game that takes into account service differentiation of users. For this case, each incoming job can choose the server where it gets service and the payment it makes. In the equilibrium, the jobs has no incentive to change the payment or the server where they get service.

6.3 Path Discovery Algorithms

We considered a complete undirected graph with n nodes where the values of the edges are initially unknown, but can be discovered asking to an oracle. Given two nodes and a function that applies to paths, we aim to find the best value path between these nodes, uncovering the minimum number of edges. This problem is the OPD problem.

We claimed that the number queried edge values is not the correct measure of the efficiency of algorithms that solve the OPD problem. Indeed, we showed that there exists

an input such that any algorithm needs to query more than $n^2/4$ edges, which is of the order of the number of edges. Hence, we suggested the query ratio, the ratio between the number of queried edge values and the least number of edge values required to solve the problem, to measure the efficiency of algorithms that solve the OPD problem.

We presented lower and upper bounds on the query ratio. First, we proved that, in a complete graph with n nodes, the query ratio of any algorithm is higher than $1 + 4/n - 8/n^2$. This result means that there is no algorithm that solves the OPD problem optimally. We then presented an algorithm whose query ratio is less than $2 - \frac{1}{n-1}$. We also analyzed the query ratio of single search algorithms and we concluded that its query ratio is of order of n.

Our work can be extended in divers directions. We note that there exists a gap between the lower bound and upper bound on the query ratio. The question we need to answer is whether there exists an algorithm, or on the contrary an adversary that produces a bad instance for any algorithm, so that this gap is closed.

We note that the OPD problem is formulated for a general graph, but we analyze the query ratio of algorithms that solve this problem in a complete graph. Hence, an open problem is to extend our results to a not necessarily complete graph. Another possibility is to show that, for a given algorithm, the minimum query ratio is given in complete graphs.

Bibliography

[1] http://www.datacenterknowledge.com/archives/2009/04/01/google-unveils-its-container-data-center/.

[2] http://www.datacenterknowledge.com/archives/2009/05/14/whos-got-the-most-web-servers/.

[3] NLNOG ring. https://ring.nlnog.net/.

[4] PANACEA Project (www.panacea-cloud.eu), European Communitys Seventh Framework Programme [FP7/2007-2013].

[5] SOP Project. ANR-11-INFR-001. http://projects.laas.fr/sop/.

[6] S. Aalto, U. Ayesta, S. Borst, V. Misra, and R. Núñez-Queija. Beyond Processor Sharing. *ACM SIGMETRICS Performance Evaluation Review*, 34:36–43, 2007.

[7] N. Alon, Y. Emek, M. Feldman, and M. Tennenholtz. Economical graph discovery. In *ICS*, pages 476–486, 2011.

[8] E. Altman, K. Avrachenkov, and U. Ayesta. A survey on discriminatory processor sharing. *Queueing systems*, 53(1-2):53–63, 2006.

[9] E. Altman, U. Ayesta, and B. Prabhu. Load balancing in processor sharing systems. *Telecommunication Systems*, 47(1-2):35–48, 2011.

[10] E. Altman, T. Boulogne, R. El-Azouzi, T. Jiménez, and L. Wynter. A survey on networking games in telecommunications. *Computers & Operations Research*, 33(2):286–311, 2006.

[11] J. Anselmi, U. Ayesta, and A. Wierman. Competition yields efficiency in load balancing games. *Performance Evaluation*, 68(11):986–1001, 2011.

[12] J. Anselmi and B. Gaujal. The price of forgetting in parallel and non-observable queues. *Performance Evaluation*, 68(12):1291 – 1311, 2011.

[13] K. Avrachenkov, U. Ayesta, P. Brown, and R. Núñez-Queija. Discriminatory processor sharing revisited. In *Proceedings of IEEE INFOCOM*, 2005.

[14] U. Ayesta, O. Brun, and B. J. Prabhu. Price of anarchy in non-cooperative load-balancing games. *Performance Evaluation*, 68:1312–1332, 2011.

[15] C. H. Bell and S. Stidham. Individual versus social optimization in the allocation of customers to alternative servers. *Management Science*, 29:831–839, 1983.

[16] D. Bertsekas and J. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, 1989.

[17] S. Borst, R. Núñez-Queija, and B. Zwart. Sojourn time asymptotics in processor-sharing queues. *Queueing Systems*, 53(1-2):31–51, 2006.

[18] S. Borst, D. van Ooteghem, and A. Zwart. Tail asymptotics for discriminatory processor sharing queues with heavy-tailed service requirements. *Performance Evaluation*, 61(2–3):281–298, 2005.

[19] M. Bouvel, V. Grebinski, and G. Kucherov. Combinatorial search on graphs motivated by bioinformatics applications: A brief survey. In *WG 2005. LNCS*, pages 16–27. Springer, 2005.

[20] O. Boxma, N. Hegde, and R. Núñez-Queija. Exact and approximate analysis of sojourn times in finite discriminatory processor sharing queues. *AEU International Journal on Electronic Communications*, 60:109–115, 2006.

[21] O. Brun, J. Doncel, and C. Thraves Caro. Shortest Path Discovery Problem, Revisited (Query Ratio, Upper and Lower Bounds). Research report, LAAS-CNRS, Oct. 2014.

[22] H. Chen, J. Marden, and A. Wierman. The effect of local scheduling in load balancing designs. In *ACM SIGMETRICS Performance Evaluation Review*, volume 36, pages 110–112, Aug. 2008.

[23] G. Christodoulou and E. Koutsoupias. The price of anarchy of finite congestion games. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 67–73. ACM, 2005.

[24] J. Cohen. The multiple phase service network with generalized processor sharing. *Acta Informatica*, 12:245–284, 1979.

[25] J. W. Cohen. *The single server queue*. Elsevier, 2012.

[26] R. Cominetti, J. R. Correa, and N. E. Stier-Moses. The impact of oligopolistic competition in networks. *Operations Research*, 57(6):1421–1437, 2009.

[27] M. E. Crovella, M. Harchol-Balter, and C. D. Murta. Task assignment in a distributed system (extended abstract): improving performance by unbalancing load. In *ACM SIGMETRICS Performance Evaluation Review*, volume 26, pages 268–269. ACM, 1998.

[28] H. W. Davis, R. B. Pollack, and T. Sudkamp. Towards a better understanding of bidirectioanl search. In *AAAI*, 1984.

[29] D. de Champeaux. Bidirectional heuristic search again. *Journal of the ACM*, 30(1):22–32, Jan. 1983.

[30] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[31] J. Doncel, U. Ayesta, O. Brun, and B. Prabhu. On the efficiency of non-cooperative load balancing. In *IFIP Networking Conference, 2013*, pages 1–9, May 2013.

[32] J. Doncel, U. Ayesta, O. Brun, and B. Prabhu. Is the price of anarchy the right measure for load-balancing games? *ACM Trans. Internet Technol.*, 14(2-3):18:1–18:20, Oct. 2014.

[33] J. Doncel, U. Ayesta, O. Brun, and B. Prabhu. A resource-sharing game with relative priorities. *Performance Evaluation*, 79:287 – 305, 2014. Special Issue: Performance 2014.

