Orchestration and Scheduling of Resources in Softwarized Networks

Hyame Alameddine

A Thesis
In
The Concordia Institute
For
Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy (Information and Systems Engineering) at
Concordia University
Montréal, Québec, Canada

February 2019

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By:	Hyame Alameddine
Entitled:	Orchestration and Scheduling of Resources in Softwarized Networks

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Information and Systems Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

______________________________ Chair
Dr. Abdel R. Sebak

______________________________ External Examiner
Dr. Jelena Mišić

______________________________ External to Program
Dr. Ivan Contreras

______________________________ Examiner
Dr. Jia Yuan Yu

______________________________ Examiner
Dr. Roch Glitho

______________________________ Thesis Supervisor
Dr. Chadi Assi

Approved ______________________________
Dr. Chadi Assi, Graduate Program Director

March 7, 2019 ______________________________
Dr. Amir Asif, Dean
Gina Cody School of Engineering and Computer Science

ABSTRACT

Orchestration and Scheduling of Resources in Softwarized Networks

Hyame Alameddine, Ph.D.
Concordia University, 2019

The Fifth Generation (5G) era is touted as the next generation of mobile networks that will unleash new services and network capabilities, opening up a whole new line of businesses recognized by a top-notch Quality of Service (QoS) and Quality of Experience (QoE) empowered by many recent advancements in network softwarization and providing an innovative on-demand service provisioning on a shared underlying network infrastructure. 5G networks will support the immerse explosion of the Internet of Things (IoT) incurring an expected growth of billions of connected IoT devices by 2020, providing a wide range of services spanning from low-cost sensor-based metering services to low-latency communication services touching health, education and automotive sectors among others. Mobile operators are striving to find a cost-effective network solution that will enable them to continuously and automatically upgrade their networks based on their ever growing customers demands in the quest of fulfilling the new rising opportunities of offering novel services empowered by the many emerging IoT devices. Thus, departing from the shortfalls of legacy hardware (i.e., high cost, difficult management and update, etc.) and learning from the different advantages of virtualization technologies which enabled the sharing of computing resources in a cloud environment, mobile operators started to leverage the idea of network softwarization through several emerging technologies. Network Function Virtualization (NFV) promises an ultimate Capital Expenditures (CAPEX) reduction and high flexibility in resource provisioning and service delivery through replacing hardware equipment by software. Software Defined Network (SDN) offers network and mobile operators programmable traffic management and delivery. These technologies will enable the launch of Multi-Access Edge Computing (MEC) paradigm that promises to complete the 5G networks requirements in providing low-latency services by bringing the computing resources to the edge of the network, in close vicinity of the users, hence, assisting the limited capabilities of their IoT devices in delivering their needed services. By leveraging network softwarization, these technologies will initiate a tremendous re-design of current networks that will be transformed to self-managed, software-based networks exploiting multiple benefits ranging from flexibility, programmability, automation, elasticity among others.

This dissertation attempts to elaborate and address key challenges related to enabling the re-design of current networks to support a smooth integration of the NFV and MEC technologies. This thesis provides a profound understanding and novel contributions in resource and service provisioning and scheduling towards enabling efficient resource and network utilization of the underlying infrastructure by leveraging several optimization and game theoretic techniques. In particular, we first, investigate the interplay existing between network function mapping, traffic routing and Network Service (NS) scheduling in NFV-based networks and present a Column Generation (CG) decomposition method to solve the problem with considerable runtime improvement over mathematical-based formulations. Given the increasing interest in providing low-latency services and the correlation existing between this objective and the goal of network operators in maximizing their network admissibility through efficiently utilizing their network resources, we revisit the latter problem and tackle it under different assumptions and objectives. Given its complexity, we present a novel game theoretic approach that is able to provide a bounded solution of the problem. Further, we extend our work to the network edge where we promote network elasticity and alleviate virtualization technologies by addressing the problem of task offloading and scheduling along with the IoT application resource allocation problem. Given the complexity of the problem, we propose a Logic-Based Benders (LBBD) decomposition method to efficiently solve it to optimality.

Acknowledgments

The knowledge, the hard work and the completion of my Ph.D. degree towards the completion of this thesis would not have been possible without the motivation, the support and the continuous guidance of a great mentor and supervisor, Dr. Chadi Assi. Thank you Dr. Assi for your valuable time, your unconditional availability whenever needed, and for your challenging and fruitful discussions. It is a pleasure working with you.

My deepest appreciation to Dr. Samir Sebbah for his time, patience, guidance and enlightening discussions on the application of many optimization techniques.

I am grateful for Dr. Sara Ayoubi, Dr. Sanaa Sharafeddine, Dr. Ali Ghrayeb, Dr. Mosaddek Hossain Kamal Tushar and Dr. Long Qu for all the enlightening discussions, comments and valuable collaboration on the projects that I completed throughout my Ph.D.

I would like to thank my committee members Dr. Roch Glitho, Dr. Jia Yuan Yu, Dr. Ivan Contreras, Dr. Brigitte Jaumard for their time, their valuable feedback and constructive comments. I would like to extend my appreciation for Dr. Jelena Misic for accepting to serve as my external examiner.

Many thanks for Dr. Daniel Migault, Dr. Stere Preda, Dr. Makan Pourzandi and all Ericsson security team whom I had a great pleasure to meet and work with. Thank you for all your kindness, support and for making my internship at Ericsson of a great value.

I dedicate this thesis to my beloved father, Assem Alameddine, who despite not having the chance to continue his education, worked hard to raise and educate his children. Thank you father for always being there for me unconditionally, believing in my potentials, pushing me to always advance in my education and my career. I love you and I hope that this achievement pays but little of all what you have given me.

I am deeply indebted to my mother, Nahed Alameddine. Nothing would have been possible without your endless care and guidance, mother. Thank you for all your support at all times. I love you.

I would like to express my extreme gratitude to my aunt, Hyame Alameddine, who passed away several years ago and to my uncle, Bassem Alameddine. I am grateful for all your efforts in educating and guiding me. I would not have been the person I am now without you. To my sister Ranime Alameddine, my brothers; Bassem, Karim and Hadi Alameddine, my cousin Zakia Alameddine and her husband Mr. Fawaz Osmani and to my entire family, nothing is more valuable than having you in my life. Thank you for handling my craziness all these years.

Last but not least, I would like to thank the Montreal Operations Research Student Chapter family and everyone who contributed to its success. It has been a great enriching experience working with you. I would like to specially thank my friend Carlos Zetina for his unconditional commitment, confidence, motivation and great ideas.

I am grateful to all my friends and my colleagues who encouraged me during my hard times, advised me when I needed it the most, laughed with me and engraved unforgettable memories. I am blessed to have you.

Finally, I would like to thank Concordia University and PERSWADE, a CREATE program of NSERC, Canada for their financial support which made my Ph.D. work possible.

To those who conquer, believe and never give up on their dreams ...

Table of Contents

List of Figures

List of Tables

List of Abbreviations

2G	Second Generation
3G	Third Generation
4G	Fourth Generation
5G	Fifth Generation
AP	Access Point
BILP	Binary Integer Linear Program
CAPEX	Capital Expenditures
CG	Column Generation
CPU	Central Processing Unit
DTOS	Dynamic Task Offloading and Scheduling
DTOS-LBBD	Dynamic Task Offloading and Scheduling - A Logic-Based Benders Decomposition
DTOS-MILP	Dynamic Task Offloading and Scheduling - A Mixed Integer Linear Program
EM	Element Management
eNB	Evolved Node B
ETSI	European Telecommunications Standards Institute
IDS	Intrusion Detection System
ILP	Integer Linear Program
IoT	Internet of Things
IT	Information Technology

ITS	Intelligent Transportation System
LASS	Latency-Aware Service Scheduling
LASS-Game	Latency-Aware Service Scheduling - A Game theoretic approach
LASS-Game-RandomPath	Latency-Aware Service Scheduling Game under a random path
LASS-Game-RelaxedDeadline	Latency-Aware Service Scheduling Game with network services of Relaxed Deadlines
LASS-Game-SuperTightDeadline	Latency-Aware Service Scheduling Game with network services of Super Tight Deadlines
LASS-Game-TightDeadline	Latency-Aware Service Scheduling Game with network services of Tight Deadlines
LASS-MaxAdmission	Latency-Aware Service Scheduling under a Maximum Admission objective
LASS-MinMaxCT	Latency-Aware Service Scheduling under the objective of Minimizing the Maximum Completion Time
LASS-MinSumCT	Latency-Aware Service Scheduling under the objective of Minimizing the Sum of Completion Times
LBBD	Logic-Based Benders Decomposition
LP	Linear Program
LTE	Long-Term Evolution
MCC	Mobile Cloud Computing
MEC	Multi-access Edge Computing
MILP	Mixed Integer Linear Program
MP	Master Problem
NAT	Network Address Translation
NF	Network Function
NF-FG	Network Function Forwarding Graph
NFV	Network Function Virtualization

NFV-MANO	Network Function Virtualization Management and Orchestration
NFVI	Network Function Virtualization Infrastructure
NFVO	Network Function Virtualization Orchestrator
NLP	Non-Linear Program
NS	Network Service
Online-CG	Online Column Generation
OPEX	Operational Expenditures
PNF	Physical Network Function
PoA	Price of Anarchy
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RC	Reduced Cost
RMP	Restricted Master Problem
SA	Sequential Algorithm
SCeNB	Small Cell Evolved Node B
SDN	Software Defined Networking
SFCS	Service Function Chaining Scheduling
SFCS-CG	Service Function Chaining Scheduling - Column Generation
SFCS-CGD	Service Function Chaining Scheduling - Column Generation with Diversification
SFCS-MILP	Service Function Chaining Scheduling - Mixed Integer Linear Program
SP	Sub-Problem
TS	Tabu-Search
TS-NFMS	Tabu-Search for Network Function Mapping and Scheduling
TSP	Telecommunication Service Provider
UE	User Equipment

VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VNE	Virtual Network Embedding
VNF	Virtual Network Function
VNFM	Virtual Network Function Manager
WAN	Wide Area Network

Chapter 1

Introduction

1.1 Overview

Over the past few years, we have experienced a highly connected lifestyle that tremendously affected our daily activities. There was a time when owning a cell phone was only to talk, to send a message or to surf the web. With the Information Technology (IT) revolution, the cell phone was transformed to a smart phone enabling access to a wide range of applications. In fact, the first smart phone was IBM's Simon which was launched in 1994 during the era of the Second Generation (2G) networks which enabled text messaging in addition to a minor support of data [4]. IBM's Simon came pre-loaded with several applications such as address book, calculator, calendar, mail, notepad, and sketch pad [5]. The evolution of smart phones to include more sophisticated types of applications such as maps, games, weather, enhanced browsers (i.e., Safari) [5] was aligned with the launch and the development of the Third Generation (3G) networks which provided higher data rates and improved Quality of Service (QoS). Throughout the life of 3G networks, many advanced smart devices (i.e., tablets, smart televisions, laptops, etc.) gained momentum as they included new features enabling computation and data intensive applications, that are no more built-in within the device, but are rather downloaded through the Internet [5]. For instance, 3G networks enabled video chatting applications including audio and video files transfers [4]. With the proliferation of different types of applications and services with 3G networks, mobile users required enhanced QoS which led to the launch of the Fourth Generation (4G) networks which we are using today. 4G networks were designed to provide higher data rates enabling a better support of multimedia messaging service, video chatting and video streaming, etc. [4].

4G networks gave birth to a wide new range of services in different sectors such as health, automotive, entertainment and social sectors [6]. The rapid expansion of services is accompanied with the development, innovation and proliferation of electronic devices which are transformed to become smart, that is, they can be connected to other devices or networks for data exchange while being capable of performing autonomous computing (i.e., without the direct command of the user (i.e., autonomously collect information through sensors and send it through a network)) [7]. For example, smart watches which are enabled with health activity tracking capabilities, smart speakers with voice-controlled assistant responding to user questions and commands, smart health devices for measuring blood and oxygen levels, etc. These smart connected devices form an interconnected network of objects, known today as the Internet of Things (IoT) [7]. IoT is an emerging paradigm that has been recently gaining the interest of telecommunication network operators especially after Cisco predicted in 2011 an ultimate growth of the number of IoT devices to reach 50 billion by 2020 [8]. In addition, Cisco presented the IoT as the first real evolution of the Internet given that it will lead to a revolutionary set of applications that will change our lives by improving the way we live, learn, work and entertain ourselves [8]. In fact, IoT devices can collect and exchange data through their deployed sensors which can sense temperature, pressure, vibration, light, humidity and stress. This data can then be processed and turned into valuable information which, combined with other sources of information can provide knowledge and wisdom that allow us to become proactive, hence bringing tangible value to our lives [8]. Clearly, IoT is carrying new challenges to the mobile network operators as their mobile network infrastructure needs to be updated to cope with the ever increasing number of IoT devices and the big data volume they need to exchange over their infrastructure. In addition, the new innovative services seeing the light today through the applications deployed on IoT devices are raising the bar of the QoS and the corresponding Quality of Experience (QoE) requirements of their users [6]. For instance, the emerging autonomous and assisted driving services require ultra-low latency and high reliability [2].

In order to meet the continuous data and service requirements, telecommunication network operators have been constantly upgrading and expanding their mobile network infrastructure by adding expensive legacy equipments that are hard to manage, to maintain and to configure which have been incurring high Capital Expenditures (CAPEX) and Operational Expenditures (OPEX) [6, 9]. Due to the high competition among themselves and the falling prices, they were, however, experiencing a low return on investment [6, 9]. Hence, telecommunication network operators have been forced to find new ways to reduce their investment costs while increasing their revenues along with meeting the QoS and QoE requirements of their mobile users. Towards this end, the telecommunication industry is entering a new phase of the evolutionary era of the Fifth Generation (5G) networks.

1.2 Vision of future 5G networks

The beyond 2020 mobile communications systems also known as 5G networks [10] emerges from the need of network operators to satisfy their consumers' demands of faster, safer and smarter wireless networks [11]. The continuous increase of IoT devices and applications of varying QoS requirements generating huge amount of data urges the need for more sophisticated networks that are highly scalable and which can provide high throughput, low latency in data delivery and high reliability guarantees [11]. The International Telecommunication Union classifies services which need to be supported by 5G networks into three different categories. The first category regroups mobile broadband services such as audio and video streaming which addresses human-centric use cases and provides access to multi-media content, services and data with end-users peak data rates reaching 10 Gbps [12]. The second category targets Machine Type Communications (MTC) which depicts use cases of large number of connected devices communicating with each other with low-volume, non-delay sensitive data such as those present in smart homes and smart cities. The third category gathers ultra low-latency (i.e., 1 ms) and high reliable (i.e., 99.999%) services such as those related to transportation safety, remote medical surgery and virtual reality [11, 12, 13, 14]. Satisfying the variable requirements of these three categories goes beyond ensuring high data rates, ultra-low latency and high reliability to also guaranteeing better coverage, providing higher spectrum and improved security. 5G networks should be highly scalable, agile, elastic, programmable and cost-efficient [10, 11]. Such characteristics cannot be incorporated to current networks due to their traditional usage of specific-purpose hardware equipment which are neither elastic nor can be easily upgraded with new functions [10].

1.3 Enabling Technologies

In light of the presented 5G vision and the limitations of current networks in fulfilling the 5G requirements, many recent technologies have been gaining the interest of both industry and academia alike as they will enable the launch of 5G networks.

1.3.1 Cloud and Multi-access Edge Computing

The on going advances in cloud computing that consists of enabling on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) will highly assist network operators in reaching their 5G networks vision [9, 11]. In fact, cloud computing gained momentum due to its capability in providing scalability, on-demand resource allocation, pay-as-you-use flexible pricing model and reduced management effort enabling an easy provisioning of applications and services [15]. Given its benefits and their need to reduce the burden of large amount of traffic generated by the emerging IoT devices on their core networks, in addition to providing the connectivity of these devices to the cloud through the Internet while satisfying their low-latency requirements, network operators extended the centralized cloud computing towards providing a distributed cloud access at the edge of the network known as Multi-access Edge Computing (MEC) [14, 15, 16]. By providing computing and storage resources at the edge of the network in close vicinity of the user, MEC allows IoT devices to overcome their limited computing resources and short battery life which prevent them from satisfying the latency requirements of their computation intensive applications [14, 17]. MEC offers these IoT devices the option to delegate their processing operations to edge servers by offloading their tasks to be processed by the required applications hosted at the edge of the network [14, 17].

1.3.2 Network Function Virtualization

The advanced virtualization techniques (e.g., use of Virtual Machines (VMs) or containers) explored in the cloud constitute a main enabler for the emerging Network

Function Virtualization (NFV) technology which will pave the way towards facilitating the re-design of current networks to become agile, flexible and scalable [9, 10]. By decoupling the physical network equipment from the functions running on top of them, NFV promises to solve the limitations of specific-purpose hardware also known as "middleboxes" by enabling their deployment as softwares in the network [6, 9]. These middleboxes represent different types of Network Functions (NFs) which offer a wide spectrum of crucial functionalities ranging from traffic shaping (e.g., rate limiters, load balancers), security (e.g, firewalls, Intrusion Detection System (IDS)), Network Address Translation (NAT) and network performance (Wide Area Network (WAN) optimizer, caches, proxies) [18]. Several studies report the rapid growth of the number of NFs. For instance, a study done by Sherry et al. shows that the number of different middleboxes is comparable to the number of routers in an enterprise network [19]. Today, introducing new NFs require the purchase of vendor-specific hardware appliances that can be deployed at fixed locations in the network in addition to specialized personnel to manage and maintain them [9, 20]. NFV suggests replacing these middleboxes by softwares known as Virtual Network Functions (VNFs) that can be deployed on general-purpose hardware platforms (e.g., physical servers, switches) on top of VMs or containers [6, 9, 18]. This provides the flexibility of their deployment on-demand at different network locations without the need for purchasing and installing any new hardware, which leads to an efficient utilization of resources and to an expense reduction [9, 18].

Some NFs can work independently to perform a certain task without any interaction with other NFs [18]. This is the case of a network monitor function responsible of traffic measurements and monitoring which help in efficiently managing network resources [18]. However, other NFs are supposed to be chained together in a certain order to describe a defined Network Service (NS). This chain of NFs can be represented by a NF Forwarding Graph (NF-FG). A NF-FG is graph composed of logical links connecting NF nodes designed to describe how traffic flows between these NFs [21]. For instance, in order to provide a security service, the network administrator may decide to process all HTTP requests by an ordered chain of NFs composed of a $firewall- > IDS- > proxy$ [18, 20]. Enforcing the chaining policy, that is, ensuring the ordered processing of the traffic by the specified NFs in the NF-FG is usually performed manually by modifying the forwarding tables entries of the routers which is a cumbersome and error prone process [22]. While such manual configuration of routing tables may be efficient in the case where the NFs are hardware middleboxes given that the locations of these latter rarely change due to the very high cost incurred by such operation, it is impractical in the case where NFs are implemented as VNFs [22]. In fact, an important advantage that VNFs bring is the option of their re-location in the network, in addition to the possibility of their instant deployment based on varying network conditions. In these cases, the routing tables need to be continuously updated and maintained which implies the necessity for a new technology, Software Defined Networking (SDN), that helps in their automatic configuration [6].

1.3.3 Software Defined Networking

SDN is an emerging paradigm that is designed to simplify network management by offering an architecture that provides flexible routing by separating the network control logic (the control plane) from the underlying routers and switches (the data plane) responsible of forwarding the traffic according to the decisions made by the control plane [6, 9, 23]. Decoupling the control functionality from the network device allows network operators to eliminate the operations related to configuring each network device separately using vendor-specific commands in order to enforce the desired network policy. In addition, they will be relieved from handling the possible faults that can be endured from this manual configuration. SDN moves the control logic to an external logically centralized entity called SDN controller. Instructed by network applications about the desired network policy (e.g., a routing application decides on the path of a flow), the control plane installs the appropriate forwarding rules in all forwarding devices' routing tables [23]. SDN provides a logically centralized network management through its controller that maintains a global view of the network. Such global view of the network helps in delivering network agility to dynamically adjust to varying network conditions [9]. Further, SDN is identified for enabling programmable networks where network configurations can be automated and new service policies can be dynamically enforced [9, 23].

1.3.4 The Key Technologies Interaction

The combination of cloud computing, NFV, SDN and MEC will play an important role in the design and implementation of 5G networks. By exploiting virtualization techniques, cloud computing offers on-demand resource pooling which provides better resource utilization of the underlying infrastructure [9, 11]. This will highly assist 5G networks in handling increasing traffic demands. Further, the cloud will offer the physical and virtual resources that can be used to host VNFs, hence, enabling the rapid deployment of new services [9]. NFV will free current networks from their dependency on specific-purpose hardware by replacing them by VNFs that can be automatically and instantly instantiated and scaled on-demand, and managed through a common interface. Thus, NFV will unleash the power of current networks due to the flexibility, manageability and cost-efficiency benefits that it brings [9, 10, 11]. NFV will help in the rapid integration of SDN in current networks given that the

SDN controller can be deployed in the form of a VNF [9]. Further, SDN will assist NFV in enforcing the desired chaining policy between different VNFs that need to communicate together in order to deliver a NS. SDN will transform current networks into programmable ones that can dynamically adapt to changing network conditions [9, 11]. Finally, all these technologies can be incorporated at the edge of the network to deliver edge computing capabilities that will provide ultra-low latency services to the emerging IoT devices through the MEC.

1.4 Challenges and Contributions

A successful integration of all the aforementioned technologies, to promote the desired QoS and QoE foreseen in 5G networks, requires an efficient strategy that can capture the dependency between the physical network infrastructure resources and the virtual resources deployed on top of them while maintaining the decoupling aspect of these latter that has been introduced by NFV. Further, such strategy should be adaptive to address the different requirements of the targeted 5G services (i.e., mobile broadband services, MTC, ultra-low latency high reliable services). Finally this strategy should deliver the promised advantages of the different used technologies including flexibility, manageability and programmability. Developing such a strategy is contingent to solving multiple challenges in a NFV-based network. These challenges can be summarized as follows:

1. *The VNF placement problem*: As VNFs are softwares that can run on-top of VMs or containers deployed on general-purpose hardware, a decision on the placement of these VNFs in the network is a must, that is, which general-purpose hardware will host which VNF instance.

2. *The NF mapping problem*: NFs can be chained together to form a NS. Since

multiple services may require the same type of NF and given that many instances (VNFs) of the same NF can be deployed in the network, a decision on which VNF instance will process the traffic of which NS is important in order to ensure efficient resource utilization.

3. *The traffic routing problem*: For an efficient service provisioning, the chaining policy of the different NFs composing the service needs to be enforced. This requires determining the route of the traffic of the mentioned service between the different VNF instances destined to process it. Such route can be specified with the help of SDN. It should account for a specific bandwidth allocation on the routing path in order to provide guaranteed performance.

4. *The NS scheduling problem*: As VNFs can be shared between the traffic of many NSs, an efficient scheduling of the processing of the traffic of these NSs on each of the shared VNFs is to be determined. Such scheduling should also be compliant with the NF-FG of each NS.

This thesis aims at providing a deep understanding of the aforementioned challenges and their interdependency while proposing several novel approaches that efficiently address them under different network designs and assumptions. We present our main contributions in the following.