[34] G. Fayolle, I. Mitrani, and R. Iasnogorodski. Sharing a processor among many job classes. *Journal of the ACM*, 27(3):519–532, 1980.

[35] D. Feitelson, L. Rudolph, and U. Schwiegelshohn. Parallel job scheduling a status report. In D. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 3277 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin Heidelberg, 2005.

[36] H. Feng, V. Misra, and D. Rubenstein. Optimal state-free, size-aware dispatching for heterogeneous M/G/-type systems. *Performance evaluation*, 62(1):475–492, 2005.

[37] R. D. Foley and D. R. McDonald. Join the shortest queue: Stability and exact asymptotics. *The Annals of Applied Probability*, 11(3):pp. 569–607, 2001.

[38] E. J. Friedman. Genericity and congestion control in selfish routing. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 5, pages 4667–4672. IEEE, 2004.

[39] J. Gao, R. Jin, J. Zhou, J. X. Yu, X. Jiang, and T. Wang. Relational approach for shortest path discovery over large graphs. *Proc. VLDB Endow.*, 5(4):358–369, Dec. 2011.

[40] E. Gelenbe and I. Mitrani. *Analysis and Synthesis of Computer Systems*. Imperial College Press, 2009.

[41] S. Ghosh and A. Mahanti. Bidirectional heuristic search with limited resources. *Information Processing Letters*, 40(6):335 – 340, 1991.

[42] J. Gittins. Allocation indices for multi-armed bandits, 1989.

[43] J. Gittins, K. Glazebrook, and R. Weber. Multi-armed bandit allocation indices. *Wiley, UK*, 2011.

[44] S. Grishechkin. On a relationship between processor sharing queues and crump-mode-jagers branching processes. *Adv. Appl. Prob.*, 24(3):653–698, 1992.

[45] V. Gupta, M. Harchol Balter, K. Sigman, and W. Whitt. Analysis of join-the-shortest-queue routing for web server farms. *Performance Evaluation*, 64(9):1062–1081, 2007.

[46] M. Harchol-Balter. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013.

[47] M. Harchol-Balter, M. E. Crovella, and C. D. Murta. On choosing a task assignment policy for a distributed server system. *Journal of Parallel and Distributed Computing*, 59(2):204–228, 1999.

[48] P. G. Harrison and N. M. Patel. *Performance Modelling of Communication Networks and Computer Architectures (International Computer S.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1992.

[49] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, pages 100–107, 1968.

[50] R. Hassin and M. Haviv. *To Queue or not to Queue: Equilibrium Behavior in Queueing Systems*. Kluwer Academic Publishers, Boston etc., 2003.

[51] R. Hassin and M. Haviv. Who should be given priority in a queue. *Operations Research Letters*, 34(2):191–198, 2006.

[52] M. Haviv and T. Roughgarden. The price of anarchy in an exponential multi-server. *Operations Research Letters*, 35:421–426, 2007.

[53] M. Haviv and J. van der Wal. Equilibrium strategies for processor sharing and random queues with relative priorities. *Probability in the Engineering and Informational Sciences*, 11(04):403–412, 1997.

[54] Y. Hayel and B. Tuffin. Pricing for heterogeneous services at a discriminatory processor sharing queue. In *Proceedings of Networking*, 2005.

[55] A. Izagirre, U. Ayesta, and I. Verloop. Sojourn time approximations in a multi-class time-sharing server. In *INFOCOM, 2014 Proceedings IEEE*, pages 2786–2794. IEEE, 2014.

[56] F. Kelly. *Stochastic Networks and Reversibility*. Wiley, Chichester, 1979.

[57] J. Kim, J. Kim, and B. Kim. Analysis of the M/G/1 queue with discriminatory random order service policy. *Performance Evaluation*, 68(3):256 – 270, 2011.

[58] L. Kleinrock. Time-shared systems: A theoretical treatment. *Journal of the ACM*, 14(2):242–261, 1967.

[59] R. E. Korf. Optimal path-finding algorithms. In *Search in Artificial Intelligence*, pages 223–267. Springer New York, 1988.

[60] Y. Korilis, A. Lazar, and A. Orda. Architecting noncooperative networks. *IEEE J.Sel. A. Commun.*, 13(7), September 2006.

[61] Y. A. Korilis, A. A. Lazar, and A. Orda. Achieving network optima using stackelberg routing strategies. *IEEE/ACM TRANSACTIONS ON NETWORKING*, 5(1), February.

[62] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th Annual Conference on Theoretical Aspects of Computer Science*, STACS'99, pages 404–413, 1999.

[63] M. Lippi, M. Ernandes, and A. Felner. Efficient single frontier bidirectional search. In *Proceeding of the Forth International Symposium on Combinatorial Search*, 2012.

[64] M. Luby and P. Ragde. A bidirectional shortest-path algorithm with good average-case behavior. *Algorithmica*, 4(1-4):551–567, 1989.

[65] D. Monderer and L. S. Shapley. Potential games. *Games and Econ. Behavior*, 14:124–143, 1996.

[66] D. Mosk-aoyama, T. Roughgarden, and D. Shah. Fully distributed algorithms for convex optimization problems. *SIAM Journal on Optimization*, 2010.

[67] J. D. Murchland. Braess's paradox of traffic flow. *Transportation Research*, 4(4):391–394, 1970.

[68] N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007.

[69] A. Orda, R. Rom, and N. Shimkin. Competitive routing in multiuser communication networks. *IEEE/ACM Transactions on Networking*, 1(5):510–521, 1993.

[70] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991.

[71] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. *IEEE/ACM Trans. Netw.*, 1(3), June 1993.

[72] A. C. Pigou. The economics of welfare. *London: Macnillam*, 1920.

[73] I. Pohl. *Bi-directional and Heuristics Search in Path Problems*. PhD thesis, Standford University, 1969.

[74] K. Rege and B. Sengupta. Queue length distribution for the discriminatory processor sharing queue. *Operations Research*, 44(4):653–657, 1996.

[75] A. W. Richa, M. Mitzenmacher, and R. Sitaraman. The power of two random choices: A survey of techniques and results. *Combinatorial Optimization*, 9:255–304, 2001.

[76] J. W. Roberts. A survey on statistical bandwidth sharing. *Computer networks*, 45(3):319–332, 2004.

[77] T. Roughgarden. *Selfish routing*. PhD thesis, Cornell University, 2002.

[78] T. Roughgarden. *Selfish routing and the price of anarchy*, volume 174. MIT press Cambridge, 2005.

[79] T. Roughgarden. Selfish routing with atomic players. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1184–1185, 2005.

[80] T. Roughgarden. Intrinsic robustness of the price of anarchy. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 513–522. ACM, 2009.

[81] T. Roughgarden and É. Tardos. How bad is selfish routing? *Journal of the ACM*, 49(2):236–259, 2002.

[82] D. Saucez, I. Cianci, L. A. Grieco, and C. Barakat. When AIMD meets ICN: a bandwidth sharing perspective. In *IFIP Networking*, 2014.

[83] L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968.

[84] D. R. Smith. A new proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 26(1):197–199, 1978.

[85] C. Sommer. Shortest-path queries in static networks. *ACM Comput. Surv.*, 46(4), Mar. 2014.

[86] S. Suri, C. Tóth, and Y. Zhou. Selfish load balancing and atomic congestion games. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, 2004.

[87] C. Szepesvári. Shortest path discovery problems: A framework, algorithms and experimental results. In *AAAI*, pages 550–555, 2004.

[88] I. M. Verloop, U. Ayesta, and R. Núñez-Queija. Heavy-traffic analysis of a multiple-phase network with discriminatory processor sharing. *Operations Research*, 59(3):648–660, 2011.

[89] Z. Wang and J. Crowcroft. Bandwidth-delay based routing algorithms. In *Global Telecommunications Conference, 1995. GLOBECOM '95., IEEE*, volume 3, Nov 1995.

[90] J. Wardrop. Some theoretical aspects of road traffic research. *Proceedings of the Institute of Civil Engineers*, 1:325–378, 1952.

[91] A. Wierman. Fairness and scheduling in single server queues. *Surveys in Operations Research and Management Science*, 16(1):39–48, 2011.

[92] Y. Wu, L. Bui, and R. Johari. Heavy traffic approximation of equilibria in resource sharing games. In *Internet and Network Economics - 7th International Workshop*, pages 351–362, 2011.

[93] Y. Wu, L. Bui, and R. Johari. Heavy traffic approximation of equilibria in resource sharing games. *IEEE Journal on Selected Areas in Communications*, 30(11):2200–2209, 2012.

[94] S. Yashkov. Processor sharing queues: Some progress in analysis. *Queueing Systems*, 2:1–17, 1987.

[95] A. Zwart and O. Boxma. Sojourn time asymptotics in the M/G/1 processor sharing queue. *Queueing Systems*, 35:141–166, 2000.

Résumé Étendu

Cette thèse porte sur l'analyse de l'efficacité des réseaux de télécommunications. Dans les réseaux modernes, les ressources sont partagées entre les utilisateurs et la façon de les partager influence les performance perçues par les utilisateurs. En effet, les utilisateurs peuvent partager les ressources de manière à ce que la performance du système de communication soit optimale, alors qu'un partage inapproprié peut conduire à des dégradations de performances significatives. L'objectif de cette thèse est d'étudier l'efficacité du partage des ressources en comparant la performance obtenue à la performance optimale du système.

Notre travail est à l'intersection des domaines de recherche suivants : la théorie des files d'attente, la théorie des jeux et la théorie des graphes. La théorie des files d'attente et la théorie des jeux sont les principaux outils que nous utilisons dans la première partie de la thèse, dans laquelle nous analysons des jeux non-coopératifs dans les réseaux de files d'attente. Dans ces jeux, les clients du réseau de files d'attente sont supposés égoïstes et l'objectif est d'analyser les performances résultantes de la prise de décision stratégique de ces agents égoïstes. Tout d'abord nous analysons la concurrence entre les utilisateurs pour le partage de la capacité d'un seul serveur. Ensuite, nous nous concentrons sur la performance obtenue lorsque des agents égoïstes partagent les ressources d'un système multi-serveurs.

Dans la deuxième partie de la thèse, nous étudions le problème de la recherche d'un chemin optimal dans un graphe dans lequel la valeur des arêtes est initialement inconnue. Nous utilisons des techniques du domaine de la théorie de graphes afin d'analyser la performance des algorithmes qui permettent de résoudre ce problème.

R.1 Préliminaires

Théorie des files d'attente

Dans cette thèse, nous nous intéressons à l'étude d'une file d'attente avec un seul serveur, qui est un modèle de base de la théorie des files d'attente [46]. Dans le modèle de serveur unique, les tâches arrivent dans le système et attendent, dans la file d'attente, jusqu'à ce qu'elles puissent être exécutées. Une fois qu'une tâche a été traîtée, elle quitte le système.

Quatre éléments caractérisent le comportement d'une file d'attente : le processus d'arrivée des tâches, la distribution des temps de service, la taille du tampon et la politique d'ordonnancement.

- Pour le processus d'arrivée des tâches, nous supposons que celui-ci est un processus de Poisson de paramètre λ, ce qui signifie que les intervales de temps entre deux arrivées consécutives dans la file d'attente sont indépendants et suivent une distribution exponentielle négative de paramètre λ.
- Le temps de service d'une tâche est le temps nécessaire pour traîter celle-ci. Dans nos travaux, les temps de service des tâches peuvent être distribués exponentiellement ou bien suivre une distribution générale, mais nous les supposerons toujours indépendants et identiquement distribués (i.i.d).
- La taille du tampon est définie comme le nombre maximum de tâches qui peuvent être en attente dans la file d'attente.
- La politique d'ordonnancement caractérise la façon dont les tâches en attente sont choisies pour accèder au service. La performance du traitement de la file d'attente est donc influencée par le type de politique d'ordonnancement utilisée.

Un système multi-serveurs est composé d'un ensemble de serveurs. Chaque serveur est modélisé comme une file d'attente et un agent de routage contrôle la façon dont les tâches sont envoyées aux serveurs. Le rôle de cet agent est d'optimiser certains paramètres du système comme par exemple le temps de réponse aux requêtes. Dans les systèmes multi-serveurs que nous considerons les serveurs sont des files d'attente de type Processeur partagé [6] et l'agent de routage utilise une politique de routage de type Bernoulli pour envoyer les tâches aux serveurs [35].

Théorie des jeux non cooperatifs

Un jeu non coopératif [68] est caractérisé par les éléments suivants :

- Un ensemble de joueurs (ou agents),
- Un ensemble d'actions (ou stratégies) que les joueurs peuvent choisir,
- Un coût associé à chaque joueur qui dépend de l'action de tous les agents.

Une hypothèse forte de la théorie des jeux non coopératifs est que les joueurs sont égoïstes et que chacun vise à minimiser son coût.

Un ensemble de stratégies est à équilibre si aucun joueur ne peut diminuer son coût en changeant unilatéralement sa stratégie. Il est important de noter que l'équilibre peut être différent de l'optimum social, qui est la stratégie qui minimise le coût total du

système. Le Prix de l'Anarchie (PoA) est une mesure standard pour évaluer l'inéfficacité de l'équilibre [62]. Il est défini comme le rapport entre le coût du pire équilibre de Nash et celui de l'optimum social.

Théorie des graphes

Un graphe est défini par une paire $G = (V, E)$, où V est l'ensemble des nœuds et E l'ensemble des arêtes. Un graphe est composé de n nœuds si $|V| = n$. Un graphe est complet si chaque paire de nœuds est connectée par une arête. Un graphe non orienté est composé d'arêtes dont le sens n'est pas précisé. En conséquence, un arc de u à v est équivalent à un arc de v à u. Dans cette thèse, nous nous concentrons sur les graphes compléts et non orientés.

Un chemin entre deux nœuds u et v est denoté par $P_{(u,v)}$ et l'ensemble de tous les chemins possibles de u à v par $\mathcal{P}_{(u,v)}$. Pour tout ensemble d'arêtes $H \subseteq E$, nous définissons la valeur de H comme $F(H)$, où $F : 2^E \to [0, \infty)$. Un chemin optimal est un chemin $P^*_{(u,v)}$ s'il optimise (maximise ou minimise) la valeur $F(P^*_{(u,v)})$. Si la fonction F est additive et que l'objectif est de minimiser la valeur associée par F à un chemin, le chemin optimal $P^*_{(u,v)}$ correspond à un plus court chemin entre u et v.