1.4.1 The Interplay Between Network Function Mapping and Scheduling

In order to deliver a NS, one needs to first place the VNFs composing the service in the network. Given the VNFs placed in the network, one can then decide on those that will process the traffic of the mentioned service, and then determine the route, compliant with the NF-FG, that should be taken by the traffic. Finally, the schedule of the service on each of the VNFs processing its traffic can be decided. While such sequential approach provides a legitimate solution of the aforementioned challenges, we notice that it does not take advantage of the interplay that exists between them. Hence, we explore, study and evaluate the interplay that exists between the NF mapping, traffic routing and service scheduling problems and highlight the benefits of jointly addressing them. Thus, we first mathematically formulate this joint problem and then propose a novel Column Generation (CG) approach to solve it with the objective of minimizing the total schedule length of the services. Our CG provides a Linear Program (LP) lower bound on the optimal Integer Linear Program (ILP) solution and an ILP upper bound. Further, we capture the offline and online aspects of the problem and we show through extensive numerical evaluation that jointly addressing the NF mapping, traffic routing and service scheduling provides an efficient resource utilization.

1.4.2 Enabling Low-Latency Services in Softwarized Networks

While the CG approach previously developed can serve as a benchmark algorithm due to its limited scalability, and since 5G networks focus on delivering ultra-low latency services, we revisit the joint NF mapping, traffic routing and service scheduling problem while addressing services with stringent deadlines. The targeted objective is to maximize the number of admitted services, that is, those that are able to meet their latency requirements. We mathematically formulate the problem and propose a novel game theoretic approach to solve it. The proposed approach provides a distributed decision model where each NS is free to decide on its own NF mapping, traffic routing and service scheduling solution. However, the coherence of the schedule of the different NSs is captured by a centralized controller that prevents them from contending for the same resources (i.e., VNFs) simultaneously. We evaluate our proposed approach in an offline and online settings under different system parameters and routing schemes while highlighting the impact on the admission rate. We show that our approach is scalable and that the price of anarchy of the game is upper bounded by $\frac{1}{2}(3+\sqrt{5})$.

1.4.3 Low-Latency IoT Services in Multi-access Edge Computing

As MEC aims at providing ultra-low latency services while making use of advancement in NFV, we extend our work to the network edge. Thus, we address the joint problem of task offloading and scheduling in MEC. While the task offloading resembles to the NF mapping problem as it aims at deciding which IoT application, that can be addressed as a VNF [24], will process the task of which IoT device; the task scheduling problem consists of determining the order in which each task should be processed on the shared application while meeting its latency requirement. Unlike our previous work where we considered that the processing resources allocated to each VNF are pre-determined, we assume, in this work, that the computing resources of the different deployed applications need to be determined with the objective of maximizing the number of served tasks. Addressing the joint task offloading and scheduling problem under undefined IoT applications computing resources brings another level of complexity to the problem due to the direct impact of the computing resources allocated to the application on the processing time of the task by this latter. However, jointly deciding on the task scheduling along with the computing resource allocation of the IoT applications is highly interesting as it promotes the elasticity aspect provided by the different virtualization techniques and allow the automatic accommodation of resources to the variable traffic size and requirements. Hence, we mathematically formulate the problem and propose a Logic-Based Benders Decomposition (LBBD) approach to solve it. LBBD provides the optimal solution of the problem. Through extensive performance evaluation, we explore valuable performance trends to highlight the impact of task offloading and scheduling in meeting the diverse QoE requirements aligned with 5G vision.

1.5 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 discusses in details the NFV technology and its main challenges while providing a thorough review of the state-of-the-art. Chapter 3 explores the interplay that exists between the NF mapping, traffic routing and service scheduling problems and presents a CG approach to jointly solve them. Chapter 4 targets low-latency services and presents a game theoretic approach to solve the aforementioned joint problem under different assumptions. Chapter 5 builds on top of the previous work by targeting the task offloading and scheduling of low-latency services. It overviews edge computing concepts and presents a LBBD approach to solve the problem. Finally, Chapter 6 concludes the thesis and highlights potential research problems for future consideration.

This thesis relies on several optimization and game theoretic techniques. Thus, for a quick review and completeness, we present in Appendix A a brief introduction and overview on the different techniques that are used in this manuscript.

Notations which are used throughout the thesis are independent from one chapter to another. Hence, some symbols may appear in different chapters and serve different purpose.

Chapter 2

Network Function Virtualization: Overview, Challenges and Literature Review

This chapter presents an overview of the NFV technology while explaining and defining different related keywords and concepts that will be used throughout this manuscript. It presents and explains the different elements composing the NFV architectural framework. Nonetheless, this chapter also discusses different challenges in NFV, mainly the VNF placement, the NF mapping, the traffic routing and the NS scheduling problems which are tackled throughout this thesis. Further, it analyses the relationship between the aforementioned problems and reviews the works in the literature.

2.1 Overview and Definitions

Traditionally, provisioning a NS requires network operators to deploy proprietary devices and equipment for each NF defined as part of the NS. These NFs, also known as components of the NS, usually have strict chaining or ordering requirements which should be reflected when deploying them in the network. They have many drawbacks from which we mention limited flexibility, high cost, manual deployment and chaining, in addition to frequent hardware upgrades. The dependence of network operators on these specialized hardware, that we refer to by middleboxes resulted in high CAPEX and OPEX. Furthermore, the emergence of new services requires Telecommunication Service Providers (TSPs) to continuously purchase, store and operate new physical equipment which demand highly trained and skilled personnel that can manage and operate these vendor-specific hardware [9, 20].

These limitations forced TSPs to look for new ways to solve the aforementioned challenges in order to be able to flexibly and automatically adapt to changing users demands. Hence, NFV was proposed as a promising solution to solve these challenges. Seven companies (AT&T, BT, Deutsche Telekom, Orange, Telecom Italia, Telefonica and Verizon) selected the European Telecommunications Standards Institute (ETSI) to be the main party responsible of addressing NFV specification and challenges [9].

NFV leverages virtualization technologies such as VMs and containers to change the way NSs are designed, deployed and managed [9, 20, 25]. NFV consists of deploying NFs as VNFs instead of Physical Network Functions (PNFs). NFs, VNFs, and PNFs can be distinguished as follows:

1. *NF*

   A NF is a functional block within a network infrastructure that has well-defined external interfaces and well-defined functional behavior [21]. Examples of NFs include firewalls, IDS, load balancer, etc.

2. *VNF*

   A VNF is an implementation of a NF that can be deployed on a virtual resource such as a VM or a container. Note that while VMs and containers represent a virtualized computation environment that behaves like a physical server or computer (i.e., has a processor, a memory, etc.); VMs run their own operating systems while containers are more light weight as they do not require a duplication of the operating system [9, 21].

3. *PNF*

   A PNF is an implementation of a NF via a tightly coupled software and hardware system [21]. In the rest of this manuscript we will use middlebox and PNF interchangeably.

A NS is an offering provided by the TSP to its end users [9]. It is composed of one or more NFs, which combined, determine a certain functionality [9, 21]. The NFs composing the NS are chained in a specified order, that is, the traffic of the NS should be sequentially processed by an ordered chain of NFs [9, 20, 21]. Such chaining of NFs is represented by a NF-FG. Violation of the ordered processing of the traffic determined by the NF-FG may lead to its erroneous processing causing degraded performance and security breaches [20]. For instance, if a cache processed the traffic prior to a firewall, the cache may be compromised by malicious external traffic putting all the network at risk [20]. For more clarity, we define a NS and a NF-FG as follows:

1. *NS*

   A NS is a composition of NF(s) defined by its functional and behavioral specification. The end-to-end NS behavior is the result of the combination of the individual NFs behaviors and their ordered chaining [21].

2. *NF-FG*

   A NF-FG is a graph of logical links connecting NF nodes for the purpose of describing traffic flow between these NFs [21].

Note that, in the remaining of this manuscript, we use “NF” to refer to the functions requested by the NS in its NF-FG; whereas, we use “VNF” to depict the softwares that are actually deployed in the physical network.

NSs determined in NFV can be different from current existing NSs as they can be defined by a chained set of virtualized (VNF) and/or non-virtualized (PNF) NFs which will require interoperability among legacy and NFV-based network domains [26]. Thus, in the following, we present a NFV architectural framework that supports the diversity of NFs and standardizes their deployment, management and orchestration.

2.2 NFV Architecture

ETSI presents a NFV architecture (Fig.2.1) composed of three different administrative domains:

1. *Network Function Virtualization Infrastructure (NFVI)*

   The NFVI regroups virtualized and non-virtualized resources encompassing different software (i.e., VMs) and hardware resources such as commercial-of-the-shelf computing hardware, storage and network (i.e., nodes and links) that enables the deployment and the connectivity of different VNFs [9, 26, 27].

2. *VNF*

   A VNF is the software implementation of a NF and run on top of the NFVI [26].

3. *NFV Management and Orchestration (NFV-MANO)*

   The NFV-MANO framework is responsible of the orchestration and life cycle management of NSs including all relevant functions ranging from the deployment of VNFs and the provisioning of their infrastructure resources provided by the NFVI [27].

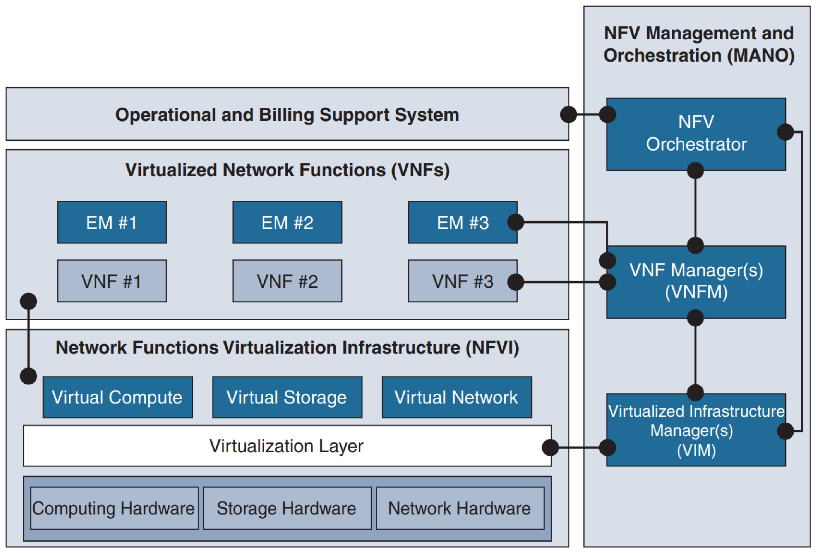

NFV Management and Orchestration (MANO)
Operational and Billing Support System
NFV Orchestrator
Virtualized Network Functions (VNFs)
EM #1
EM #2
EM #3
VNF #1
VNF #2
VNF #3
VNF Manager(s) (VNFM)
Network Functions Virtualization Infrastructure (NFVI)
Virtual Compute
Virtual Storage
Virtual Network
Virtualization Layer
Virtualized Infrastructure Manager(s) (VIM)
Computing Hardware
Storage Hardware
Network Hardware

Figure 2.1: NFV architectural framework [1].

Next, we highlight in details the different building blocks of the NFV-MANO framework.

2.2.1 NFV-Management and Orchestration Framework

The NFV-MANO framework (Fig.2.1) presented by ETSI defines the different functionalities related to the provisioning of resources required to deploy, manage and configure multiple VNFs to provide different NSs [9, 27]. It mainly regroups three building blocks:

1. *Virtual Infrastructure Manager (VIM)*

   The VIM handles the NFVI resource management including computing, storage and network resources that will enable the deployment of VNFs. It is also responsible of collecting performance and resource fault information [26, 27].

2. *VNF Manager (VNFM)*

   The life cycle management of one or more VNFs, including the instantiation, software update, scaling and termination of VNF instances, is attributed to a VNFM. The VNFM collects virtualized resource performance measurements and fault information from the VIM to take some decisions related to scaling the resources provisioned for a VNF for example. Further, it coordinates and interacts with the Element Management (EM) to perform some management operations such as fault, configuration, performance, accounting, and security [1]. The EM has some overlapping functionalities with the VNFM, however, the EM interacts with the VNFs through proprietary methods. It can play the role of a proxy by exposing the VNF management functions to the VNFM [1].

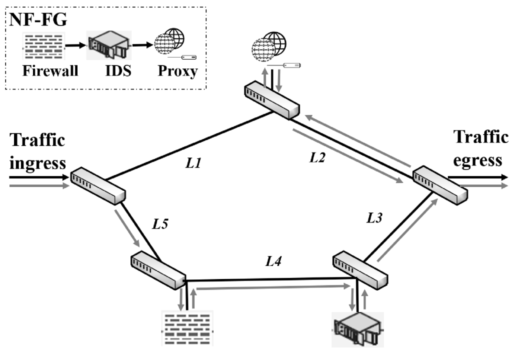
3. *NFV Orchestrator (NFVO)*

   The NFVO performs resource orchestration. That is, it plays an important role in the orchestration (allocation/de-allocation) and management of the NFVI resources across multiple VIMs. It manages the instantiation of VNFMs and coordinates with them to decide on the instantiation of VNFs. The NFVO fulfills another major functionality related to the service orchestration that yields determining the interconnection of VNFs to provide a NS [1, 27]. Hence, the NFVO is responsible of the NS life cycle management including its update, scaling, performance and termination [27].

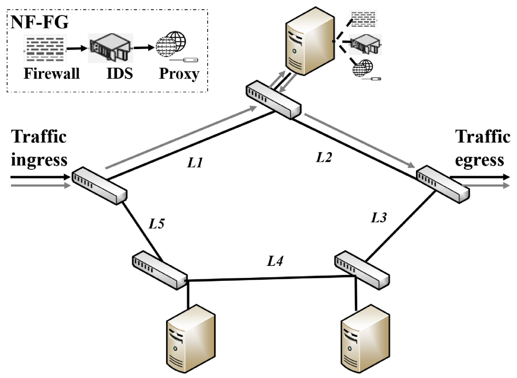
Finally, it is worth noting that the NFV-MANO framework encompasses a set of reference points to enable the communication among different NFV functional blocks as well as well the coordination with traditional network management systems such as operations support system and business/billing support systems to allow the management of both VNFs as well as functions running on legacy equipment (PNFs) [9].

The NFV architecture promotes the promised benefits of NFV in providing flexibility, manageability and short time to market of the emerging NSs by replacing PNFs by VNFs. To further highlight these benefits, we present in the following the differences existing between service chaining in a PNF-based network versus a VNF-based network and introduce the different related challenges.

2.3 Challenges in NFV



2.2.a PNF-based network.



2.2.b VNF-based network.

Figure 2.2: PNF versus VNF based network challenges and opportunities.

To highlight the challenges introduced by NFV, we consider the example in Fig.2.2. We account for a NS represented by a NF-FG depicted in Fig.2.2 and composed of a firewall, an IDS and a proxy server. For the operator to provide the NS, it needs first to provision the NFs. Thus, we first consider that the NFs are provisioned by the operator as PNFs as shown in Fig.2.2.a. Once the provisioning is performed, the traffic needs to be steered from the ingress node to the egress node through these PNFs in the same order specified in the NF-FG representing the NS. Hence, the route selected to steer the traffic shown in Fig.2.2.a by the arrows, depicts the traffic traversing link $L2$ twice, thus wasting the network bandwidth. Given that the location of middleboxes are fixed, that is, once deployed, it is hard and impractical to move them to other locations in order to adapt to traffic demands, one can note that enforcing the chaining policy in a PNF-based network can be at the cost of network resource consumption.

The use of VNFs can solve this problem given that they offer the flexibility to be deployed anywhere in the network and even collocated on the same physical server as shown in Fig.2.2.b. In this case, the traffic will not have to traverse the same route multiple times as in the case of PNFs (Fig.2.2.a), thus, significantly reducing the bandwidth consumed. It is clear from this example, that the usage of VNFs provides the flexibility of jointly optimizing their placement and the traffic route with respect to the NF-FG as they can be deployed anywhere in the network. Such optimization promotes efficient resource utilization which can lead to an increase in the TSPs revenues as they will be able to serve an increasing amount of traffic.

In the example shown in Fig.2.2, one VNF instance of each NF was deployed in the network, however, in the case where many VNFs of the same NF were provisioned in the network, the challenge becomes to choose which of these VNFs will process the traffic of the NS.

Further, given that the traffic of many NSs can be processed by the same VN-F/PNF, the load on these latter may increase and surpass their capacity. In this case, the NS may be rejected from the network when it is destined to be processed by a PNF, however, in a VNF-based network, a new VNF instance can be instantly provisioned and the traffic of the NS can be redirected to be processed on the new deployed instance. Finally, it is worth noting that when many NSs are sharing the same VNF, the processing of their traffic should be scheduled on it.

Hence, in order to reach the desired benefits of NFV, many challenges that we elucidate in the following need to be tackled.

2.3.1 The VNF Placement Problem

As VNFs can be deployed on VMs running on commodity hardware (e.g., physical servers, switches) which have finite amount of memory, compute and storage capacity, efficient resource utilization should be performed in order to achieve the economies of scale [9, 28]. While each VNF can be hosted on a dedicated VM, the authors of [9, 28] argue that such approach may result in a high VM footprint that can be avoided by allowing multiple VNFs to share the same VM. Hence, decisions about where to place each VNF should be made while guaranteeing that their computing requirements are met and while accounting for several objectives such as load balancing, energy saving, CAPEX and OPEX, etc. [9, 22, 25].

The VNF placement problem is similar to the Virtual Network Embedding (VNE) problem that consists of allocating virtual resources to both nodes and links in the substrate network [22, 28, 29]. However, unlike the case of VNE where the virtual network topology is given [30], a NS is composed of a set of VNFs with precedence constraints. In addition, the computing and network resources required are mostly static in VNE whereas they are variable in a NS and depend on the traffic load and the order of the VNFs in the chain [22]. For instance, some VNFs perform data compression/decompression (e.g., video streaming compression) and hence the bandwidth requirements between VNFs in the chain may vary [22, 31]. Thus, given a set of requested NSs, the VNF placement problem seeks at determining the number [25] and the placement of VNFs in the network while satisfying their computing requirements and optimizing a specific objective (i.e., minimize resources consumption [32], maximize the number of admitted requests [33], minimize OPEX [25], etc.).

The VNF placement problem has been proven to be NP-Hard [22, 25, 33, 34] and hence, many heuristics and optimization techniques have been proposed to solve it [25, 33, 34, 35, 36].

2.3.2 The NF Mapping Problem

Given a set of VNFs provisioned in the network, the NF mapping problem also referred to as the assignment problem [37] consists of mapping the NFs of a NF-FG of a given NS to the VNFs deployed in the network [25, 38]. NFs should be mapped to VNFs instances of the same type (i.e., firewall, proxy, etc.) while respecting their processing capacity [25]. The NF mapping problem defines which deployed VNF instances will be in charge of processing the traffic of each NS [37].

The NF mapping problem is NP-Hard as explained in [30]. It is usually jointly addressed with several other problems such as the VNF placement problem [25, 39], the scheduling problem [38], while considering several objectives ranging from minimizing the resources consumption[30] and the OPEX [25].

2.3.3 The Traffic Routing Problem

Given that the traffic of a NS needs to be processed by a chain of VNFs in a defined order, techniques to steer the traffic through them should be applied. In traditional communication networks where NFs are implemented in the form of middleboxes, changes on the traffic flow path is made so that it can traverse through the specified ordered chain of middleboxes [32]. This is usually done by manually crafting routing tables which is an error prone process [40, 41]. It is also difficult for network operators to continuously and manually adjust network configuration in response to changing network conditions [9].

To overcome these challenges, SDN has been employed to dynamically adjust to network traffic flow based on changing demands. Hence, SDN provides automatic network configuration through an SDN controller that dictates the overall network behavior [23]. Above the technicality of enforcing the service chaining policy, the selected routing path should account for the bandwidth that needs to be guaranteed for the transmission of the traffic [39] in addition to the latency requirements of the NS [9, 37]. Hence, the traffic routing problem consists of determining the routing path that interconnects the placed VNFs [37].

The traffic routing problem is NP-Hard [34] and is usually addressed in conjunction with the VNF placement problem [25, 36, 37, 39] given the interplay that exists between them. For instance, a shortest path may be selected in case of limited network bandwidth and sufficient VMs whereas a longer route may be chosen to promote the reuse of VNFs given their limited number/computing resources, while tolerating more bandwidth consumption [39].

2.3.4 The NS Scheduling Problem

NFs of different NSs may be mapped to the same deployed VNF. Hence, their traffic will be sharing the same computing resources assigned to the VM on which the VNF is deployed. For such sharing to be possible, scheduling techniques should be applied [9]. Standard operating system schedulers do not make decisions that account for chain level information (e.g. order of NFs) [42]. Determining how to dynamically schedule NFs is crucial to achieve high performance of the NSs [42].

Hence, the NS scheduling problem consists of determining the processing time slots on each VNF of the various NSs sharing the same VNF subject to their chaining and latency requirements [31, 38, 43]. In other words, the NS scheduling problem aims at deciding on the allocation of the VNFs computing resources (i.e, CPU schedule allocation) to the traffic of a given NS.

The NS scheduling problem can be formulated as a Resource Constrained Project Scheduling Problem (RCPSP) known as NP-hard [43, 44]. Few works in the literature addressed this problem by formulating it with the objective of minimizing the schedule length needed to serve all the NSs [31, 43]. Heuristics and meta-heuristics have been also proposed to solve this problem [31, 38].

In the following, we present a detailed review of the literature works that tackled the above challenges.

2.4 Literature Review

We will first present a literature review of the works that addressed the VNF placement and traffic routing problems as they are greatly tighten together and implicitly consider the NF mapping problem. We will then overview the works done on the NS scheduling problem.

2.4.1 VNF Placement and Traffic Routing problem

Many studies tackle the VNF placement and traffic routing problems jointly using different techniques which vary between algorithmic solutions, optimization techniques and game theoretic approaches.

2.4.1.1 Algorithmic Approaches

The early work of PACE [36] seeks at presenting a policy-aware application cloud embedding approach that guarantees applications isolation in the cloud. Such isolation induces no sharing of any NFs, routing and forwarding tables and no communication between VMs of different applications. To this end, they propose a flow security graph to abstract each application's requirements including network bandwidth, computing resources and reliability requirements. Using the defined flow security graph, and the given number of VNFs to deploy, they address the VNF placement problem and traffic routing problems disjointly while overlooking the ordering of the required

NFs. The authors of [36] provide an online algorithm to solve the VNF placement problem. They account for the obtained VNF placement solution to solve the traffic routing problem. Their objective is dictated by maximizing the benefits of the allocated requests. PACE strategy in preventing the sharing of VNFs between multiple applications in order to provide their isolation results in an inefficient resource utilization and over-provisioning of VNFs.