L'algorithme A^* [49] et l'algorithme de Dijkstra [30] sont deux algorithmes qui permettent de trouver le plus court chemin entre deux nœuds donnés. Dans cette thèse, nous analysons des graphes dans lesquels les valeurs des arêtes sont initialement inconnues, mais peuvent être connues en questionnant un oracle. Nous cherchons à découvrir un chemin optimal en minimisant le nombre de questions posées à l'oracle. Nous comparons la performance des algorithmes qui permettent d'obtenir le chemin optimal dans ce type de graphes avec la performance des algorithmes classiques cités ci-dessus.

R.2 Serveur Unique avec Priorités Relatives

Dans les services d'hébergement de fichiers, la bande passante disponible du fournisseur est partagée entre les utilisateurs qui téléchargent simultanément des fichiers. Cette situation peut être modélisé comme un jeu de partage de ressources, au sein duquel les utilisateurs sont en concurrence pour partager la capacité de traîtement du serveur.

Nous considérons un jeu dans lequel R joueurs partagent le capacité d'un serveur. Soit $\mathcal{C} = \{1, \ldots, R\}$ l'ensemble des joueurs. La variable aléatoire représentant le temps de service des tâches du joueur i est B_i. Nous notons $\mathbb{E}(B_i^m)$ le m-ème moment de cette variable aléatoire. Pour clarifier la notation, nous écrivons $\mathbb{E}(B_i^1) = \mathbb{E}(B_i)$ et, dans le cas d'une durée de service exponentiellement distribuée, $\mathbb{E}(B_i) = 1/\mu_i$. Le taux d'arrivée des tâches du joueur i est λ_i et la charge du joueur i est $\rho_i = \lambda_i \mathbb{E}(B_i)$. La charge totale du système est $\rho = \sum_{i \in \mathcal{C}} \rho_i$.

Nous supposons que chaque utilisateur peut choisir le prix à payer par unité de

charge g_i. Ce prix est proportionnel à sa priorité dans la file d'attente. Autrement dit, un paiement plus élevé conduit à une vitesse de service plus élevée. L'objectif du joueur i est de minimiser son paiement tout en s'assurant que le temps moyen de réponse de ses travaux ne dépasse pas un délai donné, i.e., en s'assurant que $\overline{T}_i(\mathbf{g};\rho) \leq c_i$.

$$\begin{aligned} \min_{g_i \geq \epsilon} \quad & \rho_i g_i \\ \text{sous la contrainte} \quad & \overline{T}_i(\mathbf{g};\rho) \leq c_i. \end{aligned}$$

La valeur ϵ est le prix minimal qu'un utilisateur doit payer pour pouvoir être servi. Le vecteur $\mathbf{g}^{NE}$ est à l'équilibre de Nash si

$$g_i^{NE} = \operatorname{argmin} \left\{ g_i \geq \epsilon \ : \ \overline{T}_i(g_i, \mathbf{g}_{-i}^{NE};\rho) \leq c_i \right\},$$

pour tout $i \in \mathcal{C}$, où $\mathbf{g}_{-i}^{NE} = \left(g_1^{NE}, \ldots, g_{i-1}^{NE}, g_{i+1}^{NE}, \ldots, g_R^{NE}\right)$.

Nous supposons que la capacité du serveur est partagée entre les utilisateurs en utilisant le modèle *Discriminatory Processor Sharing* (DPS) [8] , qui est une généralisation multi-classe de l'ordonnancement *Processor Sharing* (PS) [58]. Dans le modèle DPS, toutes les tâches du système sont servies simultanément, mais la vitesse de service de chaque utilisateur dépend de sa priorité relative. Ainsi, s'il y a N_i tâches de classe i à un instant donné, une tâche de classe i est traîtée au taux

$$r_i(N_1, \ldots, N_R) = \frac{g_i}{\sum_{j=1}^{R} g_j N_j}.$$

Le système DPS est très compliqué à analyser et le jeu que nous avons formulé ne peut être résolu que pour des cas particuliers. En conséquence, nous proposons d'étudier une approximation du jeu originel. En utilisant les résultat de [88], nous définissons le jeu en régime de fort trafic de la façon suivante :

$$\begin{aligned} \min_{g_i \geq \epsilon} \quad & \rho_i g_i \\ \text{subject to} \quad & \overline{T}_i(\mathbf{g};1) \leq \tilde{c}_i. \end{aligned}$$

où $\tilde{c}_i = (1-\rho)c_i$ et $\overline{T}_i(\mathbf{g};1) = \frac{\mathbb{E}(B_i)}{g_i} \frac{\sum_k \lambda_k \mathbb{E}\left(B_k^2\right)}{\sum_k \lambda_k \mathbb{E}\left(B_k^2\right) \frac{1}{g_k}}$. Nous notons que ce problème est valide pour le cas général et qu'il peut être utilisé comme approximation du jeu originel car $\overline{T}_i(\mathbf{g};\rho) \approx \frac{\overline{T}_i(\mathbf{g};1)}{1-\rho}$ en régime du trafic fort.

Définition R.2.1. *Le vecteur des temps de réponses* $\mathbf{t}$ *est réalisable s'il existe un vecteur* $\mathbf{g} > 0$ *tel que* $t_i = \overline{T}_i(\mathbf{g};\rho)$*, pour tout* $i \in \mathcal{C}$*.*

Nous dénotons par $\mathcal{T} = \{\mathbf{t} : \mathbf{t} \text{ est réalisable}\}$ l'ensemble des vecteurs réalisables.

Définition R.2.2. *Un vecteur* $\mathbf{c} \in \mathbb{R}_+^R$ *est faisable si et seulement si* $\exists \mathbf{t} \in \mathcal{T}$ *tel que* $\mathbf{t} \preceq \mathbf{c}$*, où* $\preceq$ *est l'ordre composant par composant.*

Dans la suite, nous allons dire que le jeu est faisable si le vecteur $\mathbf{c}$ est faisable.

Définition R.2.3. *Un joueur de classe i est consideré* équitable *si le temps de réponse qu'il obtient avec des poids égaux, $\mathbb{E}(B_i)/(1-\rho)$, est plus petit que c_i, i.e., si*

$$\mathbb{E}(B_i)/c_i \leq (1-\rho).$$

R.2.1 Travaux Connexes

Le temps de réponse de la file d'attente DPS pour le cas général n'a pas de forme analytique simple. L'expression explicite de la réponse moyenne a été obtenue seulement pour deux classes et pour les temps de service exponentiellement distribués. De plus, le temps de réponse moyen de travaux est la solution d'un système d'équations si les temps de service sont exponentiellement distribués [34]. Pour des temps des services généraux, la dérivé du temps moyen de réponse conditionnel satisfait un système d'équations intégro-différentielles. C'est pour cela qu'il n'y a pas beaucoup de résultats sur les comportements stratégiques dans les systèmes à temps partagé. Dans [54] les auteurs considèrent deux types d'applications dans une file d'attente DPS qui sont en concurrence pour être servies et analysent comment le prix optimal peut être trouvé. Dans [93], un travail de recherche plus récent, les auteurs définissent un jeu pour la file d'attente DPS dans laquelle chaque utilisateur cherche à minimiser la somme du coût de traitement prévu et de son paiement. Étant donnée la difficulté à analyser un tel modèle, les auteurs proposent une approximation en régime de fort trafic, c'est à dire lorsque la charge du système est proche de 1. Même si nous utilisons nous aussi le modèle DPS pour le partage de la capacité, le problème que nous considérons est différent, puisque, dans notre formulation, chaque utilisateur vise à minimiser son paiement en s'assurant que ses tâches soient servis avant un certain temps.