The work in [32] solves the VNF placement problem by taking into consideration the chaining policy depicted in the NF-FG. The authors of [32] argue that the traffic steering technique, used to enforce chaining policy, may cause forwarding loops, congestion and overload of middleboxes. Traffic steering refers to directing the traffic so that it traverses through the specified ordered set of middleboxes. To overcome these shortcomings, the authors of [32] propose placing the VNFs on the path of each flow, rather than modifying the flow path in order to enforce the chaining policy. Their VNF placement solution considers deploying one VNF instance in the network path of each flow for each required NF. They argue that VNFs can be either deployed on VMs or processed on a router or a switch. Hence, they provide a mathematical formulation of the VNF placement problem with the objective of minimizing hardware resource consumption. They seek at deploying the minimum number of VNFs by considering sharing them among different flows. Given the complexity of the VNF placement problem, they develop an algorithm to solve it. The work in [32] overlooks guaranteeing bandwidth for the communication between these VNFs.

Bari et al. [25] solve the VNF placement and traffic steering problems jointly. They provide an ILP formulation that accounts for the ordering of NFs defined in the NF-FG with the objective of minimizing the OPEX (i.e., VNF deployment cost, energy cost, traffic forwarding cost, etc.) and resource fragmentation cost. The work in [25] argues that minimizing the resource fragmentation that represents the percentage of idle resources of active servers and links can increase the admission of more traffic. Given the computational complexity of the joint problem, they propose a dynamic programming based heuristic to solve it.

The authors of [37, 45] account for the end-to-end delay of the NS. In fact, the work in [37] argues that the VNF placement and traffic routing solution should keep the NSs end-to-end delays comparable to those observed in traditional middlebox-based networks. Thus, the authors propose an ILP formulation for the joint VNF placement, NF mapping and traffic routing problem while considering that the total end-to-end latencies between VNFs, in addition to the delay incurred by the packets processing on the VNFs should be less or equal than a maximum allowable delay. With the objective of reducing the number of deployed VNFs and given the complexity of the problem, they propose a heuristic to solve it. Similarly, the authors of [45] mathematically formulate the same problem with the objective of minimizing the VNF placement and chaining of NSs including the cost of deploying the VNFs, the cost of using the servers and the cost of communication (i.e., bandwidth). They accounted for QoS guarantees which they enforce by ensuring that the processing and transmission delays of the traffic of a NS fall within a maximum allowed latency. Given the complexity of the problem, they propose a cost-efficient centrality-based VNF placement and chaining algorithm to solve it.

2.4.1.2 Optimization Techniques

The work in [33] addresses the joint VNF placement, NF mapping and traffic routing problem with the objective of maximizing the number of admitted traffic flows. The authors of [33] mathematically formulate the problem and propose a cut and solve based approach that provides the optimal solution of the problem. Similarly, the authors of [46, 47] use a decomposition technique to jointly solve the VNF placement, NF mapping and traffic routing problems by employing a column generation approach with the objective of minimizing the bandwidth consumed. Addis et al. [35] study the joint problem of VNF placement and traffic routing optimization while considering the bit-rate variations at each VNF due to specific operations (compression/decompression). They mathematically formulate the joint problem with a dual objective of minimizing the maximal link utilization and the allocated computing resources. They propose a math-heuristic approach to solve it where they first solve the problem by optimizing the link utilization, then they account for the obtained solution to optimize the allocated computing resources.

2.4.1.3 Game Theoretic Approaches

Few works in NFV used the game theoretic technique to address some of the challenges faced by this technology. For instance, the work in [48] formulates the joint problem of VNF placement and traffic routing as an ILP and presented a game theoretic approach to solve it. The authors consider that each NS is a player which chooses the placement of its VNFs and the shortest path to route its traffic with the aim of minimizing its own operating cost. Salvatore et al. [49] solve the VNF placement problem while accounting for the communication latencies between servers that can host VNFs through proposing a congestion game where forwarding graphs act as players. The authors of [50] formulate the VNF placement problem as a partitioning game where the VNFs corresponding to the NF-FG are the players. Each VNF has to choose a partition (i.e., VM) to be placed on with the objective of minimizing its allocation/moving cost. The work in [51] presents a mixed strategy non-cooperative game where each NS is a player which has to choose a provisioning scheme for its NF-FG from those provided by the broker and which maximizes its profit and meet its QoS requirements (i.e., latency).

2.4.2 NS Scheduling Problem

The NS scheduling problem received little attention. It was mainly introduced in [38, 52]. Riera et al. [52] formulate the NS scheduling problem as a flexible job-shop problem with the objective of minimizing the makespan.

The work in [38] formulate the online NF mapping and scheduling problems and presented different algorithmic solutions to solve it. Mijumbi et al. develop the greedy fast processing algorithm which consists of mapping NFs to the nodes that offer the best processing times. They propose the greedy best availability which maps NFs to nodes whose current function queue has the earliest completion time. The authors develop the greedy least loaded algorithm that maps NFs to the node with highest available buffer capacity. Finally, they propose a Tabu-Search (TS) based meta-heuristic that starts by an initial solution which performs a random placement of VNFs of a NS to candidate VMs. Each VNF is then scheduled on the VM where it has been placed. This initial solution is improved by a tabu move that consists of moving the VNF with the biggest preceding time gap from one VM to another. A preceding time gap is defined as the time between the completion of a preceding VNF and the start of processing of the current VNF.

Following these two works, Qu et al. [31] formulate the NS scheduling problem as Mixed ILP (MILP) and develop a genetic algorithm to solve it. They consider that the bit-rate demand of each flow can change along the NF-FG due to some specific NFs operations (e.g., compression/decompression). Thus, while minimizing the makespan, they explore the benefits of dynamic bandwidth allocation and bit-rate variation for the communication between the VNFs in improving the network performance. None of the above works on the NS scheduling problem study the impact of traffic routing on scheduling delays.

The scheduling problem in NFV has been addressed in different context. For instance, queuing theory has been used by [53] to solve the VNF placement and scheduling problem. The authors of [53] consider a single queue for each VM running different types of VNFs. Each VNF can process a maximum number of requests at a time. kulkarni et al. [42] propose NFVnice, a NF scheduling and NS chain management framework that provides fair resource scheduling capabilities by providing a preemptive schedule for NFs sharing the same Central Processing Unit (CPU) resources. The proposed framework achieves backpressure for NS chain-level congestion control by avoiding unnecessary packet processing early in the NF-FG if it might be dropped later on. This is the case where an overload is detected by an upstream VNF which is supposed to process the determined packet. In [54], the authors solve the multi-flow scheduling problem in the context of SDN where switch tables need to be modified to adapt to network updates. With the objective of reducing the update time, the work in [54] formulates the network update problem and proposed a linear based polynomial-time algorithm to solve it.

2.5 Conclusion

In this chapter, we presented and defined several concepts and keywords related to NFV. We explored the NFV-MANO framework. In addition, we explained through a concrete example the difference between a PNF-based network and a VNF-based network in the quest of highlighting the benefits brought by NFV. We also identified, explained and reviewed major challenges in NFV which are the VNF placement problem, the NF mapping problem, the traffic routing problem and the NS scheduling problem.

Chapter 3

On the Interplay Between Network Function Mapping and Scheduling in VNF-Based Networks[1]

The short time to market of NSs that NFV promises should be leveraged by an efficient utilization of TSPs' networks. This is motivated by a smart processing of NSs through their required NF-FGs, which is not a trivial task as it requires solving three inter-related problems; the NF mapping problem, the traffic routing problem and the NS scheduling problem. This chapter highlights the interplay that exists between the three aforementioned problems and proposes a novel primal-dual decomposition using CG that solves exactly a relaxed version of the joint problem and can serve as a benchmark approach. A numerical evaluation of the discussed method is presented and shows that CG can attain optimal solutions substantially faster than the initial MILP designed for this problem. Finally, this chapter underlines several engineering insights for improving the network performance.

[1]This chapter has been published in IEEE Transactions on Network and Service Management [55].

3.1 Introduction

Introducing NFV to current networks is a daunting task as it requires overcoming several technical challenges when deploying and managing VNFs. Among their interests in increasing their revenues, TSPs are advised to promise a certain level of QoS depicted by a guaranteed network performance and NS response time [25, 56]. Hence, in order to fulfill these requirements, they need to implement a cost-effective VNF allocation policy able to ensure efficient utilization of their network resources. Such policy should respect the chaining of NFs demanded by a NS and represented in its NF-FG. Satisfying such chaining is of paramount importance in preventing the erroneous processing of the NS traffic and the violation of its performance and security.

With the continuous change of the requirements of emerging NSs and the increasing size of networks, automated NF mapping, traffic routing and NS scheduling constitute fundamental elements for enforcing such cost-effective VNF allocation policy. Thus, in this chapter, we build our work on top of the studies done in [38, 31], while exploring the cross-layer interaction that exists between the NF mapping, traffic routing and NS scheduling problems. We jointly solve these three problems and refer to them by the *Service Function Chaining Scheduling problem (SFCS).* Not only do we consider transmission and processing delays but also we guarantee bandwidth to route the traffic between the VNFs. Through several motivational examples, we expose the combinatorial complexity of the problem given the large number of schedules and routing paths possible for a determined NF mapping corresponding to a NS. Due to the large solution space that requires an efficient technique to explore and enumerate possible feasible solutions, we investigate and present a novel primal-dual decomposition approach using CG [57, 58]. To the best of our knowledge we are the first to solve the SFCS problem using this technique.

3.1.1 Novel Contributions

The contributions of this chapter can be summarized as follows:

1. We study and present several motivational examples that highlight the interplay which exists between the NF mapping and NS scheduling problems on one hand, and the traffic routing and NS scheduling challenges on the other hand.

2. We define and formulate the SFCS as a MILP *(SFCS-MILP)* able to provide the optimal mapping, routing and scheduling of the requested NSs while minimizing the total schedule length.

3. Given the complexity of the problem, we explore and introduce a cross-layer strategy to solve the SFCS problem by applying a primal-dual decomposition using CG*(SFCS-CG)* [57, 58, 59, 60]. We divide the problem into a Master Problem (MP) and multi-pricing Sub-Problems (SPs). Each pricing SP defines a feasible column for a given NS. Here, a NS column holds the NF mapping, the traffic routing and the NS scheduling decisions. The MP selects a column for each NS while making sure that the resources capacity constraints are not violated. The MP tries to choose the subset of columns that reduces the schedule length of all the NSs.

4. CG provides an LP lower bound on the optimal ILP solution of the SFCS-MILP formulation and an ILP upper bound. In order to improve the gap that exists between the LP lower bound and the ILP upper bound. We apply several diversification techniques able to explore the solution space and feed the master model with additional columns.

5. We develop an online CG approach where the CG algorithm is run periodically on a batch of NSs. We show that by reducing the schedule length of all the NSs, network operators will be able to serve more traffic in a shorter period of time, hence efficiently utilizing their resources and increasing their revenues.

6. Our numerical evaluation shows that our SFCS-CG approach is much more scalable than our SFCS-MILP formulation.

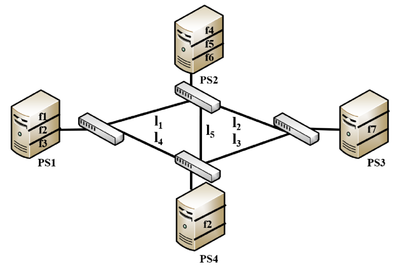
The remainder of this chapter is organized as follows: Section 3.2 exploits the interplay that exists between the NF mapping, the traffic routing and the NS scheduling problems and explains their impact on the resource utilization. Section 3.3 provides a definition and a formulation of the SFCS problem. Section 3.4 explains our CG approach. We expose our online CG algorithm in Section 3.5. Our numerical evaluation is presented in Section 3.6. We conclude in Section 3.7.

3.2 Motivation and Challenges

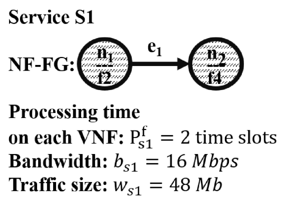
3.2.1 Problem Description

VNFs are software modules that run on VMs hosted on commodity hardware [9, 28, 56]. Thus, they require certain physical resources (i.e. CPU, RAM, storage, etc.) and have some processing capacity [28]. The problem of determining the number and the placement of VNFs in the physical network while satisfying their resource requirements is the VNF placement problem and is related to the VNE problem [29]. The VNF placement and the VNE problems have been widely studied in the literature [18, 25, 29, 32, 36] and are outside the scope of this contribution. In fact, studying the interplay that exists between the VNF placement problem along with the mapping, routing and scheduling problems is interesting. For instance, one may want to study the traffic pattern of a NS and deploy its NFs along its path, then perform the scheduling of its traffic. However, in real data centers (or TSP's network), VNFs may already be placed in the network and in use, or ready to be used by some NSs. Further, given the fact that VNFs are shared by many NSs, placing VNFs on the route of each NS upon its arrival may lead to over-provisioning and under-utilizing the network and computing resources. Hence, in the remaining of this chapter, we consider a physical network where the placement of VNFs is predetermined.

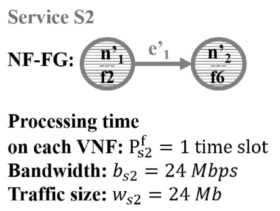
For the sake of simplicity, we assume that each VM is dedicated to exactly one VNF and that VNFs cannot share the same VM resources [38]. We also assume that VNFs are of unlimited buffer capacity. Given a NS request, requiring a certain amount of bandwidth to be guaranteed[2] for the transmission of traffic between VNFs; we seek at mapping, routing and scheduling NSs onto VNFs embedded in the network while respecting the ordering of NFs specified in the NF-FG demanded by each NS. That is, a NS needs to finish its processing on a VNF, so it can be transmitted to the following one along the chain. We assume that different NSs can share the same VNF but their traffic need to be processed at different time slots. In the remainder of this



3.1.a Physical network hosting different types of VNFs ($p_f = 25\ Mbps$; $c_l = 30\ Mbps$).


Service S1
NF-FG:
n_1
f2
e_1
n_2
f4
Processing time
on each VNF: P^f_{s1} = 2 time slots
Bandwidth: $b_{s1} = 16\ Mbps$
Traffic size: $w_{s1} = 48\ Mb$

3.1.b S1 network requirements.


Service S2
NF-FG:
n'_1
f2
e'_1
n'_2
f6
Processing time
on each VNF: P^f_{s2} = 1 time slot
Bandwidth: $b_{s2} = 24\ Mbps$
Traffic size: $w_{s2} = 24\ Mb$

3.1.c S2 network requirements.

Figure 3.1: Physical network topology and NSs used to illustrate the motivational examples.

section, we consider a physical network of 4 physical servers connected by physical

[2]A guaranteed bandwidth for the communication between VMs (hosting VNFs) implies a predictable performance of NSs [61, 62, 63, 64, 65].

links ($l \in L$) of uniform capacity $c_l = 30Mbps$. These servers are hosting VNFs of different types (i.e., $f1, ..., f7$) as shown in Fig.3.1.a. Here, $f1, ..., f7$ can represent a firewall, a proxy server, a cache, etc. The VNFs hosted in the network may have different processing capacities. The processing time P_s^f required to process the traffic of a NS s on a VNF f is calculated using Eq.(3.1) where w_s is the traffic size of NS s and p_f depicts the processing capacity of VNF f. For the sake of simplicity, we assume in our following motivational examples that all VNFs have the same processing capacity $p_f = 24Mbps$.

$$P_s^f = \frac{w_s}{p_f} \tag{3.1}$$

We explore the mapping, routing and scheduling of two NSs $S1$ and $S2$ having different computing and network requirements. Here, we consider that each NS requests a set of NFs of defined types in the form of a NF-FG. We represent the required NF-FG as a virtual network where the virtual nodes depicting the NFs are connected by virtual links. For instance, $S1$ (Fig.3.1.b) requests a chain of two NFs; $n1$ of type $f2$ and $n2$ of type $f4$ which are connected by a virtual link denoted by $e1$. The processing time of the traffic of $S1$ on each VNF is calculated based on Eq.(3.1). Similarly, the network requirements of NS $S2$ are depicted in Fig.3.1.c. Further, we assume that time is divided into slots (i.e., $t1, t2, t3$, etc.), each representing a duration of one second. In addition, we calculate the transmission delay D_s of the traffic of a NS s on each physical link on which it is routed by applying Eq.(3.2) where b_s depicts the bandwidth to be guaranteed for s and w_s is the traffic size of the NS as specified earlier.

$$D_s = \frac{w_s}{b_s} \tag{3.2}$$

3.2.2 Mapping, Routing and Scheduling Problems

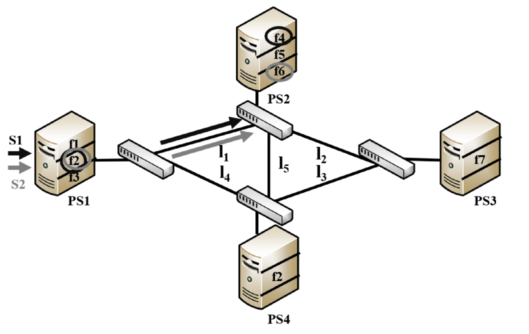
Satisfying a NS requires:

1. Mapping its NFs onto VMs hosting VNFs of the same type.

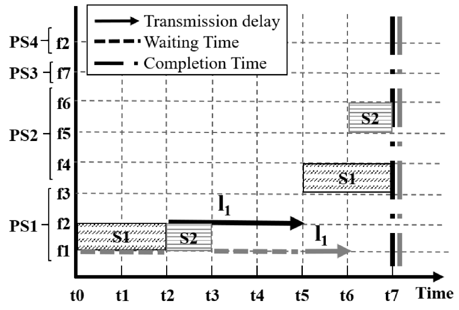
2. Routing the traffic between the VNFs on which its NFs are mapped through physical links able to guarantee the needed bandwidth while respecting the ordering of NFs depicted in the NF-FG.

3. Deciding on the processing schedule which entails determining the time slots at which the traffic of the NS gets processed on each of the VNFs.

These three pillars have an interplay where the decision/modification of one of them impacts another and affects not only the network resource utilization but the completion time of the NSs.



3.2.a S1 and S2 sharing VNFs and bandwidth.



3.2.b S1 and S2 schedules.

Figure 3.2: NF mapping and shortest path routing causing scheduling delays.

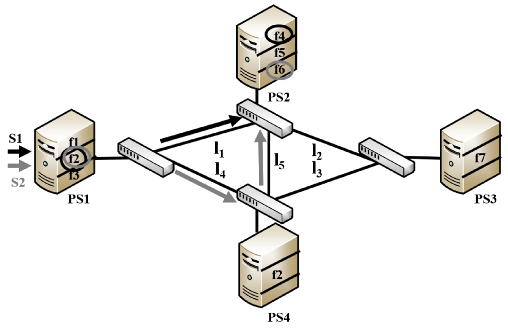
In order to illustrate this interplay, we consider the example in Fig.3.2.a where the NFs n_1 and n'_1 of NSs $S1$ and $S2$ respectively, are mapped to the same VNF $f2$ hosted on server $PS1$ at the same time slot $t0$. Given that a VNF can process the traffic of one NS at a time, $f2$ hosted on $PS1$ can start processing the traffic of either one of both NSs, $S1$ or $S2$. We assume that it starts by processing the traffic of $S1$ while the processing of the traffic of $S2$ on $f2$ is delayed for $P_{s1}^{f2} = 2$ *time slots* (Eq.(3.1)) (Fig.3.2.b). Once $f2$ finishes the processing of the traffic of $S1$, this latter can be transmitted to $f4$ on $PS2$ where the NF n_2 is mapped. Given that at $t2$, the link l_1

is not being used and has an available bandwidth $(c_l = 30Mbps) \geq (b_{s1} = 16Mbps)$, the virtual link e_1 of $S1$ can be mapped/routed through this shortest path (l_1). $D_{s1} = 3$ *time slots* (Eq.(3.2)) are needed for the transmission of all the traffic of $S1$ which arrives to $f4$ hosted on $PS2$ at $t5$ when its processing starts on this latter and lasts for $P_{s1}^{f4} = 2$ *time slots*. Hence, the completion time of $S1$ is $t7$. The traffic of $S2$ gets processed on $f2$ hosted on $PS1$ for one time slot $(t2)$ $(P_{s2}^{f2} = 1$ *time slot*$)$ (Fig.3.2.b). We choose the shortest path (link l_1) to transmit the traffic of $S2$ from $PS1$ to $PS2$ so it can get processed on $f6$ where n_2' is mapped. Even though the traffic of $S2$ is ready to be transmitted at $t3$, it has to wait for 2 time slots ($t3$ and $t4$) to be able to get routed through l_1 (Fig.3.2.b). The reason behind this is that the bandwidth $b_{s2} = 24Mbps$ required by $S2$ cannot be guaranteed on l_1 at time slots $t3$ and $t4$. In fact, the available bandwidth on l_1 at $t3$ and $t4$ is $bw_{l_1} = c_l - b_{s1} = 30 - 16 = 14Mbps$, since the traffic of $S1$ is being routed through l_1 at a guaranteed bandwidth $b_{s1} = 16Mbps$. Hence, $bw_{l_1} \leq b_{s2}$, which would delay the transmission of the traffic of $S2$ for 2 time slots until the bandwidth used by $S1$ on l_1 is released. Thus, the transmission of the traffic of $S2$ starts at $t5$ and lasts for one time slot before it gets processed by $f6$ hosted on $PS2$. The completion time of $S2$ is thus $t7$ (Fig.3.2.b).

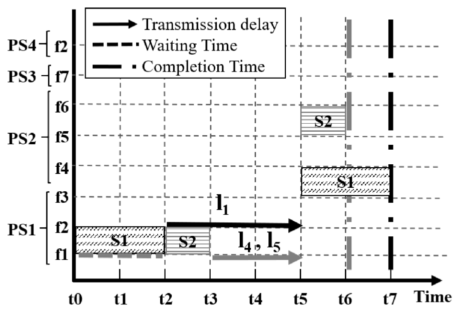
Here, we observe that $S1$ achieves its shortest schedule as it is using the shortest path route with no waiting delay for its traffic processing and transmission. However, we depict that the decisions of mapping the NFs n_1 and n_1' of $S1$ and $S2$ respectively to the same VNF $f2$ hosted on $PS1$ and routing their traffic through the same shortest path (link l_1) affected and delayed the schedule of $S2$ and under-utilized the network resources as we will explain in the next paragraphs.

3.2.3 Traffic Routing Impacts the NS Schedule

Choosing the shortest path to route the traffic of a NS was shown to be beneficial in terms of bandwidth and time consumption as presented in the previous example (Section 3.2.2) where the traffic of $S2$ needed to traverse one link only, consuming exactly $b_{s2} = 24Mbps$ and one time slot for its traffic transmission. However, this routing strategy introduced some waiting delays (2 time slots) given that it lacked sufficient bandwidth at the time slot at which the traffic of $S2$ was ready to be transmitted. Such delays impacted and delayed the completion time of $S2$.



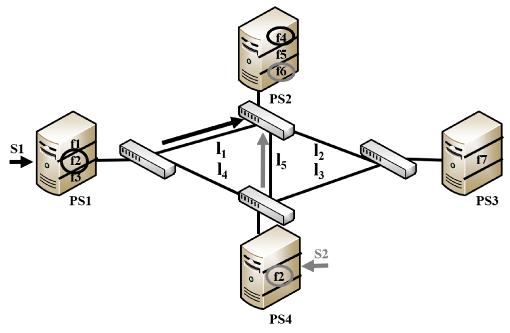
3.3.a S1 and S2 using different routes.