R.2.2 Solution du Jeu

Tout d'abord, nous démontrons certaines propietés que le jeu satisfait quand il est faisable.

Proposition R.2.1. *Avec des temps de service généraux et un nombre de joueurs arbitraire, si le jeu est faisable, nous savons que :*

- *Il existe un equilibre de Nash,*
- *Les dynamiques de l'algorithme du meilleur temps de réponse converge vers un équilibre de Nash si le point initial $\mathbf{g}$ verifie que $\overline{T}_i(\mathbf{g}, \rho) \leq c_i$ pour tout $i \in \mathcal{C}$.*

Comme expliqué précédemment, le jeu que nous avons formulé ne peut pas être résolu pour le cas général. Maintenant, nous présentons les conditions d'existence de l'equilibre de Nash.

Théorème R.2.1. *Soit $\boldsymbol{\rho}_r = \sum_{i\in r} \rho_i$, où $r \subseteq \mathcal{R}$. Pour des temps de service exponentiels, le jeu est faisable si et seulement si*

$$\sum_{i\in r} \rho_i c_i \geq W_r, \ \forall r \in \mathcal{R},$$

où $W_r = \frac{1}{1-\boldsymbol{\rho}_r} \sum_{i\in r} \frac{\rho_i}{\mu_i}$.

Le résultat suivant caractérise l'efficacité de l'équilibre.

Proposition R.2.2. *Pour les temps de service généraux, nous savons que :*

- *Pour un nombre de classes arbitraire, si* $\mathbf{c} \in \mathcal{T}$, $PoA = \infty$.
- *Pour le jeu avec deux joueurs, tel que* $\mathbf{c} \notin \mathcal{T}$ *et l'equilibre de Nash existe,* $PoA = 1$.

Nous caractérisons l'equilibre de Nash pour le jeu avec deux joueurs et des temps de service exponentiellement distribués.

Proposition R.2.3. *Pour le jeu de deux joueurs avec des temps de service exponentiels et* $c_1\mu_1 \leq c_2\mu_2$, *si le jeu est faisable et* $\mathbf{c} \notin \mathcal{T}$, *l'unique equilibre est*

- $\mathbf{g^{NE}} = (\epsilon, \epsilon)$ *si le joueur* 1 *est juste et*
- *sinon,* $\mathbf{g^{NE}} = (g_1^{NE}, \epsilon)$, *où*

$$g_1^{NE} = \epsilon \ \frac{-\mu_1\rho_2 + \mu_2(1-\rho_2)\left[\mu_1 c_1(1-\rho) - 1\right]}{-\mu_1\rho_2 - \mu_1(1-\rho_1)\left[\mu_1 c_1(1-\rho) - 1\right]}.$$

R.2.3 Le Jeu en Régime de Fort Trafic

Nous avons défini un jeu qui est valide dans un régime de fort trafic. Nous sommes capables de résoudre ce jeu pour le cas général. D'abord, nous montrons les conditions de faisabilité.

Proposition R.2.4. *Le jeu en fort trafic est faisable si et seulement si*

$$\sum_i \lambda_i \mathbb{E}\left(B_i^2\right)\left(\frac{\tilde{c}_i}{\mathbb{E}(B_i)} - 1\right) \geq 0.$$

Nous démontrons aussi que l'équilibre de Nash pour le jeu en régime de fort trafic est unique et nous le caractérisons.

Théorème R.2.2. *Si le jeu en régime de fort trafic est faisable et* $\mathbf{c} \notin \mathcal{T}^{HT}$, *l'unique équilibre de Nash est*

$$\begin{aligned} g_i^{NE} &= \epsilon \frac{\tilde{t_m}/\mathbb{E}(B_m)}{\tilde{c}_i/\mathbb{E}(B_i)}, \text{ for all } i < m, \\ g_i^{NE} &= \epsilon, \text{ for all } i \geq m, \end{aligned}$$

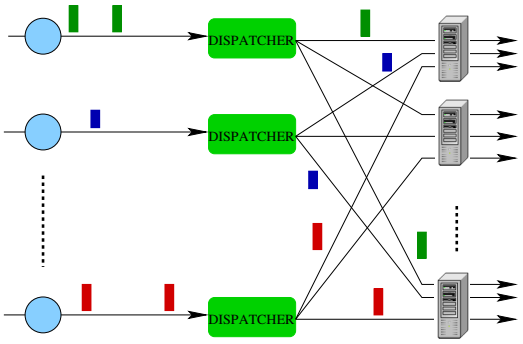

Figure R.1: Architecture centralisée

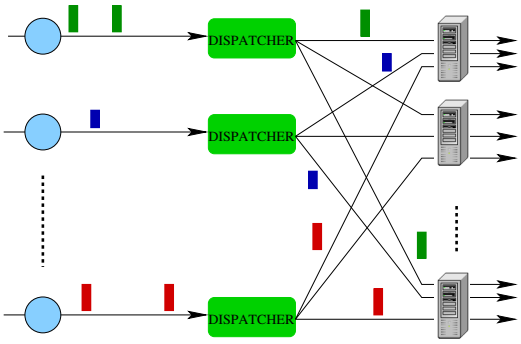

Figure R.2: Architecture decentralisée

où $m \in \mathcal{C}$ *est la valeur minimale telle qu'il existe* $\tilde{t_m} \leq \tilde{c}_m$ *qui vérifie*

$$\frac{\tilde{t_m}}{\mathbb{E}(B_m)} = \frac{\sum_{k=1}^{R} \lambda_k \mathbb{E}\left(B_k^2\right) - \sum_{k=1}^{m-1} \lambda_k \frac{\mathbb{E}\left(B_k^2\right)}{\mathbb{E}(B_k)} \tilde{c}_k}{\sum_{k=m}^{R} \lambda_k \mathbb{E}\left(B_k^2\right)}.$$

R.3 Equilibrage de Charge Non Coopératif

Nous nous concentrons sur le problème de l'équilibrage de charge d'une ferme de serveurs. Nous considerons une architecture centralisée au sein de laquelle il y a un agent de routage responsable de l'équilibrage de charge, c'est-à-dire qu'il reçoit toutes les tâches entrantes et les envoie aux serveurs, avec pour objectif de minimiser le temps de réponse moyen des tâches. Nous analysons égalementi une architecture decentralisée où plusieurs agents égoïstes font la répartition de charge. Dans l'architecture decentralisée, nous disons que les agents sont égoïstes car ils visent à minimiser le temps de réponse moyen de leurs propres tâches. Ainsi, nous pouvons définir un jeu non-cooperatif entre les agents de routage de l'architecture decentralisée.