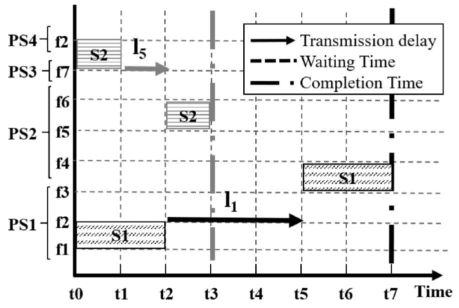
3.3.b S1 and S2 schedules.

Figure 3.3: Impact of traffic routing on the NSs schedules.

Hence, we consider in Fig.3.3.a the same mapping of NFs for both NSs $S1$ and $S2$, but we modify the routing of the traffic of $S2$. Thus, instead of mapping the virtual link e'_1 (Fig.3.1.c) to link l_1, we map it to links l_4 and l_5. Since l_4 and l_5 are not used by any other NSs, they have enough capacity ($c_l = 30Mbps$) to guarantee the bandwidth $b_{s2} = 24Mbps$ required by $S2$. Thus, the traffic of $S2$ can be transmitted at $t3$ as soon as its processing on $f2$ (hosted on $PS1$) is completed. In this case, no waiting delays are incurred (as in Fig.3.2), but the transmission delay increased by one time slot. In fact, the traffic of $S2$ need to traverse 2 links (l_4 and l_5), incurring a transmission delay equal $2 \times D_{s2} = 2 \times 1 = 2$ *time slots*. This, in fact, decreases the waiting delays caused by the chosen routing path from 2 time slots to 0 time slot



3.4.a S1 and S2 using different VNFs.



3.4.b S1 and S2 schedules.

Figure 3.4: Impact of NF mapping on the NSs schedules.

while increasing the transmission time from 1 time slot to 2 time slots which results in a reduction of the NS $S2$ completion time by 1 time slot. $S2$ completes its processing at $t6$ (Fig.3.3.b) instead of $t7$ (Fig.3.2.b).

We conclude that the path chosen to route the traffic of a NS has a reciprocal impact on its schedule and the network resource utilization. While a shortest path consumes less bandwidth, a longer route may better utilize idle network resources, reduce the waiting delays and hence, contributes to an earlier completion time of the NS in question.

3.2.4 NF Mapping Impacts the NS Schedule

In the previous examples (Fig.3.2 and Fig.3.3), $S1$ and $S2$ were sharing the same VNF $f2$ (hosted on $PS1$) which delayed the processing of the traffic of $S2$ that had to wait for $f2$ until it finishes the processing of the traffic of $S1$. To overcome the waiting delays, one can map the NF n'_1 of $S2$ to another free VNF of the same type $f2$ hosted in the network; that is, VNF $f2$ hosted on $PS4$ (Fig.3.4.a). This, not only changes the mapping of n'_1 but also the routing path of the traffic of $S2$ which needs to be transmitted from $PS4$ to $PS2$ where n'_2 is mapped to $f6$. Hence, we choose to route the traffic of $S2$ in this case through l_5 that represents the shortest path between

$PS4$ and $PS2$ and able to guarantee the bandwidth $b_{s2} = 24Mbps$ required by $S2$. Given this mapping and routing, the traffic of $S2$ can start processing on $f2$ hosted on $PS4$ at $t0$ for $P_{s2}^{f2} = 1$ *time slot*, starts transmission at $t1$ for $D_{s2} = 1$ *time slot*, then gets processed by $f6$ hosted on $PS2$ at $t2$ (Fig.3.4.b). Thus, with this mapping and routing, $S2$ achieves its shortest schedule that gets completed at $t3$.

A different mapping of the NFs of $S2$ allowed better utilization of idle network resources ($f2$ hosted on $PS4$ and l_5) and decreased the completion time of $S2$. This highlights the interplay that exists between NF mapping and NS scheduling. Further, it is important to note that reducing the completion time of a NS is in the best interest of network operators which will have their network resources freed sooner, and thus can be reused by other NSs. Hence, they can admit and process more requests.

3.3 SFCS - A Mixed Integer Linear Program (SFCS-MILP)

Solving the SFCS problem entails considering collectively the NF mapping, traffic routing, and NS scheduling problems due to the interplay that exists between them (Section 3.2). In the following, we define and present a mathematical formulation for the SFCS problem.

3.3.1 Problem Definition

Let $G(K, L)$ be a physical network of set K of nodes (i.e, servers, switches, etc.) hosting VNFs and a set L of physical links connecting them. Given a set S of NSs, each NS requests its traffic to be processed by a chain of NFs in a defined order and transmitted from one function to another at a guaranteed bandwidth. Satisfying these NSs requires providing an efficient NF mapping, traffic routing and NS scheduling of

Physical network inputs	
$G(K, L)$	Physical network of set K of nodes and a set L of physical links connecting them.
F	Set of VNF instances running on VMs hosted on the physical servers $k \in K_p$.
T	Set of types of all VNFs $f \in F$.
$t_f \in \mathbb{Z}^+$	Type of a VNF instance $f \in F$ ($t_f \in T$)
$c_{ij} \in \mathbb{Z}^+$	Capacity of a physical link $(ij) \in L$.
$x_f^k \in \{0, 1\}$	VNF instance $f \in F$ is hosted on the physical server $k \in K_p$ (1) or not (0).

NS inputs	
S	Set of NS requests.
$H_s(N_s, E_s)$	A directed graph representing the NF-FG of a NS $s \in S$.
$b_s \in \mathbb{Z}^+$	Bandwidth demanded by NS $s \in S$.
$w_s \in \mathbb{Z}^+$	Traffic demand of NS $s \in S$.
$m_{ns} \in \mathbb{Z}^+$	Type of a NF $n \in N_s$ of NS $s \in S$.
$p_{ns} \in \mathbb{Z}^+$	Processing time of the traffic of NS $s \in S$ on the NF $n \in N_s$.
Δ	Set of time slots $\delta \in \Delta$.

Table 3.1: Parameters of the SFCS-MILP.

the traffic of each of those NSs. The most beneficial schedule is the one that satisfies all the NSs while minimizing their total schedule length. Hence, we define the SFCS problem as follows:

Definition 3.1. *Given a physical network $G(K, L)$ hosting and running different types of VNFs, a set S of NS requests, each demanding the processing of its traffic by a chain of NFs as specified in its NF-FG; find the optimal mapping, routing and scheduling of the traffic of these NSs such that their total schedule length is minimized.*

3.3.2 Problem Formulation

Table 3.1 delineates the parameters used in the formulation of the SFCS-MILP problem presented below. We define the decision variable $\varrho \in \mathbb{N}^+$ to represent the schedule length that depicts the time needed to complete the processing of the traffic of all

Decision variables of the SFCS-MILP	
$\varrho \in \mathbb{N}^+$	Schedule length of all the NSs.
$y_{ns}^{f\delta} \in \{0,1\}$	Specifies that the traffic of NS s started processing at time slot δ on VNF $f \in F$ to which NF n is mapped (1) and (0) otherwise.
$z_{ns}^{f\delta} \in \{0,1\}$	Specifies that the traffic of NS s is processing at time slot δ on VNF $f \in F$ to which NF n is mapped (1) and (0) otherwise.
$q_{ns}^{k} \in \{0,1\}$	Determines that NF n of NS $s \in S$ is mapped to a VNF hosted on server k (1) and (0) otherwise.
$h_{ns}^{k} \in \{0,1\}$	Indicates that NFs $n,(n+1)$ of NS $s \in S$ are mapped to VNFs hosted on the same server k (1) and (0) otherwise.
$\theta_{s}^{\delta e} \in \{0,1\}$	Designates that a NF $o(e) \in N_s$ of NS s begins the transmission of the traffic to its successor NF $d(e)$ at time slot $\delta \in \Delta$ on the virtual link e (1) and (0) otherwise.
$\hat{\theta}_{s}^{\delta e} \in \{0,1\}$	Indicates that the virtual link e is being used for traffic transmission between the NFs $o(e), d(e)$ of NS s at time slot δ (1) and (0) otherwise.
$l_{ij}^{e} \in \{0,1\}$	Denotes that the virtual link e of NS s is routed through the link $(ij) \in L$ (1) and (0) otherwise.

Table 3.2: Decision variables of the SFCS-MILP.

the NSs. Our objective (Eq.3.3) seeks at minimizing the schedule length needed to process all the NSs.

$$\text{Minimize} \quad \varrho \tag{3.3}$$

This objective is subject to several constraints. Thus, we define a new decision variable $y_{ns}^{f\delta} \in \{0,1\}$ which specifies that the traffic of NS $s \in S$ started processing at time slot $\delta \in \Delta$ on VNF $f \in F$ to which NF $n \in N_s$ is mapped.

$$y_{ns}^{f\delta} = \begin{cases} 1 \text{ if traffic of NS } s \text{ started processing at } \delta \text{ on VNF } f \text{ to which NF } n \text{ is mapped,} \\ 0 \text{ otherwise.} \end{cases}$$

We introduce the decision variable $z_{ns}^{f\delta} \in \{0,1\}$ to specify that the traffic of NS $s \in S$ is processing at time slot $\delta \in \Delta$ on VNF $f \in F$ to which NF $n \in N_s$ is mapped.

$$z_{ns}^{f\delta} = \begin{cases} 1 \text{ if traffic of NS } s \text{ is processing at } \delta \text{ on VNF } f \text{ to which NF } n \text{ is mapped,} \\ 0 \text{ otherwise.} \end{cases}$$

$q_{ns}^{k} \in \{0, 1\}$ is a new decision variable which depicts that the NF $n \in N_s$ of NS $s \in S$ is mapped to a VNF instance hosted on a physical server $k \in K_p$ (1) (or not, 0).

$$q_{ns}^{k} = \begin{cases} 1 \text{ if NF } n \text{ is mapped to a VNF hosted on server } k, \\ 0 \text{ otherwise.} \end{cases}$$

Further, we declare the $h_{ns}^{k} \in \{0, 1\}$ as a new decision variable to indicate that NFs $n, (n + 1) \in N_s$ of NS $s \in S$ are mapped to VNFs hosted on the same physical server $k \in K_p$ (1) (or not, 0).

$$h_{ns}^{k} = \begin{cases} 1 \text{ if NFs } n, (n + 1) \text{ of NS } s \text{ are mapped to VNFs hosted on the same server } k, \\ 0 \text{ otherwise.} \end{cases}$$

We also define the variable $\theta_s^{\delta e} \in \{0, 1\}$ to designate that a NF $o(e) \in N_s$ of NS $s \in S$ begins the transmission of the traffic to its successor NF $d(e) \in N_s$ at time slot $\delta \in \Delta$ on the virtual link $e \in E_s$ (1) (or not, 0).

$$\theta_s^{\delta e} = \begin{cases} 1 \text{ if NF } o(e) \text{ of NS } s \text{ started the transmission of the traffic to NF } d(e) \text{ at time} \\ \text{ slot } \delta \text{ on virtual link } e, \\ 0 \text{ otherwise.} \end{cases}$$

The variable $\hat{\theta}_s^{\delta e} \in \{0, 1\}$ indicates that the virtual link $e \in E_s$ is used for traffic transmission between the NFs $o(e), d(e) \in N_s$ of NS $s \in S$ at time slot $\delta \in \Delta$ (1) (or not, 0).

$$\hat{\theta}_s^{\delta e} = \begin{cases} 1 \text{ if NF } o(e) \text{ of NS } s \text{ is transmitting the traffic to NF } d(e) \text{ at time slot } \delta \\ \text{on virtual link } e, \\ 0 \text{ otherwise.} \end{cases}$$

We declare $l_{ij}^e \in \{0,1\}$ to denote that the virtual link $e \in E_s$ of NS $s \in S$ is routed through the physical link $(ij) \in L$ (1) (or not, (0)).

$$l_{ij}^e = \begin{cases} 1 \text{ if virtual link } e \text{ is routed through physical link } (ij), \\ 0 \text{ otherwise.} \end{cases}$$

1. *NF mapping constraints*

   We first start by determining the NF mapping constraints. Thus, we define Eq.(3.4) to ensure that each NF n of NS s should be mapped to exactly one VNF instance f.

$$\sum_{f \in F} \sum_{\delta \in \Delta} y_{ns}^{f\delta} = 1 \quad \substack{\forall n \in N_s \\ \forall s \in S} \tag{3.4}$$

   Eq.(3.5) guarantees that a NF n of NS s is mapped to a VNF instance f of the same type.

$$\sum_{f \in F} \sum_{\delta \in \Delta} y_{ns}^{f\delta} t_f = m_{ns} \quad \substack{\forall n \in N_s \\ \forall s \in S} \tag{3.5}$$

2. *NS scheduling constraints*

   To ensure a correct and feasible scheduling of NSs, we prevent in Eq.(3.6) the transmission of the traffic of a NS s between two consecutive NFs $o(e)$ and $d(e)$

if its processing on $o(e)$ has not been completed.

$$\theta_s^{\delta' e} \leq 1 - \sum_{f \in F} y_{o(e)s}^{f\delta} \quad \substack{\forall \delta, \delta' \in \Delta;\ \delta' < \delta + p_{o(e)s} \\ \forall e \in E_s \\ \forall s \in S} \tag{3.6}$$

We define Eq.(3.7) to ensure that the traffic of a NS s cannot be processed by a NF $d(e)$ if it was not transmitted to it by its predecessor NF $o(e)$.

$$\sum_{f \in F} y_{d(e)s}^{f\delta'} \leq 1 - \theta_s^{\delta e} \quad \substack{\forall \delta, \delta' \in \Delta;\ \delta' < \delta + \frac{w_s}{b_s} \\ \forall e \in E_s \\ \forall s \in S} \tag{3.7}$$

Further, Eq.(3.8) prevents a NF $(n+1)$ to start processing the traffic of a NS s before its predecessor NF n finishes its execution.

$$\sum_{f \in F} y_{(n+1)s}^{f\delta'} \leq 1 - \sum_{f \in F} y_{ns}^{f\delta} \quad \substack{\forall \delta, \delta' \in \Delta;\ \delta' < \delta + p_{ns} \\ \forall n, (n+1) \in N_s \\ \forall s \in S} \tag{3.8}$$

Eq.(3.9) guarantees that VNF f processes the traffic of NS s during all the processing period.

$$\sum_{\delta \in \Delta} z_{ns}^{f\delta} = p_{ns} \sum_{\delta \in \Delta} y_{ns}^{f\delta} \quad \substack{\forall n \in N_s \\ \forall s \in S \\ \forall f \in F} \tag{3.9}$$

Eq.(3.10) sets the processing period of the traffic of NS s on VNF f. That is, Eq.(3.10) ensures the consecutive processing of the traffic of NS s by VNF f (prevents preemption).

$$z_{ns}^{f\delta'} \geq y_{ns}^{f\delta} \quad \substack{\forall \delta, \delta' \in \Delta;\ \delta \leq \delta' < \delta + p_{ns} \\ \forall n \in N_s \\ \forall s \in S \\ \forall f \in F} \tag{3.10}$$

We define Eq.(3.11) to ensure that a VNF f cannot process the traffic of more than one NS at a certain time slot δ.

$$\sum_{s \in S} \sum_{n \in N_s : m_{ns} = t_f} z_{ns}^{f\delta} \leq 1 \quad \substack{\forall \delta \in \Delta \\ \forall f \in F} \tag{3.11}$$

To determine the schedule length, we define Eq.(3.12) to ensure that the latter is greater or equal to the completion time of each of the NSs.

$$\varrho \geq \sum_{f \in F} \sum_{\delta \in \Delta} y_{|N_s|s}^{f\delta} (\delta + p_{|N_s|s}) \quad \forall s \in S \tag{3.12}$$

Eq.(3.13) determines the physical server k hosting the VNF on which a NF n of NS s is mapped.

$$q_{ns}^{k} = \sum_{f \in F} \sum_{\delta \in \Delta} y_{ns}^{f\delta} x_f^k \quad \substack{\forall n \in N_s \\ \forall s \in S \\ \forall k \in K_p} \tag{3.13}$$

To be able to track if two consecutive NFs n and $(n+1)$ of a NS s are mapped to VNFs hosted on the same physical server k, we define Eq.(3.14).

$$h_{ns}^{k} = q_{ns}^{k} q_{(n+1)s}^{k} \quad \substack{\forall k \in K_p \\ \forall n,(n+1) \in N_s \\ \forall s \in S} \tag{3.14}$$

Eq.(3.15) prevents the start of the transmission of traffic of a NS s between two consecutive NFs $o(e)$ and $d(e)$ if they are mapped to VNFs hosted on the same physical server.

$$\sum_{\delta \in \Delta} \theta_s^{\delta e} = 1 - \sum_{k \in K_p} h_{o(e)s}^{k} \quad \substack{\forall e \in E_s \\ \forall s \in S} \tag{3.15}$$

Eq.(3.16) specifies that the virtual link e is used to transmit traffic of NS s during all the required transmission time ($\frac{w_s}{b_s}$) between NFs o(e) and d(e) only if the transmission is required (i.e, NFs o(e) and d(e) are mapped to VNFs hosted on different physical servers).

$$\sum_{\delta \in \Delta} \hat{\theta}_s^{\delta e} = \frac{w_s}{b_s} \sum_{\delta \in \Delta} \theta_s^{\delta e} \quad \substack{\forall e \in E_s \\ \forall s \in S} \tag{3.16}$$

Eq.(3.17) ensures that the virtual link e is occupied throughout the transmission period ($\frac{w_s}{b_s}$) starting at time slot δ (when the transmission begins).

$$\sum_{\delta' \in [\delta,\ \delta + \frac{w_s}{b_s} - 1]} \hat{\theta}_s^{\delta' e} \geq \frac{w_s}{b_s} \theta_s^{\delta e} \quad \substack{\forall e \in E_s \\ \forall s \in S \\ \forall \delta \in \Delta} \tag{3.17}$$

3. *Traffic routing constraints*

To solve the traffic routing problem, we dictate Eq.(3.18) to guarantee that the physical links capacity is not violated.

$$\sum_{s \in S} \sum_{e \in E_s} l_{ij}^e \hat{\theta}_s^{\delta e} b_s \leq c_{ij} \quad \substack{\forall \delta \in \Delta \\ \forall (ij) \in L} \tag{3.18}$$

Specifying the route of the traffic of a NS s is done through Eq.(3.19) which represents the flow conservation constraints. Eq.(3.19) determines the physical links ($(ij) \in L$) of the network through which the virtual links ($e \in E_s$) of a NS s are routed.

$$\sum_{j:(i,j) \in L} l_{ij}^e - \sum_{j:(j,i) \in L} l_{ji}^e = q_{o(e)s}^i - q_{d(e)s}^i \quad \substack{\forall e \in E_s \\ \forall s \in S \\ \forall i \in K_p} \tag{3.19}$$

Finally, Eq.(3.20) prevents the routing of a virtual link e of a NS s through a physical link $(ij) \in L$ if $o(e)$ and $d(e)$ are mapped to VNFs hosted on the same physical server.

$$l_{ij}^e \leq 1 - \sum_{k \in K_p} h_{o(e)s}^k \quad \substack{\forall e \in E_s \\ \forall s \in S \\ \forall (ij) \in L} \tag{3.20}$$

It is worth noting that Eq.(3.14) and Eq.(3.18) are non linear and can be easily linearized as explained in Appendix B. The SFCS is a MILP that we denote by *SFCS-MILP* which is complex to solve. Next, we discuss the complexity of the SFCS problem.

3.3.3 Problem Complexity

The SFCS-MILP formulation is complex and the model is clearly hard to solve even for a small network (Section 3.6). The complexity of the SFCS problem can be highlighted through the complexity of the different problems it solves. In fact, each of the NF mapping, the traffic routing and the NS scheduling problems has been proven as NP-Hard (Section 2.3). Hence, the SFCS problem is NP-Hard and is of combinatorial nature. Due to its large solutions space, we present a CG approach to solve it.

3.4 SFCS - A Column Generation Approach

Column generation is a very powerful method used for solving large-scale optimization problems [58, 60]. Based on a primal-dual decomposition technique, CG solves very large linear programs by considering a small subset of the variables at once. The principle of the decomposition lies on two separate problems; the *Master problem* that operates on a set of general constraints and the *Pricing sub-problem* which considers a specific set of constraints [59]. Both problems keep on exchanging information until an optimal criteria (or a stopping condition) is met. More precisely, the MP is initialized with a subset of feasible solutions that we refer to by “columns” and which satisfy all of its constraints. When solved, the MP passes a revised set of cost coefficients (or prices) associated with its dual optimal solution to the pricing SP. These prices are used by the pricing SP to generate a new column that improves the solution of the MP. Only few columns are needed by the MP to obtain the LP optimal solution of the problem, which distinguishes the CG technique from an exhaustive enumeration approach that may be computationally impossible [57]. The LP optimal solution obtained by CG is a lower bound for the ILP optimal solution of the initial problem (without a decomposition).

3.4.1 Decomposition Strategy

To apply the CG technique on the SFCS problem, we need to design an efficient decomposition of the problem. Thus, we decompose it into a MP and multiple pricing SPs. Each pricing SP solves the SFCS problem for a single NS. More precisely, it constructs a column per NS holding the NF mapping, the traffic routing and the NS scheduling decisions with respect to the ordering of NFs in the NF-FG of the NS and to the links capacity constraints. At each iteration, the pricing SP of each NS will generate a column and sends it to the MP. The MP will select the best combination of columns from those gathered by the pricing SPs. Such combination is composed by one column per NS and minimizes the total schedule length of all the NSs.

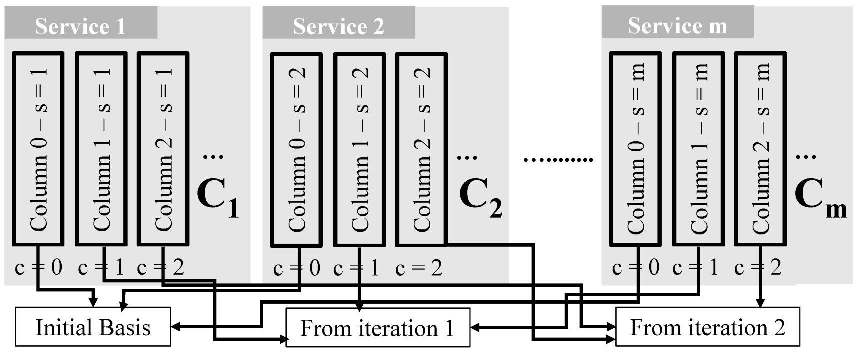


Figure 3.5: Representation of the columns passed to the MP.

Fig.3.5 details the master columns representation where $s = 1, s = 2, .., s = m \in S$ depict the NSs. Every NS can have many columns where each is generated by its corresponding pricing SP at each iteration. We represent by $C = \cup_{s=1}^{m} C_s$ the set of all the columns sent to the MP. C_s depicts the set of columns generated for NS s. The subset of columns selected by the MP for all NSs should ensure that the resource utilization constraints are respected; that is, guaranteeing that each VNF is processing the traffic of at most a single NS at a time slot and that the links capacity

MP inputs	
$r_{fs}^{\delta c} \in \{0,1\}$	Specifies that VNF f is processing the traffic of NS s of a column $c \in C$ at time slot δ (1) and (0) otherwise.
$o_{(ij)s}^{\delta c} \in \mathbb{N}^{+}$	Bandwidth used by NS s in column $c \in C$ on the physical link $(ij) \in L$ at time slot δ.
$v_s^c \in \mathbb{N}^{+}$	Completion time of NS s in column $c \in C$.
$c_{ij} \in\in \mathbb{Z}^{+}$	Capacity of physical link $(ij) \in L$ (as in Section 3.3).

Table 3.3: Parameters of the MP.

Decision variables of the MP	
$\varrho \in \mathbb{N}^{+}$	Minimum schedule length needed to process all the NSs $s \in S$.
$\lambda_s^c \in [0-1]$	Indicates that column $c \in C$ of NS $s \in S$ is selected or not.

Table 3.4: Decision variables of the MP.

constraints are not violated. The MP and the SP formulations are discussed in the following.

3.4.2 Master Problem

Table 3.3 delineates the parameters of the MP which can be formulated as explained in the following. The objective (Eq.(3.21)) of the MP consists of minimizing the schedule length needed to process all the NSs which is represented by the decision variable $\varrho \in \mathbb{N}^{+}$ to represent the minimum schedule length needed to process all the NSs $s \in S$.

$$Minimize \quad \varrho \tag{3.21}$$

The objective of the MP is subject to several constraints. Thus, we define $\lambda_s^c \in \{0,1\}$ as a new decision variable which indicates that column $c \in C$ of NS s is selected (1) and (0) otherwise. Given that the MP should be a LP, the integrality of its decision variable λ_s^c should be relaxed (i.e, $\lambda_s^c \in [0-1]$). The decision variables of the MP are summarized in Table 3.4. We define Eq.(3.22) to ensure that one column for each NS is selected.

$$\alpha_s : \sum_{c\in C} \lambda_s^c \geq 1 \quad \forall s \in S \tag{3.22}$$

Eq.(3.23) guarantees that a VNF cannot process the traffic of more than one NS at each time slot.

$$\beta_f^\delta : \sum_{c\in C}\sum_{s\in S} r_{fs}^{\delta c}\lambda_s^c \leq 1 \quad \substack{\forall\delta\in\Delta \\ \forall f\in F} \tag{3.23}$$

We assert that the links capacity is not violated through Eq.(3.24).

$$\gamma_{ij}^\delta : \sum_{c\in C}\sum_{s\in S} o_{(ij)s}^{\delta c}\lambda_s^c \leq c_{ij} \quad \substack{\forall\delta\in\Delta \\ \forall (ij)\in L} \tag{3.24}$$

We declare Eq.(3.25) to specify that the schedule length needed to process all the NSs is greater or equal to the completion time of each of them.

$$\phi_s^c : \varrho \geq \sum_{c\in C} v_s^c \lambda_s^c \quad \forall s \in S \tag{3.25}$$

α_s, β_f^δ, γ_{ij}^δ, ϕ_s^c are the dual variables associated with Eqs.(3.22), (3.23), (3.24), (3.25) respectively.

3.4.3 Pricing Sub-Problem

Since the pricing SP solves the SFCS problem for a single NS, its formulation is similar to the SFCS-MILP (Section 3.3). Hence, almost the same parameters and decision variables used in the SFCS-MILP are used in the pricing SP, except that we remove the subscript s (representing a NS $s \in S$) from their abbreviations given that each pricing SP is related to a single NS.

The dual variables α_s, β_f^δ, γ_{ij}^δ, ϕ_s^c associated with the constraints of the MP are as explained in Section (3.4.2). These variables represent the cost coefficients that will guide the pricing SP to generate a column for one NS which will improve the solution

Decision variables of the pricing SP	
$r_f^\delta \in \{0,1\}$	specifies that VNF f is processing the traffic of NS s at time slot δ (1) and (0) otherwise.
$o_{ij}^\delta \in \mathbb{Z}^+$	Bandwidth used by NS $s \in S$ on the physical link (ij) at time slot δ.
$v \in \mathbb{N}^+$	Completion time of NS s.
$y_n^{f\delta}$, $z_n^{f\delta}$, h_n^k, q_n^k, $\theta^{\delta e}$, $\hat{\theta}^{\delta e}$, l_{ij}^e	Refer to Table 3.2.

Table 3.5: Decision variables of the pricing SP.

of the MP. The remaining parameters are listed in Table 3.1.

We define the decision variable $r_f^\delta \in \{0,1\}$ to specify that VNF $f \in F$ is processing the traffic of NS $s \in S$ at time slot $\delta \in \Delta$ (1) and (0) otherwise.

$$r_f^\delta = \begin{cases} 1 \text{ if VNF is processing the traffic of NS } s \text{ at time slot } \delta, \\ 0 \text{ otherwise.} \end{cases}$$

We declare as well the decision variable $o_{ij}^\delta \in \mathbb{Z}^+$ to represent the amount of bandwidth used by NS $s \in S$ on the physical link $(ij) \in L$ at time slot $\delta \in \Delta$. $v \in \mathbb{N}^+$ is also a new defined decision variable that delineates the completion time of NS $s \in S$. Other decision variables such as $y_n^{f\delta}$, $z_n^{f\delta}$, h_n^k, q_n^k, $\theta^{\delta e}$, $\hat{\theta}^{\delta e}$, l_{ij}^e are as defined in Table 3.2. Table 3.5 summarizes the SP decision variables.

The objective of the pricing SP is to minimize the Reduced Cost (RC) depicted in Eq.(3.26). Based on the dual variables α, β_f^δ, γ_{ij}^δ and ϕ representing the prices provided by the master model, the RC will guide the pricing to generate a new column that will minimize the completion time v of the NS by ϕ while using the available resources (VNFs and bandwidth available at certain time slots) in the network depicted by the values of β_f^δ, γ_{ij}^δ.

$$RC = -\,\alpha - \sum_{\delta\in\Delta}\sum_{f\in F} r_f^\delta \beta_f^\delta - \sum_{\delta\in\Delta}\sum_{(ij)\in L} o_{ij}^\delta \gamma_{ij}^\delta + \phi v \tag{3.26}$$

This objective is subject to several constraints. We start by introducing the NF mapping constraints.

1. *NF mapping constraints*

   A NF n of a NS s should be mapped to exactly one VNF instance f as specified in Eq.(3.27).

   $$\sum_{f\in F}\sum_{\delta\in\Delta} y_n^{f\delta} = 1 \quad \forall n \in N \tag{3.27}$$

   Further, a NF n of a NS s should be mapped to a VNF instance f of the same type Eq.(3.28).

   $$\sum_{f\in F}\sum_{\delta\in\Delta} y_n^{f\delta} t_f = m_n \quad \forall n \in N \tag{3.28}$$

2. *NS scheduling constraints*

   A valid NS scheduling entails preventing the transmission of the traffic between two consecutive NFs, $o(e)$ and $d(e)$, if its processing on $o(e)$ has not been completed (Eq.(3.29)).

   $$\theta^{\delta' e} \leq 1 - \sum_{f\in F} y_{o(e)}^{f\delta} \quad {\scriptstyle \substack{\forall \delta,\delta'\in\Delta;\ \delta'<\delta+p_{o(e)} \\ \forall e\in E}} \tag{3.29}$$

   In addition, the traffic cannot be processed by a NF $d(e)$ if it was not transmitted to it by its predecessor NF $o(e)$ (Eq.(3.30)).

   $$\sum_{f\in F} y_{d(e)}^{f\delta'} \leq 1 - \theta^{\delta e} \quad {\scriptstyle \substack{\forall \delta,\delta'\in\Delta;\ \delta'<\delta+\frac{w}{b} \\ \forall e\in E}} \tag{3.30}$$

   We defined Eq.(3.31) to prevent a NF $(n+1)$ to start processing the traffic of the NS before its predecessor NF n finishes its execution.

$$\sum_{f\in F} y_{(n+1)}^{f\delta'} \leq 1 - \sum_{f\in F} y_n^{f\delta} \quad {\scriptstyle \substack{\forall \delta,\delta'\in\Delta;\ \delta'<\delta+p_n \\ \forall n,(n+1)\in N}} \tag{3.31}$$

Eq.(3.32) ensures that the VNF f is processing the traffic of the NS during all the processing period.

$$\sum_{\delta\in\Delta} z_n^{f\delta} = p_n \sum_{\delta\in\Delta} y_n^{f\delta} \quad {\scriptstyle \substack{\forall n\in N \\ \forall f\in F}} \tag{3.32}$$

To determine the processing period of the traffic of a NS on a VNF f, we define Eq.(3.33).

$$z_n^{f\delta'} \geq y_n^{f\delta} \quad {\scriptstyle \substack{\forall \delta,\delta'\in\Delta;\ \delta\leq\delta'<\delta+p_n \\ \forall n\in N \\ \forall f\in F}} \tag{3.33}$$

Eq.(3.34) helps in determining the physical server k hosting the VNF to which a NF n is mapped.

$$q_n^k = \sum_{f\in F}\sum_{\delta\in\Delta} y_n^{f\delta} x_f^k \quad {\scriptstyle \substack{\forall n\in N \\ \forall k\in K_p}} \tag{3.34}$$

We add Eq.(3.35) to specify if two consecutive NFs n and $(n+1)$ are mapped to VNFs hosted on the same physical server $k \in K_p$.

$$h_n^k = q_n^k q_{(n+1)}^k \quad {\scriptstyle \substack{\forall k\in K_p \\ \forall n,(n+1)\in N}} \tag{3.35}$$

Eq.(3.36) prevents the start of the transmission of the traffic between two consecutive NFs $o(e)$ and $d(e)$ if they are mapped to VNFs hosted on the same physical server.

$$\sum_{\delta\in\Delta} \theta^{\delta e} = 1 - \sum_{k\in K_p} h_{o(e)}^k \quad \forall e \in E \tag{3.36}$$

We use Eq.(3.37) to specify that the virtual link e is used to transmit the traffic during all the required transmission time ($\frac{w}{b}$) between NFs $o(e)$ and $d(e)$ if the transmission is possible (i.e, NFs $o(e)$ and $d(e)$ are mapped to VNFs hosted on different physical servers).

$$\sum_{\delta \in \Delta} \hat{\theta}^{\delta e} = \frac{w}{b} \sum_{\delta \in \Delta} \theta^{\delta e} \quad \forall e \in E \tag{3.37}$$

Eq.(3.38) ensures that the virtual link e is occupied throughout the transmission period ($\frac{w}{b}$) starting at time slot δ (when the transmission begins).

$$\sum_{\delta' \in [\delta, \delta + \frac{w}{b} - 1]} \hat{\theta}^{\delta' e} \geq \frac{w}{b} \theta^{\delta e} \quad {\scriptstyle \substack{\forall e \in E \\ \forall \delta \in \Delta}} \tag{3.38}$$

3. *Traffic routing constraints*

For a valid traffic routing, we need to guarantee that the physical links capacity is not violated (Eq.(3.39)).

$$o_{ij}^{\delta} \leq c_{ij} \quad {\scriptstyle \substack{\forall \delta \in \Delta \\ \forall (ij) \in L}} \tag{3.39}$$

To specify the route of the traffic of a NS, we add the flow conservation constraint which determines the physical links ($(ij) \in L$) of the network through which the virtual link $e \in E$ is routed (Eq.(3.40)).

$$\sum_{j:(i,j) \in L} l_{ij}^{e} - \sum_{j:(j,i) \in L} l_{ji}^{e} = q_{o(e)}^{i} - q_{d(e)}^{i} \quad {\scriptstyle \substack{\forall e \in E \\ \forall i \in K_p}} \tag{3.40}$$

Eq.(3.41) prevents the routing of a virtual link e through a physical link $(ij) \in L$ if $o(e)$ and $d(e)$ are hosted on the same physical server.

$$l_{ij}^{e} \leq 1 - \sum_{k \in K_p} h_{o(e)}^{k} \quad {\scriptstyle \substack{\forall e \in E \\ \forall (ij) \in L}} \tag{3.41}$$

Finally, it is worth noting that the values of v, o_{ij}^{δ} and r_{f}^{δ} are represented by

Eqs.(3.42), (3.43) and (3.44) respectively.

$$v = \sum_{f \in F} \sum_{\delta \in \Delta} y^{f\delta}_{|N_s|}(\delta + p_{|N|}) \tag{3.42}$$

$$o^{\delta}_{ij} = \sum_{e \in E} l^{e}_{ij} \hat{\theta}^{\delta e} b \quad {\scriptstyle \forall (ij) \in L \atop \forall \delta \in \Delta} \tag{3.43}$$

$$r^{\delta}_{f} = \sum_{n \in N : m_n = t_f} z^{f\delta}_{n} \quad {\scriptstyle \forall f \in F \atop \forall \delta \in \Delta} \tag{3.44}$$

Eq.(3.35) and Eq.(3.43) are non linear and can be linearized as explained for Eq.(3.14) and Eq.(3.18) respectively in Appendix B.

3.4.4 Column Generation Algorithm (SFCS-CG)

The CG technique warm starts the LP MP by initializing it with a basic feasible solution represented by a set of columns constituting the initial basis. In order to generate this solution, we develop a heuristic that loops over all the NSs, map the NFs of each of them to VNFs chosen randomly from the set of VNFs of the same type existing in the network, route each of their traffic through the shortest path connecting each pair of VNFs to which the NFs are mapped and finally schedule the NSs sequentially starting from time slot 0. More precisely, the processing of the traffic of each NS starts when the traffic of its previous one has completed its processing through the requested NF-FG.

Using this initial basis, the LP MP is run to optimality. Its dual values are then sent to the pricing SP of each NS where they are used to generate a column holding the NF mapping, the traffic route and the schedule of the NS it represents. Here, we benefit from the independency that exists between the pricing SPs by performing their parallel processing using threads. Thus, each pricing SP will be processed by a thread which will provide gains in the total execution time of our CG approach.

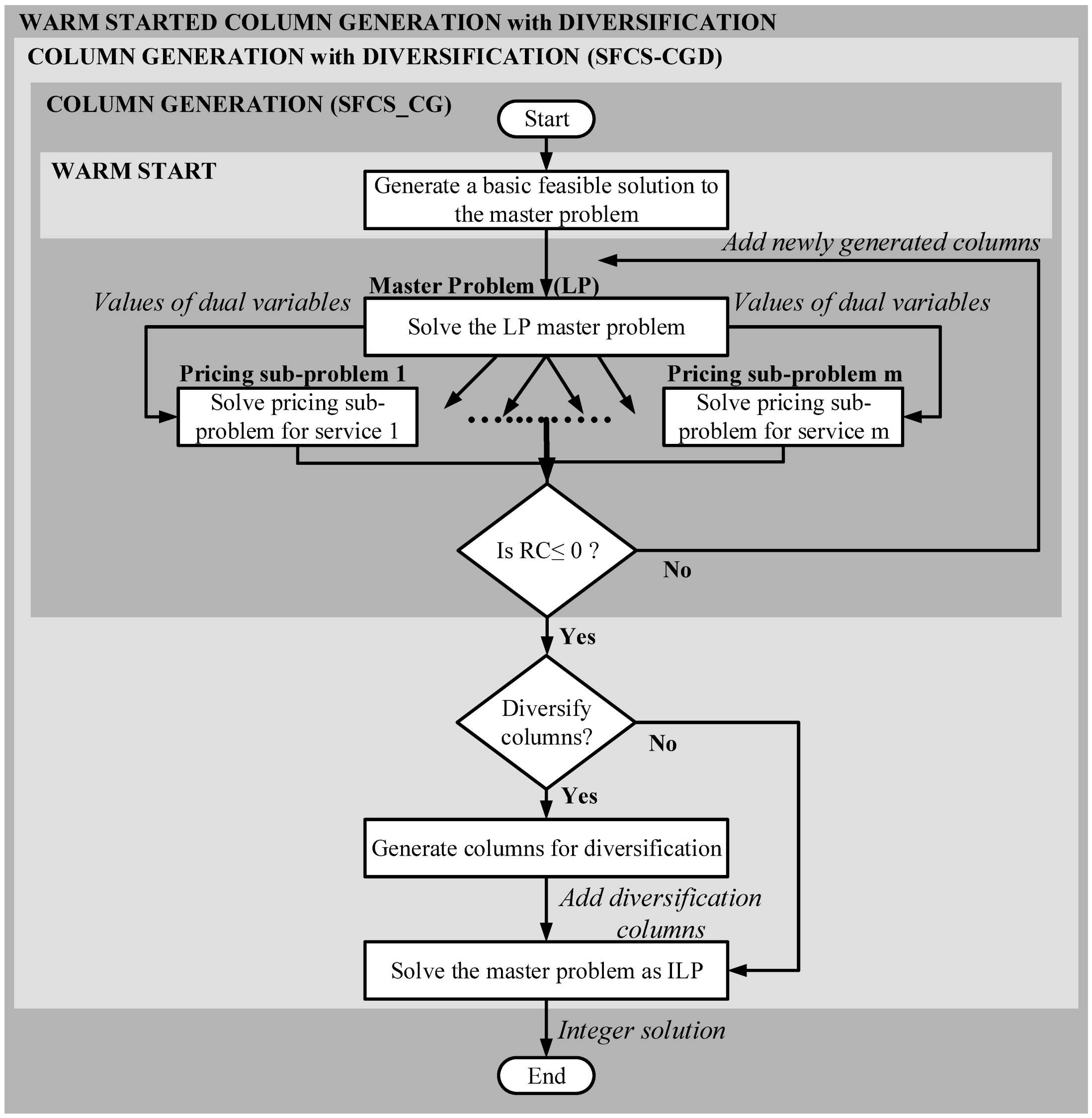

WARM STARTED COLUMN GENERATION with DIVERSIFICATION
COLUMN GENERATION with DIVERSIFICATION (SFCS-CGD)
COLUMN GENERATION (SFCS_CG)
Start
WARM START
Generate a basic feasible solution to the master problem
Add newly generated columns
Master Problem (LP)
Values of dual variables
Solve the LP master problem
Values of dual variables
Pricing sub-problem 1
Solve pricing sub-problem for service 1
Pricing sub-problem m
Solve pricing sub-problem for service m
Is RC≤ 0 ?
No
Yes
Diversify columns?
No
Yes
Generate columns for diversification
Add diversification columns
Solve the master problem as ILP
Integer solution
End

Figure 3.6: Column generation flow chart.

Hence, once a pricing SP is solved, the value of its RC (Eq.(3.26)) is evaluated; if the RC is positive, then the generated column is added to the LP MP. On the contrary, if the value of the RC is negative, the generated column is disregarded since it cannot improve the LP MP objective value. The pricing SPs and the LP MP keep iterating until an optimal solution is found. At each iteration, a pricing SP for each NS is run and the column it generates is added to the LP MP. The CG algorithm ends when the RCs of all the pricing SPs are negative [66]. At the end of the CG algorithm, an LP optimal solution of the SFCS problem, that we denote by ρ^*_{LP}, is reached. Fig.3.6 explores the steps of the CG algorithm.

Since we are interested in obtaining an ILP solution, we solve the MP (using the same columns generated by the pricing SPs at all the iterations) one last time without relaxing its integer variables ($\lambda^c_s \in \{0, 1\}$) and we denote by $\hat{\rho}^*_{ILP}$ its ILP objective value. Clearly, $\rho^*_{LP} \leq \rho^*_{ILP} \leq \hat{\rho}^*_{ILP}$ where ρ^*_{ILP} is the ILP optimal objective value obtained by solving the SFCS-MILP.

3.4.5 Diversification Approach (SFCS-CGD)

When solved as ILP, the MP provides an upper bound on the ILP optimal solution of the SFCS problem obtained by solving the SFCS-MILP. Our numerical evaluation on small networks (Section 3.6.1) shows a gap between ρ^*_{ILP} and $\hat{\rho}^*_{ILP}$ where ρ^*_{ILP} and $\hat{\rho}^*_{ILP}$ are as defined in Section 3.4.4. Such gap is due to solving the MP as ILP with only a subset of columns; those generated by the pricing SPs. In order to improve $\hat{\rho}^*_{ILP}$, we apply a diversification technique that consists of adding additional columns that may be beneficial to the MP (Fig.3.6).

To generate these diversification columns, we develop a heuristic as follows. Given a NS, and the number of columns to generate for this NS, the heuristic will map the NFs to VNFs chosen at random from the list of VNFs of the same type existing in the network. It will route the traffic between these VNFs using the shortest path. As for the NS schedule, the heuristic will choose different time slots to start the NS schedule for each column. For instance, for the first column (column 0), the NS schedule will start at time slot 0. This will give a lower bound on the minimum completion time of the NS, that will be used to decide on the start of the NS schedule in the other columns to generate. Let $minCompletionTime$ represent this completion time and let j be the ID of the column we are generating (i.e, $j = 0$ for column 0, $j = 1$ for column 1, etc.). We use the following three options to decide on the start time ($startTimeSlot$) of the schedule of the NS in each column:

1. Start the schedule of the NS in the next column at its completion time slot in the previous one. That is, $startTimeSlot = minCompletionTime * j$ $(\forall j, j \neq 0)$.
2. Start the schedule of the NS in the next column by adding the column id to the $minCompletionTime$. That is, $startTimeSlot = minCompletionTime + j$ $(\forall j, j \neq 0)$.
3. Start the NS schedule at a random time slot within the time line (Δ in Table 3.1).

The solution provided by the generated column should be feasible in the sense that the NS should complete its processing in the specified time line (Δ in Table 3.1) if it started processing its traffic at $startTimeSlot$. We will refer to this latter statement as $feasibleColumnCondition$. Thus, we alternate between options 1, 2 and 3 respectively when deciding on the $startTimeSlot$ in the sense that if $startTimeSlot$ generated using option 1 does not respect the $feasibleColumnCondition$, we use option 2 to generate it. Similarly, if $startTimeSlot$ generated using option 2 does not respect the $feasibleColumnCondition$, we keep on generating it randomly using option 3 until we get a value able to provide a feasible column. Using these three options will allow us to obtain several solutions for the same NS where some of them may have overlapping schedules and others may not. This method is beneficial and is able to improve the value of $\hat{\rho}^*_{ILP}$ as we will show in Section 3.6.

3.5 Online Column Generation (online-CG)

Solving the SFCS problem in an offline mode where all the NSs are known a priori is not practical due to the difficulty of the problem; further, in practice, the traffic is dynamic with random arrival and departure of the NSs. In this section, we present an online CG approach that solves the SFCS problem for dynamic traffic arrival.

During our study (i.e., Section 3.2), we have shown that the NF mapping, traffic routing and scheduling of a NS affects the SFCS solution of another one. We have also emphasized on the importance of reducing the total schedule length in contributing to a better utilization of resources. Given these motivations, solving the SFCS problem upon the arrival of a single NS is not of the best interest of the network operator as it will provide him with a local optimal solution for that specific NS, which may negatively affect the solution of future NSs and delay their processing.