Nous voulons comparer la performance de l'architecture decentralisée à l'équilibre avec K agents de routage et la performance de l'architecture centralisée pour évaluer l'efficacité de l'équilibrage de charge non-cooperatif. Soit $D_K(\boldsymbol{\lambda}, \mathbf{r})$ la performance de l'architecture decentralisée à l'équilibre avec K agents de routage et $D_1(\bar{\lambda}, \mathbf{r})$ celle de l'architecture centralisée. Nous définisons l'inefficacité de la façon suivante :

$$\textit{inefficacité } I_K^S(\mathbf{r}) = \sup_{\boldsymbol{\lambda} \in \Lambda(\bar{\lambda}), \bar{\lambda} < \overline{R}} \frac{D_K(\boldsymbol{\lambda}, \mathbf{r})}{D_1(\bar{\lambda}, \mathbf{r})}.$$

où $\overline{R}$ est la somme des capacités des serveurs, $\bar{\lambda}$ le trafic entrant total et $\Lambda(\bar{\lambda})$ est l'ensemble des vecteurs positifs $(\lambda_1, \lambda_2, \ldots, \lambda_K)$ tels que $\sum_i \lambda_i = \bar{\lambda}$. Dans ce contexte, λ_i représente l'intensité du trafic contrôlé par l'agent de routage i. Le Prix de l'Anarchie pour ce système correspond alors à la configuration des capacités des serveurs ayant

l'efficacité la moins bonne, autrement dit

$$PoA(K,S) = \sup_{\mathbf{r}} I_K^S(\mathbf{r}).$$

R.3.1 Travaux Connexes

L'efficacité de l'équilibrage de charge non-cooperatif a été étudiée auparavant avec le Prix de l'Anarchie. Nous résumons maintenant les bornes superieures sur le PoA dans les jeux d'équilibrage de charge non-cooperatif disponibles dans la littérature. Dans [52], les auteurs montrent que pour le cas non atomique, i.e., quand il y a un nombre infini de joueurs, le PoA est inférieur au nombre de serveurs. Au contraire, les auteurs de [14] analysent le cas atomique et prouvent que le PoA est inférieur à la racine carrée du nombre d'agents de routage. Selon ces résultats, la valeur du PoA est très grande, donc l'équilibre de Nash peut être extrêmement inefficace. Nous prouvons ci-dessous que l'équilibre de Nash est efficace dans la plupart des cas.

R.3.2 Pire Cas des Condition de Trafic

Tout d'abord, nous observons que, parmi tous les vecteurs de trafic avec une intensité totale $\bar{\lambda}$, le maximum de $D_K(\boldsymbol{\lambda}, \mathbf{r})$ est obtenu quand tous les agents de routage contrôlent la même quantité de trafic $\bar{\lambda}/K$. Ainsi, pour analyser l'inefficacité nous avons seulement besoin de regarder le trafic entrant total, i.e.

$$\textit{inefficacité } I_K^S(\mathbf{r}) = \sup_{\bar{\lambda}<\overline{R}} \frac{D_K(\frac{\bar{\lambda}}{K}\mathbf{e}, \mathbf{r})}{D_1(\bar{\lambda}, \mathbf{r})},$$

où $\mathbf{e} = (1, \ldots, 1)$.

Nous montrons également que la performance des deux architectures est la même lorsque la charge du système est proche de 1.

Théorème R.3.1. *Pour $K < \infty$ fixé,*

$$\lim_{\bar{\lambda}\to\overline{R}} \frac{D_K(\frac{\bar{\lambda}}{K}\,\mathbf{e}, \mathbf{r})}{D_1(\bar{\lambda}, \mathbf{r})} = 1.$$

R.3.3 Inefficacité pour Deux Classes de Serveurs

Nous nous concentrons sur l'inefficacité d'une ferme de serveurs avec S serveurs, en supposant que S_1 sont des serveurs rapides de capacité r_1 et que $S_2 = S - S_1$ sont des serveurs lents de capacité r_2, avec $r_1 > r_2$. Nous conjecturons que, pour un nombre de serveurs donné et pour chaque instance du problème avec des capacités de serveur arbitraires, nous pouvons construire un scénario avec le même nombre de serveurs rapides et lents dont l'inefficacité est pire.

Conjecture R.3.1. *Pour une ferme de serveurs de S serveurs avec $r_1 > r_S$*

$$I_K^S(\mathbf{r}) \geq I_K^S(\mathbf{r}^*),$$

où $\mathbf{r} = (\overbrace{r_1, \ldots, r_1}^{m}, \overbrace{r_S, \ldots, r_S}^{S\text{-}m})$ *et* $\mathbf{r}^* = (\overbrace{r_1, \ldots, r_1}^{m}, \overbrace{r_{m+1}, \ldots, r_{S-1}, r_S}^{S\text{-}m})$, *avec* $r_1 > r_{m+1} \geq \cdots \geq r_{S-1} \geq r_S$ *et* $m \geq 1$.

Nous définisons $\bar{\lambda}^{NE} = S_1 r_1 \left(1 - \frac{2}{\sqrt{(K-1)^2 + 4K\frac{r_1}{r_2}} - (K-1)}\right)$.

Nous prouvons que l'inefficacité est atteinte quand $\bar{\lambda} = \bar{\lambda}^{NE}$, autrement dit, quand l'architecture décentralisée commence à utiliser les serveurs lents.

Théorème R.3.2. *Pour une ferme de serveurs de deux classes de serveurs,*

$$I_K^S(\mathbf{r}) = \frac{D_K(\frac{\bar{\lambda}^{NE}}{K}\mathbf{e}, \mathbf{r})}{D_1(\bar{\lambda}^{NE}, \mathbf{r})}.$$

En plus, nous montrons que l'inefficacité depend uniquement des ratios r_1/r_2 et S_1/S_2.

Proposition R.3.1. *Soit* $\beta = \frac{r_1}{r_2} > 1$ *et* $\alpha = \frac{S_1}{S_2} > 0$, *nous avons que*

$$I_K^S(\mathbf{r}) = \frac{1}{2} \frac{\sqrt{(K-1)^2 + 4K\beta} - (K+1)}{\frac{(\frac{1}{\alpha} + \sqrt{\beta})^2}{\frac{1}{\alpha} + \frac{2\beta}{\sqrt{(K-1)^2 + 4K\beta} - (K-1)}}} - (\frac{1}{\alpha} + 1).$$

Nous montrons également que le PoA d'une ferme de serveurs avec S serveurs et K agents de routage est obtenu quand il y a un seul serveur rapide qui est infiniment plus rapide que les serveurs lents, autrement dit

$$PoA(K, S) = \lim_{\beta \to \infty} I_K(\frac{1}{S-1}, \beta).$$

Nous pouvons observer sur les Figures R.3 and R.4 que le PoA est atteint pour les conditions que nous avons définies précédemment et que la valeur de l'inefficacité est proche de 1 pour le reste des cas. Selon ce résultat, nous pouvons conclure que l'équilibre du jeu de répartition de charge est efficace pour la plupart des cas.