To overcome this issue and provide the operator with a solution that addresses the online arrival of NSs while efficiently utilizing the network resources, we develop an online CG heuristic designed to run periodically on a batch of NSs. Given a batch of NSs arriving within a time interval, the online algorithm applies our CG approach on the NSs it holds. Batches generated at distinct periods can be of different size (hold different number of NSs). Our online CG algorithm is designed to apply one of both CG techniques, SFCS-CG or SFCS-CGD. In case of employing the diversification technique (SFCS-CGD), the number of diversification columns needs to be specified.

3.6 Numerical Evaluation

We carry out an extensive empirical study to evaluate the performance of our CG approach (SFCS-CG and CFCS-CGD) against the SFCS-MILP. Further, we explore the engineering impact of the SFCS problem on the network by pursuing numerical evaluation of our online CG algorithm (Section 3.5). During our numerical study, we use sets of NSs randomly generated, and demanding varying NFs ([3-5] NFs). The generated NSs are of varying traffic demands ([500-1500] Mbits) and bandwidth requirements ([300-500] Mbps). In addition, we consider a mesh network topology interconnecting the physical machines. We assume that each physical machine either represents a single server or a server rack and that the network can host as many servers as each rack can support. However, our work can be applied to any other topology, since the constraints of the presented formulations (SFCS-MILP, SFCS-CG and SFCS-CGD) are independent from the network topology. All our numerical evaluations are conducted using Cplex version 12.4 to solve the optimization problems on an Intel core i7-4790 CPU at 3.60 GHZ with 16 GB RAM.

3.6.1 SFCS-MILP vs CG

We consider a small test mesh network consisting of 4 physical servers hosting 5 VNFs placed at random. Each server is connected to a switch and the switches are interconnected by 5 links of 500 Mbps each. We use this network to run the SFCS-MILP and obtain reference results to compare with those acquired using our CG approach. To evaluate the performance of our proposed CG method, we also adopt a sequential methodology (using a sequential algorithm (SA)) in solving the mapping, routing and scheduling problems. In particular, rather than solving the three problems jointly, we first map (at random) the demanded NFs for each NS into VNFs of the same type in the network; then, we route the traffic along the shortest path (using Dijkstra algorithm) between each pair of the mapped functions. Finally, we solve the scheduling problem; here, once the routes of the NS are determined and the NF mapping is decided, we invoke a MILP to optimally solve the scheduling of the NSs on the corresponding functions with the objective always being minimizing the total schedule length of all the NSs. In addition, to the best of our knowledge, the only similar work that addressed a scheduling problem is that of [38]; hence, we compare our work with the best algorithm of [38] (a TS meta-heuristic). We should note that [38] only schedule functions once the NFs are mapped to VNFs and does not route the traffic in the network. We hence add the routing problem to make the comparison more realistic. We consider that all the NSs are known a priori (offline mode) and we run our tests over sets of NSs of different size.

	SFCS-MILP	**SFCS-CG**			**SFCS-CGD**	**TS**	**SA**
Nb. of NSs	*ILP Objective*	*LP Objective*	*ILP Objective*	*Nb. of iterations*	*ILP Objective*	*Schedule Length*	*Schedule Length*
3	16	16	16	10	16	22	19
6	22	21.4	36	19	36	41	38
9	24	22.092	65	25	36	49	53
12	35	26.576	65	40	63	61	63

Table 3.6: Optimality gap comparison.

1. *Optimality gap*

   Based on Table 3.6, one can clearly confirm that the SFCS-CG LP objective is a lower bound for the SFCS-MILP ILP objective and that the SFCS-CG ILP solution is an upper bound for it. When the number of NSs is small (3 NSs), both CG methods (SFCS-CG and SFCS-CGD) find the optimal solution (schedule length = 16 time slots) that the SFCS-MILP provides while the TS and SA algorithms fail to do so. However, when the number of NSs becomes larger, the optimality gap between the ILP objective value provided by the SFCS-CG and the SFCS-MILP increases (gap of 63% for 9 NSs). By adding 3 diversification columns for each NS in the set, our SFCS-CGD technique was able to decrease this gap by 29.74% for 9 NSs without incurring any valuable increase in the CG runtime (Table 3.7). Further, it is clear that our SFCS-CGD outperforms the SA and the TS algorithm in terms of optimality gap. However, it is important to note that even though the TS performed better than the SFCS-CGD for 12 NSs, we argue that the solution of the SFCS-CGD can be further improved by increasing the number of diversification columns used. Finally, one can note that with the increase of the number of NSs, the amount of resources (VNFs and bandwidth) available in the network decreases as they will be occupied by the traffic of some NSs whereas others will suffer from certain delays, waiting for some resources to be freed. This contributes to the increase of the schedule length with the increase of the number of NSs (Table 3.6).

2. *Execution time*

   In order to study the scalability of the SFCS-MILP model, we compare its execution time against our CG approach, the TS and the SA algorithms. Table 3.7 shows that when the number of NSs is small (3 NSs), all the five methods were able to find a solution in less than 10 seconds. However, as the number of NSs increases, the SFCS-MILP becomes much harder to solve and its runtime increases exponentially. For instance, when the number of NSs is 12, the execution time of the SFCS-MILP reaches 25 hours whereas our CG method returned a solution in less than 33 minutes which shows that the CG technique is much more scalable than the SFCS-MILP. Even though the execution times of the SFCS-CG and the SFCS-CGD increase when the number of NSs becomes larger, such increase is at slower pace than the SFCS-MILP which clearly shows that the CG method is much more scalable than the SFCS-MILP. As for the SA algorithm, its runtime mainly reflects the execution time of its scheduling model and shows that it is slightly more scalable than the CG approach as its model only performs the NS scheduling and is simpler than the pricing SP of the CG which solves the mapping, routing and scheduling problems. Finally, Table 3.7 depicts that the TS algorithm only needs few milliseconds to provide a solution and is the fastest as it does not involve solving any formulation.

3. *Impacts of diversification*

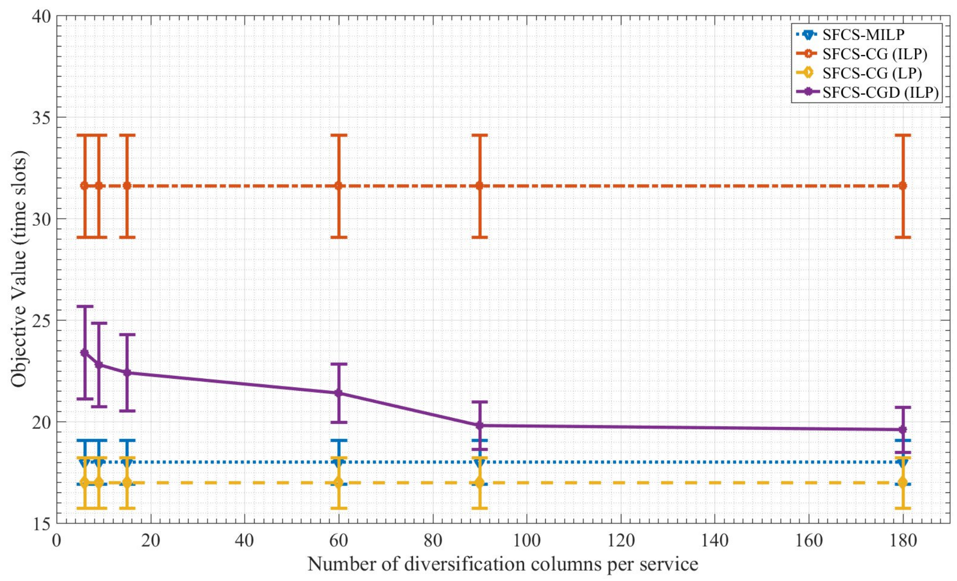
Nb. of NSs	SFCS-MILP	SFCS-CG	SFCS-CGD	TS	SA
3	1 362	6 146	5 459	4	5 291
6	168 879	79 213	78 410	4	40 695
9	11 121 471	280 209	276 422	3	162 327
12	91 285 122	1 935 455	1 977 515	3	1 068 856

Table 3.7: Execution Time (ms) comparison.

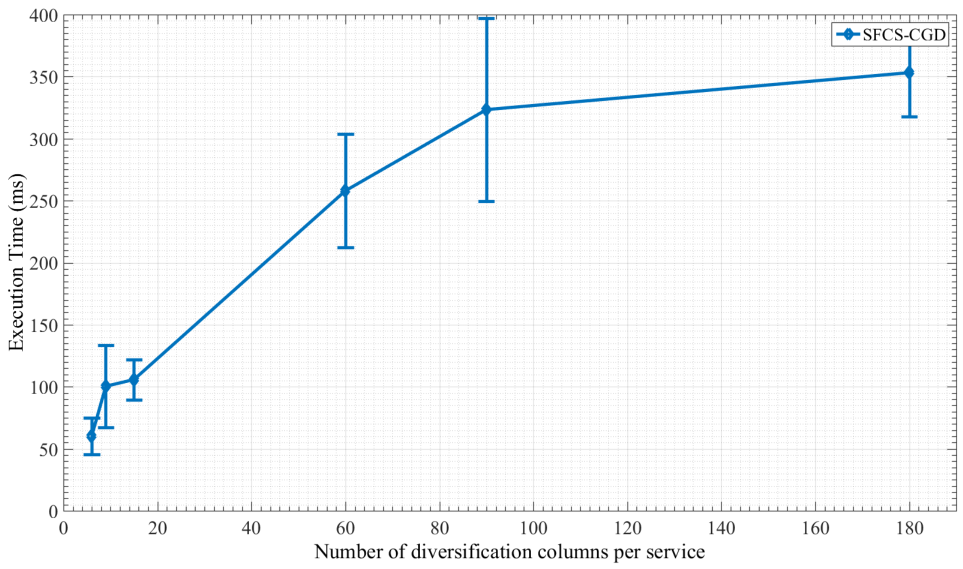
In the previous paragraph, we have shown that by applying our SFCS-CGD approach with 3 columns per NS, the optimality gap has decreased by 29.74% (for a set of 9 NSs). To further explore the benefits of our diversification technique, we investigate in Fig.3.7.a the impact of varying the number of columns on improving the optimality gap between the ILP solution obtained by the SFCS-MILP formulation and our CG approach. Thus, we vary the number of diversification columns on 5 sets of 6 NSs each and present the average objective value in Fig.3.7.a with 95% confidence interval. Our numerical evaluation shows that by increasing the number of diversification columns, the optimality gap decreases from $\rho^*_{ILP} = 23.4$ to reach an objective value $\hat{\rho}^*_{ILP} = 19.6$ which is only 1.6 time slots away from the optimal value provided by the SFCS-MILP, that is after adding 180 diversification columns per NS. Note that the large number of diversification columns needed to reach the optimal solution only costs a fraction of second in the execution time of the master model as ILP as shown in Fig.3.7.b.

3.6.2 Online CG

In the rest of our numerical evaluation, we focus on evaluating the performance of the online CG method. Hence, in order to show the impact of varying the network resources on reducing the total schedule length, we either vary the number of VNFs deployed in the network or the capacity of the links interconnecting the switches.



3.7.a Impact of diversification on the objective value.



3.7.b Impact of diversification on the execution time.

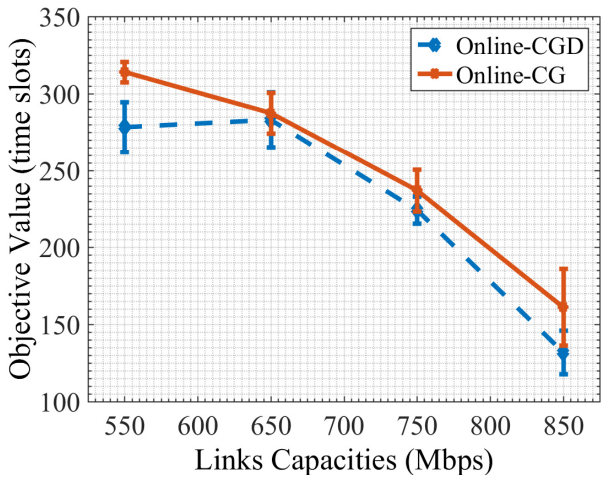
Figure 3.7: Impact of diversification.

We consider a mesh network of 8 physical servers, each connected to a switch. The switches are interconnected by 8 links. Hence, We run our online CG algorithm by considering 25 NSs randomly generated as described earlier, following a Poisson arrival with a rate of 5 NSs per time slot. We fix the period length to 10 time slots which will allow the division of the 25 NSs into batches of different size. Our results, depicted in Fig.3.8, are averaged over 5 sets and presented with 95% confidence interval. We represent by *Online-CG* the online CG algorithm run without applying any diversification technique. We also characterize by *online-CGD* our online CG heuristic applied by adding 3 diversification columns for each NS in the batch. We further exploit the execution time of our online-CG and online-CGD by comparing it to the TS and SA heuristics.

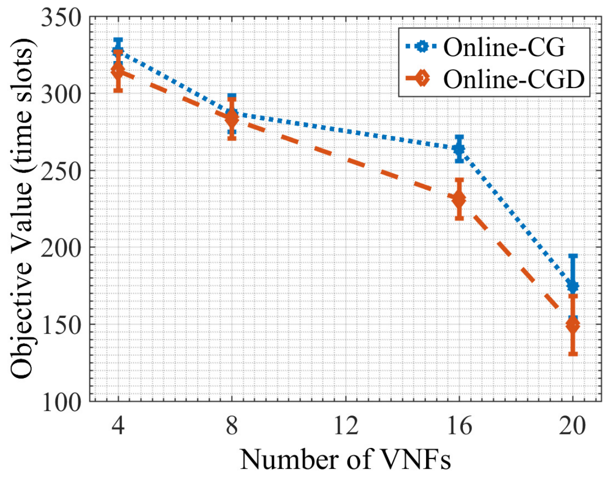
1. *Varying the capacity of the physical links*

   Some NSs may suffer from waiting delays due to insufficient bandwidth available on their routes. Hence, increasing the links capacity allow to reduce the waiting time of these NSs which will reduce their completion time and hence decrease the total schedule length. This will eventually affect the state of the network (occupied/free resources) at a certain time slot and thus, impact the schedule of the NSs in future batches. While considering 8 VNFs randomly deployed in the network, we show in Fig.3.8.a that increasing the links capacity decrease the schedule length of the 25 NSs averaged over 5 sets from 278.2 to 132 time slots with the online-CGD (gain of 52.55%). Note that, the increase in the objective value of the online-CGD when the capacity of the links becomes 650 Mbps can be explained by the fact that the ILP objective value obtained by the CG is an upper bound on the SFCS-MILP optimal value. In addition, it is clear that our diversification technique is able to reduce the gap between the ILP solution provided by CG and the optimal value given by the SFCS-MILP. Further, Fig.3.8.a affirms our initial statement about the decrease of the schedule length with the increase of the bandwidth in the network.

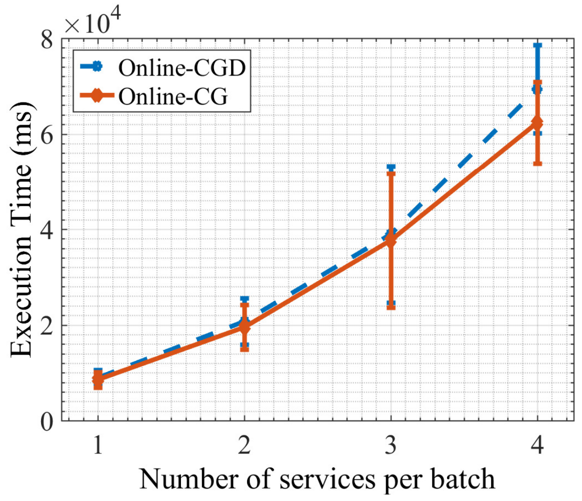
2. *Varying the number of VNFs*



3.8.a Varying the network bandwidth.



3.8.b Varying the number of VNFs in the network.



3.8.c Varying the batch size.

Figure 3.8: Impact of the network resources (VNFs, bandwidth) and batch size variation.

Many NSs may have some of their NFs mapped to the same VNFs. Since each VNF can process the traffic of one NS at a time, the traffic of the others have to wait till the processing of the traffic of the previous NS is completed. Thus, adding VNFs in the network can reduce the waiting time of the NSs by providing their NFs the possibility to be mapped to other available VNFs. Hence, we vary the number of VNFs randomly deployed in the network which have links of capacity equal to 500 Mbps connecting the switches. Fig.3.8.b depicts the decrease in the schedule length from 314.4 to 194.6 time slots while increasing the number of VNFs from 4 to 20 VNFs. Thus, providing a gain of 38% in reducing the schedule length with the Online-CGD approach. Further, the online-CGD algorithm was able to enhance the online-CG solution by an average of 7.43%.

3. *Varying the batch size*

   In order to evaluate the scalability of our CG methods in the online case, we simulate the same network of 8 physical servers and consider that some of its resources (VNFs, bandwidth) are occupied by a certain number of NSs. We consider the arrival of a batch of NSs and evaluate the execution time needed by our online-CG and online-CGD to provide a solution for each of the NSs in the batch. We consider batches of 1, 2, 3 and 4 NSs each. Our averaged results depicted in Fig.3.8.c show that the runtime of our online-CG and online-CGD increases with the size of the batch. However, one can note that it remains within the order of seconds even for a batch of 4 NSs. Hence, our CG methods can be easily used as benchmark algorithms.

3.7 Conclusion

In this chapter, we have presented a cross-layer strategy that solves three inter-related problems jointly; the NF mapping problem, the traffic routing problem and the NS scheduling problem. To the best of our knowledge, this is the first attempt that tackles these three problems jointly while considering bandwidth guarantees requirements for the requested NSs.

We have mathematically formulated the SFCS problem and highlighted its complexity. Given its complexity, we have presented a primal-dual decomposition approach that solves the SFCS problem using CG. To the best of our knowledge, we are the first to apply the CG technique to find a solution for this problem. Our CG technique is able to provide an LP lower bound and an ILP upper bound to the ILP optimal solution obtained by the SFCS-MILP. Using a diversification technique, we have shown through numerical evaluation that our SFCS-CGD method can decrease the gap between the CG ILP objective and the SFCS-MILP optimal objective value and can eventually reach it by increasing the number of diversification columns. In addition, we have compared our CG methods to a sequential method (SA) and to the TS developed in [38] and showed that our primal-dual decomposition outperforms them in terms of objective value. Further, we have shown that our CG approach is much more scalable than the SFCS-MILP. In addition, we have explored the impact of provisioning more network resources (VNFs, bandwidth) in decreasing the total schedule length of the NSs.

Finally, the major advantage of this work is revealed by its ability to serve as a benchmark for evaluating the performance of any low complexity method for solving the SFCS problem on larger network instances, where no known exact solutions can be found.

Chapter 4

Enabling Low-Latency Services in Softwarized Networks[1]

To support diverse business verticals (i.e., manufacturing, health care, etc.) with varying QoS requirements (e.g., ultra-low latency, ultra-reliability, etc.), 5G mobile networks are envisioned to encourage agility, programmability and elasticity through enabling a software-based architecture promoted by network slicing. Network slicing is a new paradigm consisting of partitioning the underlying network infrastructure into different logical network slices, each dedicated to address the requirements of a group of NSs. Motivated by these challenges, we revisit in this chapter the same joint problem of NF mapping, traffic routing and NS scheduling by accounting for strict latency requirements of different NSs in addition to determined buffer capacities of VNFs. After mathematically formulating the problem and given its complexity, we present a novel game theoretic approach to solve it, that yields much more scalable that the CG technique that we proposed in the previous chapter. Finally, we highlight through numerical evaluation the efficiency of the proposed method under different system parameters, in addition to presenting several insights related to the use of

[1]This chapter has been submitted to IEEE Transactions on Cloud Computing [67].

different routing methods.

4.1 Introduction

A wide variety of new use cases and business models in the areas of health care, manufacturing, transport and entertainment industries are being introduced today with the emergence of IoT devices [2]. Assistant driving, traffic safety, smart parking, remote surgery, tactile Internet and many more applications are currently being developed and envisioned to be introduced with the launch of 5G networks [2, 11]. While current networks are far from meeting the 5G requirements given their limited scalability and elasticity, network slicing emerged as a new paradigm to enable the accommodation of heterogeneous NSs sharing the same infrastructure [68, 69]. Network slicing consists of partitioning a common network infrastructure into multiple virtual logical networks or slices, each designed to support a group of NSs with similar requirements [69]. Thus, one or more network slice can be specifically designed to accommodate each of the 5G vertical industries. For instance, slices guaranteeing high data rate (peak data rate of 10 Gbps) are dedicated for mobile broadband services (i.e. audio/video streaming, etc.), others designed to achieve stringent throughput, latency (less than 1 ms) and reliability demands, are designed for ultra-reliable and ultra-low latency services (i.e. remote medical surgery, virtual reality, etc.), while slices transmitting a relatively low volume of non delay-sensitive data can accommodate the machine type communication services (i.e., smart home and cities, etc.) [11].

The partitioning of a shared infrastructure into slices is enabled by the latest virtualization technologies such as NFV and SDN. In fact, each network slice consists of a set of VNFs that run on top of a partially shared infrastructure composed of generic hardware resources such as NFVI resources [9] in addition to some dedicated hardware such as network elements in the Radio Access Network (RAN) [69].

Processing the traffic of a NS by a chain of NFs requires mapping these NFs to VNFs of the same type (i.e., cache, proxy, etc.) already deployed in the network, routing the traffic through the physical path and scheduling its processing by these VNFs. Thus, we consider an ultra-low latency network slice to achieve the requirements of NSs with stringent deadlines by jointly addressing the aforementioned challenges through a game theoretic approach. The proposed game theoretic technique solves this joint problem under a hybrid strategy which captures the centralized aspect of the problem in providing a coherent schedule between the NSs while leveraging the decentralization of the mapping, routing and scheduling decisions to be taken by each NS with the guidance of a centralized controller.

4.1.1 Novel Contributions

Our contributions are summarized as follows:

1. While accounting for the interplay between the NF mapping, traffic routing and NS scheduling, we formulate the joint problem composed of the aforementioned problems as a MILP and refer to it by the *Latency-Aware Service Scheduling (LASS)* problem.

2. Owing to its complexity, we model the LASS problem as a non-cooperative extensive-form game where the NSs act as the players of the game. To the best of our knowledge, we are the first to address this problem as a mixed strategy game (*LASS-Game*) which provides NSs the freedom to decide on their own mapping, routing and scheduling solution while orchestrating their schedules through a centralized controller.

3. We show that the LASS-Game admits a mixed strategy Nash equilibrium. We

provide an upper bound on its price of anarchy and we develop a best response algorithm to find an approximate equilibrium.

4. We evaluate through extensive simulations our game theoretic approach under different system parameters and using different routing methods.

The rest of this chapter is organized as follows: Section 4.2 discusses the system model and motivates the problem. Section 4.3 defines and formulates the LASS problem. Section 4.4 presents the LASS-Game. Section 4.5 depicts our numerical evaluation. We conclude in Section 4.6.

4.2 System Model

4.2.1 Ultra-Low Latency Network Slice

A network slice is a virtual network running on top of a physical network designed to address the specific requirements of the services it targets in terms of latency, reliability, security, availability and speed [69]. We consider in this work, an ultra-low latency network slice to address the specific requirements of ultra-low latency NSs. For instance, NSs of autonomous driving cars require a NF-FG composed of authentication, video encoding and screen rendering functions that should be available in the network slice hosting these NSs. Hence, we consider a network slice running such type of NFs and designed to support such low-latency requirements. Fig.4.1 depicts such an ultra-low latency slice composed of virtual nodes representing the forwarding devices in addition to the VNFs dedicated for it and running on top of the physical infrastructure. Thus, we define a network slice as a virtual network depicted by a connected graph $V(F, C)$ of a set F of VNFs of different types and a set C of links connecting them [70]. Each link $(ij) \in C$ has a capacity denoted by

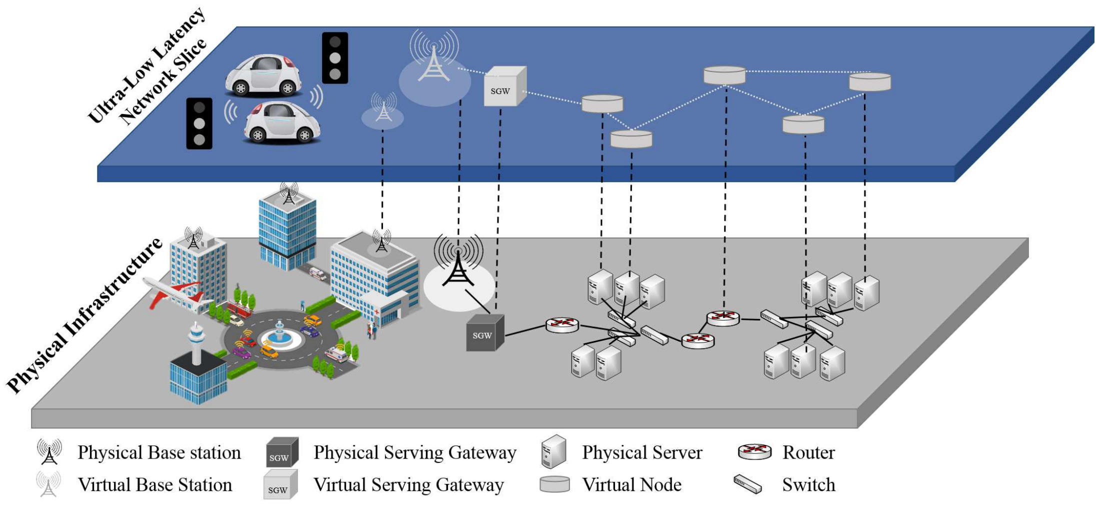


Figure 4.1: Ultra-low latency network slice running on top of a physical infrastructure.

c_{ij}. For simplicity and without loss of generality, we consider that the VNFs in F are dedicated to the ultra-low latency slice and are not shared by any other slice. However, they can be shared by many NSs processed by that same slice. Further, given that VNFs are software components that run on VMs hosted on commodity hardware in the NFVI [9, 28, 56], they require certain computing resources (i.e., CPU, RAM, storage, etc.), have a defined processing capacity that we denote by p_f and a buffer capacity that we depict by ϕ_f [28]. For simplicity, we assume that each VM is dedicated to exactly one VNF and that VNFs cannot share the same VM resources [38].

Finally, it is worth noting that in this chapter, we consider that the VNFs dedicated for the network slice are already placed in the physical network and are guaranteed their required computing (i.e, CPU, memory) and network (i.e. bandwidth) resources.

4.2.2 Problem Description

We consider a NFV-based physical network virtualized into several network slices. We account for ultra-low latency NSs, each requesting its traffic to be processed by a chain of NFs with a specified delay constraint. Each of these NSs requires a certain amount of bandwidth to be guaranteed for the transmission of its traffic between the NFs composing its NF-FG. Satisfying it, demands:

1. Determining the VNFs that will process its traffic with respect to its NF-FG. More precisely, determining the mapping of each NF demanded by the NS to a VNF deployed in the network slice.

2. Routing the traffic between the VNFs on which the NFs are mapped while guaranteeing the needed bandwidth and respecting the order of NFs in the NF-FG. The transmission delays= (d_s) between each two consecutive VNFs in the chain of a NS s can be calculated as in Eq.(4.1) where b_s depicts the bandwidth to be guaranteed for s and w_s represents its traffic size.

$$d_s = \frac{w_s}{b_s} \tag{4.1}$$

3. Deciding on the NS schedule which entails determining the time slots at which its traffic get processed on each of the VNFs. Such scheduling is important as it satisfies the NS latency requirement given that different NSs can share the same VNF. Note that we assume that each VNF can process the traffic of at most one NS at a certain time slot (i.e., no sharing). The processing time p_{fs} required to process the traffic of a NS s on a VNF f is calculated using Eq.(4.2)

where w_s and p_f are as defined earlier.

$$p_{fs} = \frac{w_s}{p_f} \tag{4.2}$$

4.3 LASS - A Mixed Integer Linear Program

4.3.1 Problem Definition

We consider a physical network $G(K, L)$ of a set K of nodes ($K = K_p \cup K_n$, K_p denotes the set of physical servers, K_n represents the set of physical forwarding devices (i.e., routers/switches)) and L is a set of physical links connecting them. A set F of VNFs are deployed in $G(K, L)$, each is of a specified type $t_f \in T$ and has a buffer capacity ϕ_f and a processing capacity p_f. Without loss of generality, we consider that VNFs of the same type have a uniform processing capacity. We use $x_f^k \in \{0, 1\}$ to specify that the VNF $f \in F$ is hosted on physical server $k \in K_p$ (1) or not (0).

We consider a set S of ultra-low latency NSs where each NS requests its traffic to be processed by one or a chain of NFs in a determined order within a specified deadline and transmitted from one NF to another at a guaranteed bandwidth. Hence, we represent each NS $s \in S$ by a tuple $(H_s(N_s, E_s), w_s, b_s, u_s)$ where $H_s(N_s, E_s)$ represents the forwarding graph of the NS, N_s is the set of NFs requested by the NS and E_s depicts the set of virtual links connecting them (Fig.4.2.b). Each NF $n \in N_s$ is of specified type denoted by m_{ns}. The processing time of the traffic demands w_s is denoted by p_{ns} and can be calculated based on Eq.(4.2) given that the NF $n \in N_s$ will be mapped to a VNF of the same type. b_s represents the bandwidth to be guaranteed for the communication between the NFs $n \in N_s$ (b_s is an attribute of each of the virtual links $e \in E_s$) and u_s is the deadline of the NS $s \in S$ (in terms of time slots). We represent the transmission delay of the traffic of NS $s \in S$ on a virtual link $e \in E_s$

by d_s (Eq.(4.1)). We designate the time line (set of time slots) by Δ. Hence, we define the LASS problem with the objective of maximizing the number of admitted NSs as follows, and we refer to it by *LASS-MaxAdmission*.

Definition 4.1. *Given $G(K, L)$ hosting and running different types of VNFs, a set S of NSs, each demanding to be processed by a chain of NFs, find their optimal NF mapping, traffic routing and scheduling which maximize the number of admitted ones while respecting their deadlines.*

4.3.2 Problem Formulation

	Physical network inputs
$G(K, L)$	Physical network.
F	Set of VNFs hosted in $G(K, L)$.
$t_f \in \mathbb{Z}^+$	Type of a VNF instance $f \in F$ ($t_f \in T$)
$\phi_f \in \mathbb{Z}^+$	Capacity of the buffer of a VNF instance $f \in F$.
$p_f \in \mathbb{Z}^+$	Processing capacity of a VNF instance $f \in F$.
$x_f^k \in \{0, 1\}$	VNF instance $f \in F$ is hosted on the physical server $k \in K_p$ (1) or not (0).
$c_{ij} \in \mathbb{Z}^+$	Capacity of a physical link $(ij) \in L$.
	NS inputs
S	Set of NSs.
$H_s(N_s, E_s)$	NF-FG of a NS $s \in S$ constituted by a set N_s of NFs and a set E_s of virtual links connecting them.
$b_s \in \mathbb{Z}^+$	Bandwidth demanded by NS $s \in S$.
$w_s \in \mathbb{Z}^+$	Traffic demands of NS $s \in S$.
$u_s \in \mathbb{Z}^+$	Deadline of NS $s \in S$.
$m_{ns} \in \mathbb{Z}^+$	Type of a NF $n \in N_s$ of NS $s \in S$.
$p_{ns} \in \mathbb{Z}^+$	Processing time of the traffic of NS $s \in S$ on the NF $n \in N_s$.
Δ	Set of time slots $\delta \in \Delta$ (time line).

Table 4.1: Parameters of the LASS-MaxAdmission.

Table 4.1 delineates the parameters used in the formulation of the LASS-MaxAdmission problem presented below. We define the decision variable $a_s \in \{0, 1\}$ to determine if

Decision variables of the LASS-MaxAdmission	
$a_s \in \{0,1\}$	Determines that a NS $s \in S$ is admitted to the network (1) and (0) otherwise.
$y_{ns}^{f\delta} \in \{0,1\}$	Specifies that the traffic of NS s started processing at time slot δ on VNF $f \in F$ to which NF n is mapped (1) and (0) otherwise.
$\psi_s^{f\delta} \in \{0,1\}$	Denotes that the traffic of NS s is queued at δ in the buffer of the VNF f (1) and (0) otherwise.
$q_{ns}^k \in \{0,1\}$	Determines that NF n of NS $s \in S$ is mapped to a VNF hosted on server k (1) and (0) otherwise.
$h_{ns}^k \in \{0,1\}$	Indicates that NFs $n, (n+1)$ of NS $s \in S$ are mapped to VNFs hosted on the same server k (1) and (0) otherwise.
$\theta_s^{\delta e} \in \{0,1\}$	Designates that a NF $o(e) \in N_s$ of NS s begins the transmission of the traffic to its successor NF $d(e)$ at time slot δ on the virtual link e (1) and (0) otherwise.
$\hat{\theta}_s^{\delta e} \in \{0,1\}$	Indicates that the virtual link e is being used for traffic transmission between the NFs $o(e)$ and $d(e)$ of NS s at time slot δ (1) and (0) otherwise.
$l_{ij}^e \in \{0,1\}$	Denotes that the virtual link e of NS s is routed through the physical link $(ij) \in L$ (1) and (0) otherwise.

Table 4.2: Decision variables of the LASS-MaxAdmission.

a NS $s \in S$ is admitted to the network. A NS is admitted to the network if it can be scheduled within its deadline.

$$a_s = \begin{cases} 1 \text{ if NS } s \text{ is admitted to the network,} \\ 0 \text{ otherwise.} \end{cases}$$

Our objective is to maximize the number of admitted NSs.

$$\text{Maximize} \quad \sum_{s \in S} a_s \tag{4.3}$$

This objective is subject to several constraints. Thus, we define a new variable $y_{ns}^{f\delta} \in \{0,1\}$ to specify that the traffic of NS s started processing at time slot $\delta \in \Delta$ on the

VNF $f \in F$ to which NF $n \in N_s$ is mapped (1) and (0) otherwise.

$$y_{ns}^{f\delta} = \begin{cases} 1 \text{ if NF } n \text{ of NS } s \text{ started processing on VNF } f \text{ at } \delta, \\ 0 \text{ otherwise.} \end{cases}$$

We define $\psi_s^{f\delta} \in \{0,1\}$ to denote that the traffic of NS s is queued at $\delta \in \Delta$ in the buffer of the VNF f.

$$\psi_s^{f\delta} = \begin{cases} 1 \text{ if the traffic of } s \text{ is queued at } \delta \text{ in the buffer of } f, \\ 0 \text{ otherwise.} \end{cases}$$

In addition, we define the variable $q_{ns}^k \in \{0,1\}$ to depict that the NF $n \in N_s$ of NS s is mapped to a VNF instance hosted on physical server $k \in K_p$.

$$q_{ns}^k = \begin{cases} 1 \text{ if NF } n \text{ of } s \text{ is mapped to VNF } f \text{ hosted on } k, \\ 0 \text{ otherwise.} \end{cases}$$

$h_{ns}^k \in \{0,1\}$ is another decision variable which indicates that NFs $n, (n+1) \in N_s$ of NS s are mapped to VNFs hosted on the same physical server $k \in K_p$.

$$h_{ns}^k = \begin{cases} 1 \text{ if NFs } n \text{ and } (n+1) \text{ of NS } s \text{ are hosted on } k, \\ 0 \text{ otherwise.} \end{cases}$$

In order to handle the routing and transmission delays, we declare $\theta_s^{\delta e} \in \{0,1\}$ to designate that a NF $o(e) \in N_s$ of NS $s \in S$ begins the transmission of the traffic to its successor NF $d(e) \in N_s$ at time slot $\delta \in \Delta$ on the virtual link $e \in E_s$ (1) (or not, 0).

$$\theta_s^{\delta e} = \begin{cases} 1 \text{ if NF } o(e) \text{ started the transmission of the traffic of} \\ \; s \text{ to NF } d(e) \text{ at time slot } \delta, \\ 0 \text{ otherwise.} \end{cases}$$

We also declare $\hat{\theta}_s^{\delta e} \in \{0,1\}$ to indicate that the virtual link $e \in E_s$ is being used for the transmission of the traffic between the NFs $o(e)$ and $d(e)$ at time slot $\delta \in \Delta$ (1) (or not, 0).

$$\hat{\theta}_s^{\delta e} = \begin{cases} 1 \text{ if } e \text{ is transmitting the traffic of } s \text{ at time slot } \delta, \\ 0 \text{ otherwise.} \end{cases}$$

Further, we denote by $l_{ij}^e \in \{0,1\}$, a decision variable that depicts that the virtual link $e \in E_s$ of NS $s \in S$ is routed through the physical link $(ij) \in L$ (1) (or not, (0)).

$$l_{ij}^e = \begin{cases} 1 \text{ if } e \text{ is routed through the physical link } (ij) \in L, \\ 0 \text{ otherwise.} \end{cases}$$

Table 4.2 summarizes the LASS-MaxAdmission decision variables.

1. *NF mapping constraints*

   For a NS s to be admitted to the network, each of its NFs $n \in N_s$ has to be mapped to exactly one VNF $f \in F$ (Eq.(4.4)).

$$\sum_{f\in F}\sum_{\delta\in\Delta} y_{ns}^{f\delta} = a_s \quad \substack{\forall n\in N_s \\ \forall s\in S} \tag{4.4}$$

   Such mapping should guarantee that the requested NF n and the VNF f to which it is mapped are of the same type (Eq.(4.5)).

$$\sum_{f \in F} \sum_{\delta \in \Delta} y_{ns}^{f\delta} t_f = a_s m_{ns} \quad \substack{\forall n \in N_s \\ \forall s \in S} \tag{4.5}$$

2. *NS scheduling constraints*

Further, we define Eq.(4.6) to ensure that a VNF $f \in F$ is processing the traffic of NS s during all the processing period of this latter and prevents it from processing the traffic of another NS s' during the same period.

$$\sum_{s\prime \in S : s\prime \neq s} \sum_{n\prime \in N_{s\prime}} y_{n's'}^{f\delta'} \leq 1 - y_{ns}^{f\delta} \quad \substack{\forall n \in N_s \\ \forall s \in S \\ \forall f \in F \\ \forall \delta, \delta' \in \Delta ; \delta \leq \delta' < \delta + p_{ns}} \tag{4.6}$$

Given that the traffic of a NS s should be processed by the NFs in the requested order depicted in its NF-FG, we define Eq.(4.7) to prevent a NF $(n+1)$ to start processing the traffic of s before its predecessor NF n finishes its execution.

$$\sum_{f \in F} y_{(n+1)s}^{f\delta'} \leq 1 - \sum_{f \in F} y_{ns}^{f\delta} \quad \substack{\forall \delta, \delta' \in \Delta ;\ \delta' < \delta + p_{ns} \\ \forall n, (n+1) \in N_s \\ \forall s \in S} \tag{4.7}$$

Further, such processing of the traffic cannot start on NF $(n+1)$ before being transmitted to it by its predecessor NF n. Hence, transmission delays should be taken into account in the schedule in the case where NFs n and $(n+1)$ are mapped to VNFs running on different physical servers. Thus, we first determine the physical server hosting the VNF f to which the NF n of NS s is mapped (Eq.(4.8)).

$$q_{ns}^k = \sum_{f \in F} \sum_{\delta \in \Delta} y_{ns}^{f\delta} x_f^k \quad \substack{\forall n \in N_s \\ \forall s \in S \\ \forall k \in K_p} \tag{4.8}$$

Given the above information, we specify if the two consecutive NFs n and $(n+1)$ of the NS s are mapped to VNFs hosted on the same physical server $k \in K_p$

(Eq.(4.9)).

$$h_{ns}^k = q_{ns}^k q_{(n+1)s}^k \quad \substack{\forall k \in K_p \\ \forall n,(n+1) \in N_s \\ \forall s \in S} \tag{4.9}$$

Using the value of h_{ns}^k, we determine Eq.(4.10) to prevent the start of the transmission of traffic of a NS s between two consecutive NFs $o(e)$ and $d(e)$ if they are mapped to VNFs hosted on the same physical server or if the NS is not admitted.

$$\sum_{\delta \in \Delta} \theta_s^{\delta e} = (1 - \sum_{k \in K_p} h_{o(e)s}^k) a_s \quad \substack{\forall e \in E_s \\ \forall s \in S} \tag{4.10}$$

In addition, we guarantee that the transmission of traffic of a NS s between two consecutive NFs $o(e)$ and $d(e)$ can only start when its processing on $o(e)$ is completed (Eq.(4.11)).

$$\theta_s^{\delta' e} \leq 1 - \sum_{f \in F} y_{o(e)s}^{f\delta} \quad \substack{\forall \delta, \delta' \in \Delta;\ \delta' < \delta + p_{o(e)s} \\ \forall e \in E_s \\ \forall s \in S} \tag{4.11}$$

Similarly, the processing of the traffic of a NS s cannot start on NF $d(e)$ before its transmission to it is finalized (Eq.(4.12)).

$$\sum_{f \in F} y_{d(e)s}^{f\delta'} \leq 1 - \theta_s^{\delta e} \quad \substack{\forall \delta, \delta' \in \Delta;\ \delta' < \delta + \frac{w_s}{b_s} \\ \forall e \in E_s \\ \forall s \in S} \tag{4.12}$$

We account for the transmission delays while ensuring that the virtual link e is used to transmit traffic of a NS s during all the required transmission time ($\frac{w_s}{b_s}$) (Eq.(4.13)).

$$\sum_{\delta \in \Delta} \hat{\theta}_s^{\delta e} = \frac{w_s}{b_s} \sum_{\delta \in \Delta} \theta_s^{\delta e} \quad \substack{\forall e \in E_s \\ \forall s \in S} \tag{4.13}$$

Eq.(4.14) ensures that the virtual link e is occupied during all the transmission period ($[\delta,\ \delta+\frac{w_s}{b_s}-1]$).

$$\sum_{\delta'\in[\delta,\ \delta+\frac{w_s}{b_s}-1]} \hat{\theta}_s^{\delta' e} \geq \frac{w_s}{b_s}\theta_s^{\delta e} \quad \substack{\forall e\in E_s\\ \forall s\in S\\ \forall \delta\in\Delta} \tag{4.14}$$

To account for each VNF buffer capacity, we define Eq.(4.15) to specify that the traffic of a NS s is queued in the buffer of VNF f during all the waiting time between the end of its transmission to f and the beginning of its processing by f.

$$\sum_{\delta\in\Delta:\delta'+\frac{w_s}{b_s}\leq\delta<\delta''} \psi_s^{f\delta} \geq \delta'' y_{d(e)s}^{f\delta''} - (\delta'+\frac{w_s}{b_s})\theta_s^{\delta' e} \quad \substack{\forall\delta',\delta''\in\Delta:\delta'\leq\delta''\\ \forall e\in E_s\\ \forall s\in S} \tag{4.15}$$

We define Eq.(4.16) to determine that the traffic of NS s cannot be queued in the buffer of VNF f at any time slot after the beginning of its processing on f.

$$\psi_s^{f\delta} \leq 1 - \sum_{n\in N_s} y_{ns}^{f\delta'} \quad \substack{\forall\delta,\delta'\in\Delta:\delta\geq\delta'\\ \forall f\in F\\ \forall s\in S} \tag{4.16}$$

Further, Eq.(4.17) guarantees that the traffic of NS s cannot be queued in the buffer of VNF f before the end of its transmission to f.

$$\psi_s^{f\delta} \leq 1 - \theta_s^{\delta' e} \sum_{\delta''\in\Delta} y_{d(e)s}^{f\delta''} \quad \substack{\forall\delta,\delta'\in\Delta:\delta<\delta'+\frac{w_s}{b_s}\\ \forall e\in E_s\\ \forall f\in F\\ \forall s\in S} \tag{4.17}$$

We determine Eq.(4.18) to guarantee that the traffic of a NS s cannot be waiting in the buffer of a VNF f which is not used by s, that is, non of the NFs $n\in N_s$ is mapped to f.

$$\psi_s^{f\delta} \leq \sum_{n\in N_s}\sum_{\delta'\in\Delta} y_{ns}^{f\delta'} \quad \substack{\forall\delta\in\Delta\\ \forall f\in F\\ \forall s\in S} \tag{4.18}$$

Eq.(4.19) ensures that the buffer capacity is respected.

$$\sum_{s \in S} \psi_s^{f\delta} w_s \leq \phi_f \quad \substack{\forall \delta \in \Delta \\ \forall f \in F} \tag{4.19}$$

We guarantee that a NS s meets its deadline by ensuring that the processing of its traffic by the last NF ($|N_s|$) in its NF-FG is finalized before its deadline (Eq.(4.20)).

$$\sum_{f \in F} \sum_{\delta \in \Delta} y_{|N_s|s}^{f\delta} (\delta + p_{|N_s|s}) \leq u_s \quad \forall s \in S \tag{4.20}$$

3. *Traffic routing constraints*

The traffic routing is handled by Eq.(4.21) which represents the flow conservation constraint.

$$\sum_{j:(i,j) \in L} l_{ij}^e - \sum_{j:(j,i) \in L} l_{ji}^e = q_{o(e)s}^i - q_{d(e)s}^i \quad \substack{\forall e \in E_s \\ \forall s \in S \\ \forall i \in K_p} \tag{4.21}$$

However, we prevent through Eq.(4.22) the routing of a virtual link e of a NS s through a physical link $(ij) \in L$ if $o(e)$ and $d(e)$ are mapped to VNFs hosted on the same physical server.

$$l_{ij}^e \leq 1 - \sum_{k \in K_p} h_{o(e)s}^k \quad \substack{\forall e \in E_s \\ \forall s \in S \\ \forall (ij) \in L} \tag{4.22}$$

In addition, we define Eq.(4.23) to guarantee that the physical links capacity is not violated.

$$\sum_{s \in S} \sum_{e \in E_s} l_{ij}^e \hat{\theta}_s^{\delta e} b_s \leq c_{ij} \quad \substack{\forall \delta \in \Delta \\ \forall (ij) \in L} \tag{4.23}$$

Eq.(4.24) is defined to prevent the routing of a virtual link e of a NS s through a physical link $(ij) \in L$ if s is not admitted.

$$l_{ij}^{e} \leq a_s \quad \substack{\forall e \in E_s \\ \forall s \in S \\ \forall (ij) \in L} \tag{4.24}$$

Eqs.(4.9), (4.10), (4.17) and (4.23) are non linear and can be easily linearized as explained in Appendix C.