R.3.4 Inefficacité pour un Nombre Infini d'Agents de Routage

Nous comparons la performance des deux architectures lorsque le nombre d'agents de routage tend vers l'infini. Nous montrons d'abord que, si la charge est proche de 1, la performance des deux systèmes ne coïncide pas.

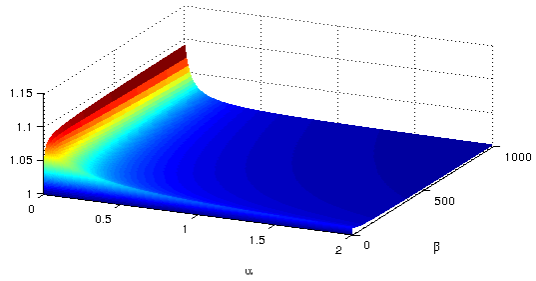

Figure R.3: Evolution de l'inefficacité en fonction de α et β. $K = 2$ et $S = 1000$.

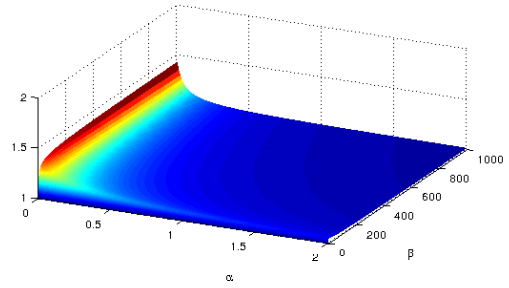

Figure R.4: Evolution de l'inefficacité en fonction de α et β. $K = 5$ et $S = 1000$.

Proposition R.3.2. *Pour une ferme de serveurs avec S serveurs,*

$$\lim_{\bar{\lambda}\to\overline{R}} \lim_{K\to\infty} \frac{D_K(\frac{\bar{\lambda}}{K}\,\mathbf{e}, \mathbf{r})}{D_1(\bar{\lambda}, \mathbf{r})} = \frac{S\ \overline{R}}{\left(\sum_{i=1}^{S} \sqrt{r_i}\right)^2}.$$

Par rapport a l'inefficacité, nous nous concentrons sur les fermes de serveurs avec deux types de serveurs. Nous montrons que, pour ce cas, l'inefficacité est atteinte quand l'architecture decentralisé commence à utiliser les serveurs lents. La valeur de l'inefficacité pour ce cas depend uniquement des ratios r_1/r_2 et S_1/S_2.

Corollary R.3.1. *Soit $\alpha = \frac{r_1}{r_2}$ et $\beta = \frac{S_1}{S_2}$. Nous avons que*

$$I_\infty(\alpha, \beta) = \frac{(\beta - 1)(1 + \frac{1}{\alpha})}{(\sqrt{\beta} + \frac{1}{\alpha})^2 - (\frac{1}{\alpha} + 1)^2}.$$

Nous obtenons aussi l'expression du PoA pour une ferme de serveurs avec deux types de serveurs et un nombre infini d'agents de routage.

Proposition R.3.3.

$$PoA(\infty, S) = S.$$

Nous observons que cette expression coïncide avec celle donnée dans [52].

R.4 Algorithmes de Découverte de Chemin

Nous considérons un graphe complet non orienté $G = (V, E)$, où V est l'ensemble des nœuds et E est l'ensemble des arêtes, et le nombre de nœuds est n. La valeur des arêtes est inconnue au début, mais peut être demandée. La performance d'un chemin $P_{(s,t)}$ est donnée par sa valeur $F(P_{(s,t)})$. Ainsi, pour une fonction donnée F, l'objectif est de découvrir un chemin P^* entre s et t qui optimise la performance. Le problème réside

dans le fait que les valeurs des arêtes ne sont pas connues. N'importe quel algorithme qui vise à résoudre ce problème a besoin de découvrir la valeur de certaines arêtes pour trouver le chemin donnant la performance optimale. L'objectif est alors de trouver un chemin de performance optimale en découvrant le minimum d'arêtes. Ce problème est connu comme le problem de Découverte de Chemin Optimal (OPD). Pour autant que nous savons, ce problème a été étudié dans la littérature que pour des cas particuliers. Un exemple est étudié dans [87], dans lequel les auteurs introduisent le problème de découverte de plus court chemin et cherchent à trouver le chemin de moindre coût avec le nombre minimum d'arêtes demandées.

Une instance du problème OPD I est défini par le graphe complet G, les valeurs des arêtes et un facteur d'approximation requis α sur la performance de la solution par rapport au chemin de performance optimale. Nous disons qu'un algorithme résout le problème OPD pour l'instance I si la solution que l'algorithme donne satisfait ce niveau de performance. Pour le cas $\alpha = 1$, un algoritme trouve un chemin optimal.

Nous définisons le problème OPD pour le cas de minimisation et de maximisation et pour une fonction générale. Par contre, pour analyser ce problème nous avons besoin de certaines hypothèses sur la function F:

- si $H \subset E$, alors
$$F(H) = 0 \iff H = \{\emptyset\},$$
- si $H, H', H'' \subseteq E$ tel que $H \cap H'' = \emptyset$ et $H \cap H' = \emptyset$, alors
$$\text{if } F(H') \leq F(H'') \Rightarrow F(H \cup H') \leq F(H \cup H''),$$
- si $H, H' \subseteq E$, où H' est un sous-ensemble particulier H, alors
$$F(H') < F(H), \text{ pour le cas de minimisation et}$$
$$F(H') > F(H), \text{ pour le cas de maximisation.}$$

De plus, nous supposons que les valeurs des arêtes sont entre deux valeurs positives a et b. Dans la suite, nous nous concentrons sur le problème de minimisation. Cependant, les preuves des résultats obtenus s'étendent sans difficulté au cas où l'objectif est de maximiser la valeur du chemin.

Nombre de Requêtes comme Mesure de Performance

Soit $U(\mathcal{A}(I))$ l'ensemble des arêtes découvertes par l'algorithme $\mathcal{A}$ quand il résout le problème OPD pour une instance I. Pour estimer la performance des algorithmes qui résolvent le problème OPD, une possibilité est d'utiliser le nombre de requêtes.

Définition R.4.1. *Le nombre de requêtes d'un algorithme qui résout le problème OPD est β_n si et seulement si*

$$|U(\mathcal{A}(I))| \leq \beta_n \, \forall \, I \text{ such that } |I| = n.$$

Nous montrons que le nombre de requêtes β_n de tout algorithme est, au moins, $n^2/4$, ce qui est de l'ordre du nombre d'arêtes dans un graphe complet. Ainsi, selon ce résultat, nous pouvons affirmer que le nombre de requêtes n'est pas une bonne mesure de la performance des algorithmes qui résolvent le problème OPD.

Soit $\mathcal{C}^{\alpha}_{\min}(I)$ l'ensemble des arêtes de taille minimale qui peut résoudre le problème OPD pour une instance I. Le ratio des requêtes d'un algorithme $\mathcal{A}$ est défini de la façon suivante.

Définition R.4.2. *Le ratio des requêtes d'un algorithme $\mathcal{A}$ qui résout le problème OPD est defni comme suit:*

$$\max_I \frac{|U(\mathcal{A}(I))|}{|\mathcal{C}^{\alpha}_{\min}(I)|}.$$

Analyse du Ratio des Requêtes

Nous analysons le ratio des requêtes des algorithmes qui résolvent le problème OPD. Tout d'abord, nous montrons une borne inférieure pour n'importe quel algorithme qui résout le problème OPD.

Théorème R.4.1. *Tout algorithme $\mathcal{A}$ qui résout le problème OPD satisfait*

$$\max_I \frac{|U(\mathcal{A}(I))|}{|\mathcal{C}^{\alpha}_{\min}(I)|} \geq 1 + \frac{4}{n} - \frac{8}{n^2}.$$

Nous disons qu'un algorithme résout le problème OPD de façon optimale s'il a un ratio des requêtes égal à 1. Selon cette définition et le résultat prsenté ci-dessus, nous pouvons conclure qu'il n'y a pas d'algorithme qui résout le problème OPD en posant systématiquement le nombre minimal de questions.

Nous présentons un algorithme et nous étudions la valeur du ratio des requêtes. Cet algorithme est décrit dans l'Algorithme 1. Nous prouvons que le ratio de requêtes de cet algorithme est plus petit que 2.

Théorème R.4.2. *Soit $\mathcal{A}^{OPD}$ l'algorithme décrit dans Algorithme 1. Donc, quand $\mathcal{A}^{OPD}$ trouve un chemin optimal, nous avons*

$$\max_I \frac{|U(\mathcal{A}^{OPD}(I))|}{|\mathcal{C}^{1}_{\min}(I)|} \leq 2 - \frac{1}{n-1}.$$

Algorithme 1 Algorithme pour le problème OPD

INITIALISER ensembles $m_s = \{\emptyset\}$, $m_t = \{\emptyset\}$;
chemin $PATH_{prop} = \emptyset$;
chemins $PATH_{(s,u)} = \emptyset$, et $PATH_{(u,t)} = \emptyset \, \forall u \in V/\{s,t\}$;
variable $approx = \infty$;
variables $s^* = s$ et $t^* = t$;
tant que $approx > \alpha$ **faire**
  ACTUALISER $m_s := m_s \cup s^*$.
  ACTUALISER $m_t := m_t \cup t^*$.
  DEMANDER $\{(s^*, t^*)\}$.
  DEMANDER $\{(s^*, u) : \forall u \in V/\{m_s \cup m_t\}\}$.
  DEMANDER $\{(u, t^*) : \forall v \in V/\{m_s \cup m_t\}\}$.
  CALCULER le chemin optimal entre s et t qui contient uniquement des arêtes connues.
  ACTUALISER $PATH_{prop}$ au chemin optimal entre s et t qui contient uniquement des arêtes connues.
  CALCULER le chemin entre s et u optimal qui contient uniquement des arêtes connus $\forall \, u \in V/\{m_s \cup m_t\}$.
  ACTUALISER $PATH_{(s,u)}$ au chemin optimal entre s et u qui contient uniquement des arêtes connues $\forall \, u \in V/\{m_s \cup m_t\}$.
  CALCULER le chemin optimal entre u et t qui contient uniquement des arêtes connus $\forall \, u \in V/\{m_s \cup m_t\}$.
  ACTUALISER $PATH_{(t,u)}$ le chemin optimal entre u et t qui contient uniquement des arêtes connues $\forall \, u \in V/\{m_s \cup m_t\}$.
  CALCULER $s^* := \text{argmin}_{u \in V/\{m_s \cup m_t\}} F(PATH_{(s,u)})$.
  CALCULER $t^* := \text{argmin}_{u \in V/\{m_s \cup m_t\}} F(PATH_{(u,t)})$.
  ACTUALISER $approx := \frac{F(PATH_{prop})}{F(PATH_{(s,s^*)} \cup PATH_{(t^*,t)})}$.
fin tant que
RETURNER $P_{(s,t)} := PATH_{prop}$.

Comparaison avec d'autres Algorithmes

Soit $\mathcal{A}^{Gre}_{hom}$ une modification de l'algorithme glouton présenté dans [87] pour résoudre le problème OPD quand les estimations sont homogènes, i.e. les estimateurs initiaux sont tous égaux à a. Nous étudions le ratio des requêtes de $\mathcal{A}^{Gre}_{hom}$ quand il trouve la solution optimale et nous observons qu'il coïncide avec celui de l'algorithme $\mathcal{A}^{OPD}$.

Theorem R.4.1. *Pour l'algorithme $\mathcal{A}^{Gre}_{hom}$, nous avons*

$$\max_I \frac{|U(\mathcal{A}^{Gre}_{hom}(I))|}{|\mathcal{C}^1_{\min}(I)|} \leq 2 - \frac{1}{n-1}.$$

Nous analysons maintenant la performance des algorithmes de recherche tel que l'algorithme A^*. Nous montrons que le ratio des requêtes est de l'ordre de n pour ces

algorithmes. Nous observons que cette valeur est beaucoup plus grande que la borne supérieure du ratio des requêtes de l'algoritme $\mathcal{A}^{OPD}$.

List of Publications

- **The Optimal Path Discovery Problem with Homogeneous Knowledge**
  O. Brun, J. Doncel, C. Thraves Caro.
  Submitted for publication 2015.

- **A Resource Sharing Game with Relative Priorities**
  J. Doncel, U. Ayesta, O. Brun, B. J. Prabhu.
  Performance Evaluation, Volume 79, September 2014.

- **Predicting Response Times of Applications in Virtualized Environments**
  H. Ben Cheikh, J. Doncel, O. Brun, B. Prabhu.
  Proceedings of IEEE NCCA 2014.

- **Is the Price of Anarchy the right measure for Load Balancing Games?**
  J. Doncel, U. Ayesta, O. Brun, B. J. Prabhu.
  ACM Transactions on Internet Technology (ToIT) 4, 2-3, Article 18 (October 2014).

- **Congestion Control of TCP Flows in Internet Routers by Means of Index Policy**
  K. Avratchenkov, U. Ayesta, J. Doncel, P. Jacko.
  Computer Networks Volume 57 Issue 17 (2013).

- **On the Efficiency of Non-Cooperative Load Balancing**
  J. Doncel, U. Ayesta, O. Brun, B. J. Prabhu.
  Proceedings of IFIP/TC6 Networking 2013.

- **Optimal Congestion Control of TCP Flows for Internet Routers**
  K. Avratchenkov, U. Ayesta, J. Doncel, P. Jacko.
  Performance Evaluation Review, Volume 40 Issue 3, December 2012.

About the Author

Josu Doncel obtained an Industrial Engineering degree in 2007 and a B.Sc. in Mathematics in 2010, both from the University of the Basque Country. In 2011, he received his Master's degree in mathematical modelling, statistics and computation from the University of the Basque Country. His master thesis entitled "Development and Testing of Index Policies in Internet Routers" has been carried out during an intership at BCAM. Josu Doncel defends his Ph.D. thesis at LAAS-CNRS on March 2015.

His research is devoted to the performance analysis of communication networks. In particular, he uses queueing theory, game theory, graph theory and optimization techniques to study the efficiency of these systems.