4.3.3 Variations of the LASS problem

In order to highlight the interaction between the scheduling of different NSs, and the impact of the schedule of one on another, we consider different variations of the LASS problem by varying the objective function and removing the latency constraint (Eq.(4.20)). Thus, in the following, we present two other formulations of the LASS problem where the parameters and the decision variables are respectively defined in Table 4.1 and Table 4.2. Additional decision variables are depicted below.

4.3.3.1 LASS - Minimize the Maximum Completion Time (LASS-MinMaxCT)

We define the LASS problem under the objective of minimizing the maximum completion time of the NSs, that is minimizing the schedule length of all the scheduled NSs. Thus, we declare a new decision variable $\varrho \in \mathbb{N}^{+}$ which represents the maximum completion time of all the scheduled NSs. The objective is presented by Eq.(4.25).

$$\text{Minimize} \quad \varrho \tag{4.25}$$

This objective is subject to several constraints depicted by Eq.(4.4) to Eq.(4.19), Eq.(4.21) to Eq.(4.23) and Eq.(4.26) where Eq.(4.26) ensures that the maximum completion time of all NSs, is greater or equal to the completion time of each of them.

$$\varrho \geq \sum_{f \in F} \sum_{\delta \in \Delta} y^{f\delta}_{|N_s|s}(\delta + p_{|N_s|s}) \quad \forall s \in S \tag{4.26}$$

Note that we drop the decision variable a_s and replace it by 1 in Eq.(4.4), Eq.(4.5) and Eq.(4.10).

4.3.3.2 LASS - Minimize the Sum of Completion Times (LASS-MinSumCT)

We reformulate the LASS problem under the objective of minimizing the sum of completion times of all the NSs (Eq.4.27).

$$\text{Minimize} \quad \sum_{s \in S} \sum_{f \in F} \sum_{\delta \in \Delta} y^{f\delta}_{|N_s|s}(\delta + p_{|N_s|s}) \tag{4.27}$$

This objective is subject to several constraints presented in Eq.(4.4) to Eq.(4.19), Eq.(4.21) to Eq.(4.23) and Eq.(4.26). Note that we drop the decision variable a_s and replace it by 1 in Eq.(4.4), Eq.(4.5) and Eq.(4.10).

4.3.4 Problem Complexity

The LASS-MaxAdmission, the LASS-MinMaxCT and the LASS-MinSumCT are MLIPs which are complex to solve. They are NP-Hard since they solve three NP-Hard problems which are the NF mapping problem [25], the traffic routing problem [34] and the NS scheduling problem [71]. These programs detail the multiple constraints that should be respected to provide a feasible solution for the LASS problem. While they can be used as benchmark methods to compare against given that they provide an optimal solution of the problem, it is impractical to use them in real deployments of networks of realistic size due to their non scalability (as we will show in Section 4.5). Thus, in the following, we propose a more scalable approach to solve the LASS problem with respect to its presented constraints. We use a game theoretic technique that can provide some performance guarantees on the quality of the solution.

4.4 LASS- A Game Theoretic Approach (LASS-Game)

To overcome the complexity of centralized approaches (i.e., LASS-MaxAdmission, LASS-MinMaxCT, LASS-MinSumCT), we seek at delegating the mapping, routing and scheduling decisions to each NS, hence departing from the centralization of the decision making process. Thus, we present the *LASS-Game: Latency-Aware Scheduling Game*, a game-theoretic approach to solve the LASS problem.

4.4.1 Exploring Mapping, Routing and Scheduling Solutions

Giving each NS the freedom to decide on its mapping, routing and scheduling solution requires providing the NS with a procedure to explore the possible solutions of its LASS problem. Such a procedure consists of building a connected virtual graph of all the aforementioned possibilities. To this end, each NS $s \in S$ builds its own virtual directed network $H'_s(N'_s, E'_s)$ where N'_s is a set of nodes and E'_s is a set of edges connecting them.

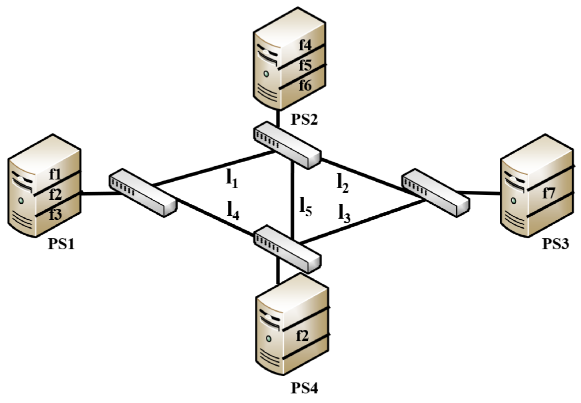
Given $G(K, L)$, the set F of VNFs it is hosting in addition to the NF-FG of s depicted by $H_s(N_s, E_s)$; for each NF $n \in N_s$, the NS identifies all the VNFs in F of the same type as n and to which n can be mapped. For each of those VNFs, the NS s creates a node $n' \in N'_s$ representing them. Each node $n' \in N'_s$ will be identified by a tuple $< f, a^f_{n's}, t^f_{n's}, p^f_{n's} >$ where $f \in F$ is the VNF instance which n' represents, $a^f_{n's}$ is the arrival time of the traffic of the NS s to n', $t^f_{n's}$ denotes the start processing time of the traffic of s by n', $p^f_{n's}$ is the processing delay of the traffic of s on n' (Eq.(4.2)). The created nodes in N'_s are interconnected by directed virtual links $e' \in E'_s$ showing the NFs ordering required by the NS NF-FG. Each virtual link $e' \in E'_s$ represents the physical route between the VNFs depicted by the connected nodes $n' \in N'_s$. For simplicity, we consider that the shortest path between each two consecutive NFs in the NF-FG is used to route the traffic of the NS. Thus, we identify each virtual link $e' \in E'_s$ by a tuple $< L_{e's}, a_{e's}, t_{e's}, d_{e's} >$ where $L_{e's}$ depicts the set of physical links to which e' is mapped, that is, the shortest path route between the source $o(e')$ and the destination $d(e')$ NFs of e'. $a_{e's}$ is the arrival time of the traffic of s to the virtual link $e' \in E_s$, $t_{e's}$ denotes the start transmission time of the traffic on e' and $d_{e's}$ depicts the transmission delay on e' (Eq.(4.1)). It is worth noting that the arrival time to a node n' is calculated as the sum of the start transmission time and the transmission delay on its precedent virtual link (Eq.(4.28)). Similarly, the arrival time on a virtual link e' is calculated as the sum of the start processing time and the processing delay on its precedent NF (Eq.(4.29)).

$$a^f_{n's} = t_{e's} + d_{e's} \text{ where } n' = d(e') \tag{4.28}$$

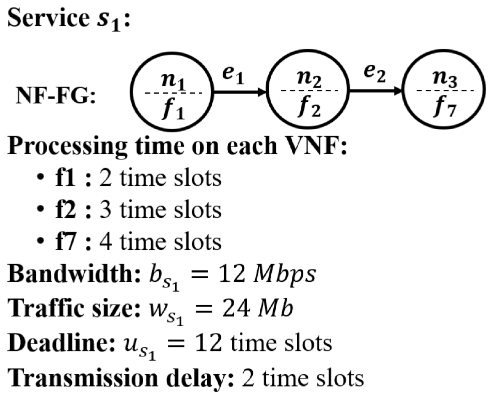
$$a_{e's} = t^f_{(n'-1)s} + p^f_{(n'-1)s} \text{ where } (n'-1) = o(e') \tag{4.29}$$

Note 4.1. *If a NF $n \in N_s$ can be mapped to a certain number g of VNF instances, then g nodes will be added to N'_s, each representing a VNF instance. Given that the arrival and start processing times can differ on each of the g nodes, the arrival time to the next NF in the chain will also differ. Thus, if the successor NF $(n+1) \in N_s$ can be mapped to a certain number x of VNF instances, we create $g * x$ nodes $\in N'_s$ where each of the g nodes is connected to each of the x nodes.*

Further, to ensure the connectivity of its graph, each NS adds a virtual source and destination nodes where the source node is connected by links with transmission delays equal to zero to all the nodes $n' \in N'_s$ that represent the first NF $n \in N_s$ in the NF-FG and the destination node is connected to all the nodes $n' \in N'_s$ that represent the last NF $n \in N_s$.



4.2.a Physical network hosting different types of VNFs.

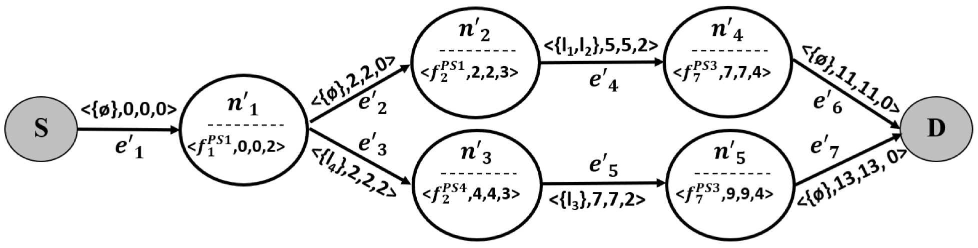


Service s_1:

Processing time on each VNF:
- **f1 :** 2 time slots
- **f2 :** 3 time slots
- **f7 :** 4 time slots

Bandwidth: $b_{s_1} = 12\ Mbps$
Traffic size: $w_{s_1} = 24\ Mb$
Deadline: $u_{s_1} = 12$ time slots
Transmission delay: 2 time slots

4.2.b Service s_1 network requirements.



4.2.c Service s_1 virtual graph $H'_{s_1}(N'_{s_1}, E'_{s_1})$.

Figure 4.2: Illustrative example showing the virtual graph $H'_{s_1}(N'_{s_1}, E'_{s_1})$ of NS s_1. Each node $n' \in N'_{s_1}$ is represented by a tuple $< f, a^f_{n's_1}, t^f_{n's_1}, p^f_{n's_1} >$ respectively denoting the VNF which n' represents, the arrival time of the traffic of s_1 to n', its start processing time and its processing delay. Similarly, each link $e' \in E'_{s_1}$ is represented by a tuple $< L_{e's_1}, a_{e's_1}, t_{e's_1}, d_{e's_1} >$ respectively indicating the set of physical links to which e' is mapped, the arrival time, the start transmission time and the transmission delay of the traffic of s_1 on e'.

As an example, we consider a physical network of 4 physical servers connected by physical links ($l \in L$). These servers are hosting VNFs of different types (i.e., $f1, ..., f7$) as shown in Fig.4.2.a. The VNFs hosted in the network may have different processing capacities, however, for the sake of simplicity, we consider in this example that VNFs of the same type have the same processing capacity. We account for a NS

s_1 requiring the NF-FG $H_{s_1}(N_{s_1}, E_{s_1})$ as depicted in Fig.4.2.b where n_1 is of type $f1$, n_2 of type $f2$ and n_3 is of type $f7$. The processing time of the traffic of s_1 on each NF is calculated based on Eq.(4.2). Similarly, the transmission delay between the NFs is calculated based on Eq.(4.1). Further, we assume that time is divided into slots (i.e., $t1, t2, t3$, etc.), each representing a duration of one second. To build the virtual network $H'_{s_1}(N'_{s_1}, E'_{s_1})$ of NS s_1, we start by identifying the VNFs of the same type of NF $n_1 \in N_{s_1}$ to which it can be mapped. That is, VNF $f1$ hosted on physical server $PS1$ which we will refer to by f_1^{PS1}. Thus, we create a node $n'_1 \in N'_{s_1}$ to represent f_1^{PS1}. Similarly, n_2 is of type $f2$ and can be mapped to either f_2^{PS1} or f_2^{PS4}. Thus, we add n'_2 and n'_3 to N'_{s_1} to represent f_2^{PS1} and f_2^{PS4} respectively and we connect n'_1 to each of them with a directed link showing the same order required in the NF-FG. n_3 is of type f_7 and can be mapped to f_7^{PS3}. Given that n_2 can be mapped to two VNFs (f_2^{PS1} or f_2^{PS4}), based on note 4.1, two nodes n'_4 and n'_5 (both representing f_7^{PS3}) are created for its successor NF n_3 connected to n'_2 and n'_3 respectively (Fig.4.2.c). Finally, two nodes S and D are added to the graph $H'_{s_1}(N'_{s_1}, E'_{s_1})$, respectively representing a virtual source and destination, are respectively connected to nodes depicting the first and last NF in the NF-FG. Each virtual link $e' \in E'_s$ is mapped to the shortest path route between its source and destination nodes. For instance, e'_3 is mapped to l_4, hence, $L_{e'_3 s_1} = l_4$. However, virtual links connected to S and D are not mapped to any physical links ($L_{e'_1 s_1} = L_{e'_6 s_1} = L_{e'_7 s_1} = \{\emptyset\}$) and no transmission delays are considered on those links ($d_{e'_1 s_1} = d_{e'_6 s_1} = d_{e'_7 s_1} = 0$). Similarly, virtual links connecting nodes (NFs) mapped to VNFs hosted on the same server are not mapped to any physical links, nor traffic transmission is needed on those links. Hence, $L_{e'_2 s_1} = \{\emptyset\}$ and $d_{e'_2 s_1} = 0$.

After building its virtual graph $H'_s(N'_s, E'_s)$, each NS s populates its nodes and links with their proper information (i.e., arrival time, start processing/transmission time, etc.) as explained above. NS s considers that it is the only NS in the network and that its traffic can be served by each VNF and transmitted by each physical link as soon as it reaches them. Thus, the traffic of s does not need to wait in the buffer of the VNF nor wait to be transmitted through the physical links to which the virtual links $e' \in E'_s$ are mapped. Hence, $a^f_{n's} = t^f_{n's}$ $\forall n' \in N'_s$ and $a_{e's} = t_{e's}$ $\forall e' \in E'_s$ in $H'_s(N'_s, E'_s)$ (Fig.4.2.c). Each path $p \in P_s$ where P_s is the set of paths in $H'_s(N'_s, E'_s)$ is a solution for the LASS problem of s. Hence, the updated graph $H'_s(N'_s, E'_s)$ reveals all the possible mapping, routing and scheduling solutions of s.

The assumption made by the NS s to populate its virtual graph (i.e., s is the only NS in the network), is not always accurate as many NSs are usually sharing the computing and network resources (i.e. VNFs, physical links bandwidths) of the physical network $G(K, L)$. Hence, in order to ensure a valid sharing of these resources where each VNF processes the traffic of one NS at a time and the physical links capacity constraints are respected, each NS needs to know the time at which it can be served by each VNF/physical link in the network based on the load on these resources. Thus, a centralized controller is needed to ensure that each NS is exploring feasible scheduling solutions. Hence, in the following we present the LASS-Game that defines the interaction between the NSs and a centralized controller in order to determine a valid solution for the LASS-problem.

4.4.2 LASS-Game Formulation

The LASS-Game is defined to allow each NS to determine a solution for its LASS problem through providing it with the needed network information in order to guide its exploration of all its possible mapping, routing and scheduling solutions. LASS-Game will help each NS s, to build and update its virtual graph $H'_s(N'_s, E'_s)$ (Section 4.4.1) based on the congestion on the shared resources (i.e., VNFs, physical links) in

$G(K, L)$ that it requires. Guiding each NS in exploring its possible schedules is of paramount importance for determining a feasible and coherent solution for the LASS problem between all NSs. Thus, we formulate the LASS-Game as a non-cooperative extensive-form game. In contrast to strategic form games where players play simultaneously, extensive-form games are used to model the sequential interactions between players [72] which yield suitable to represent and solve the LASS problem as it will allow the NSs to choose their mapping, routing and scheduling solution based on some insights about their opponents solutions, as we will explain in this section. Hence, we define the LASS-Game by the tuple $\eth(S, \mathcal{K}(V, \rho), I, A, X, \gamma)$ where:

1. S: represents the set of players in the game. We assume that each NS $s \in S$ acts as a player. Thus, in the following we use NS and player interchangeably.

2. $\mathcal{K}(V, \rho)$: is a decision tree where $V = \{v_0\} \bigcup D \bigcup T$ consists of a set V of nodes depicted by a root node v_0, a set D of decision or strategic nodes and a set T of terminal nodes. ρ is an immediate predecessor function $\rho : V \rightarrow D$.

3. $I = \bigcup_{s=0}^{S} I_s$: represents the information set of the game which includes the information that the players have at the time when they must take action. It is composed of the information sets I_s of each player s. I_s includes the physical network topology, the possible start processing time of the traffic of NS s on each VNF to which its NFs $n \in N_s$ can be mapped, in addition to the start transmission time of the traffic of s on each determined route between each two chosen consecutive VNFs and the player's actions, payoffs, moves and strategies.

4. $A = \bigcup_{s=0}^{S} A_s$: is the set of actions available during the game and consists of subsets of each player's actions A_s available for its information set $I_s \in I$. A_s includes the possible mapping, routing and scheduling solutions of s.

5. X : is the probabilities set which includes the probability x_{as} of each of the actions $a \in A$ for each player $s \in S$.

6. $\gamma = (\gamma_s)_{s\in S} : T \to \mathbb{N}^S$ is a payoff function.

Table 4.3 summarizes all the notations used in the LASS-Game.

Game notations	
$\eth(S, \mathcal{K}(V,\rho), I, A, X, \gamma)$	S: set of players. $\mathcal{K}(V,\rho)$: decision tree of $V = \{v_0\} \bigcup D \bigcup T$ nodes and precedent function ρ. $I = \bigcup_{s=0}^{S} I_s$: Set of all players information sets (I_s). $A = \bigcup_{s=0}^{S} A_s$: Set of all players action sets (A_s). X: Probability set of all the players actions. $\gamma = (\gamma_s)_{s\in S} : T \to \mathbb{N}^S$: payoff function.
Player specific notations	
$H'_s(N'_s, E'_s)$	Virtual graph of player $s \in S$. N'_s: set of nodes representing the NFs requested by s. E'_s: set of virtual links connecting the nodes in N'_s.
$< f, a^f_{n's}, t^f_{n's}, p^f_{n's} >$	Tuple representing a node $n' \in N'_s$. f: VNF instance which the node n' represents. $a^f_{n's}$: Arrival time of the traffic of s to n'. $t^f_{n's}$: Start processing time of the traffic of s by n'. $p^f_{n's}$: Processing delay of the traffic of s on n'.
$< L_{e's}, a_{e's}, t_{e's}, d_{e's} >$	Tuple representing a virtual link $e' \in E'_s$. $L_{e's}$: Set of physical links to which the virtual link e' is mapped. $a_{e's}$: Arrival time of the traffic of s to the virtual link $e' \in E_s$. $t_{e's}$: Start transmission time of the traffic of s on e'. $d_{e's}$: Transmission delay of the traffic of s on e'.
P_s	Set of paths in $H'_s(N'_s, E'_s)$.
$\sigma_s : A_s(i) \to [0,1]$	Mixed strategy of player s.
γ_s	Payoff of $s \in S$.
γ_{as}	Completion time of the schedule of player $s \in S$ when playing action $a \in A_s$.
$x_{as} \in [0,1]$	Probability of player s for playing action $a \in A_s$.

Table 4.3: Notations of the LASS-Game.

The choice of an extensive-form game allows the modeling of the consecutive interactions between players where one player cannot play before acquiring some information reflecting the updated status of the network (i.e., network topology, available VNFs, etc.) based on the moves of its precedent players. In fact, in LASS-Game, players play sequentially one after the other in order to decide on the mapping, routing and scheduling of their traffic through the NFs of their NF-FGs. Thus, at each stage of the game, each player (in the sequence) needs to choose a move from all the possible ones. It is in the best interest of each NS to get the best possible QoS and hence, complete the processing of its traffic along its NF-FG before its deadline u_s. Each player will therefore act selfishly by trying to maximize its benefits, by choosing a move that will minimize its completion time.

To be able to decide on a move, each player needs to explore the different set of actions (A_s) it can take. Here, each action depicts a decision to apply a certain mapping of its NFs, a routing of its traffic through its NF-FG and a schedule for the processing/transmission of its traffic. Thus, each path $p \in P_s$ in $H'_s(N'_s, E'_s)$ represents an action $a \in A_s$. Hence, to build and update its virtual graph, the player needs some locally available information. These information include the physical network $G(K, L)$ topology, the locations and the type of the hosted VNFs $f \in F$. In addition, the player requires some information computed based on the moves of its opponents such as an estimate of its *start* processing time on each of the VNFs its NFs can be mapped to, and its *start* transmission time on the physical routes depicting each of the virtual links $e' \in E'_s$. Thus, instead of allowing the sharing of the moves and information of the players between each other, a centralized controller is used to retain the information set I representing the complete history of all players' actions, payoffs, moves and strategies. Hence, the controller will compute and communicate to each player its start processing time on each of the VNFs its NFs can be mapped to, and its start transmission time on each of the physical routes its virtual links are mapped to. Such computation is based on a defined processing and transmission policies that we will elucidate in the following. Once a player becomes aware of its *new start* processing/transmission times, it updates its virtual graph, chooses its next move based on a mixed strategy approach and communicates it to the controller. NSs will repeatedly update and share their moves with the controller until the game ends.

4.4.3 Processing and Transmission Policies

The sharing of the physical network resources, mainly the deployed VNFs and the physical links capacity, requires some resource allocation policies in order to arrange and organize such sharing. Such organization consists of ensuring that each VNF is processing the traffic of one NS at each time slot and that the physical links capacity is respected. With the knowledge of the information set I, the centralized controller organizes the sharing of resources by applying a processing and transmission scheduling policies in order to communicate to each player currently playing, the earliest start processing/transmission time it can get to its required VNFs/physical routes. Thus, before choosing its move, each player s needs first to identify all the possible schedule lengths it can achieve given its objective in minimizing its completion time and meeting its deadline u_s. Hence, s needs to update its virtual graph $H'_s(N'_s, E'_s)$. Thus, it forwards to the controller its tentative arrival time $a^f_{n'_s}$ and processing time $p^f_{n'_s}$ on each VNF represented by a node $n' \in N'_s$ as well as its deadline u_s. With the knowledge of the list of players requesting the processing of their traffic by a certain VNF, the controller determines and communicates to each of them their start processing time on that VNF. In fact, the controller maintains an ordered list of arrival times, deadlines, and processing times for all the virtual nodes $n' \in N'_s$ constituting these moves. Let $z_{n's} \in \{0, 1\}$ depicts that node $n' \in N'_s$ is part of the move of player s (1) and (0) otherwise. The ordered list retained by the controller can then be represented by Eq.(4.30).

$$\left\{(a^f_{n's}, u_s, p^f_{n's})| \begin{smallmatrix} \forall f \in F \\ \forall n' \in N'_s : z_{n's}=1 \\ \forall s \in S \end{smallmatrix}\right\} \tag{4.30}$$

1. *Processing policy*

   For each VNF $f \in F$, the controller orders the players $(i, j \in S)$ requesting it based on the following processing policy:

$$a^f_{n'i} < a^f_{n''j} \tag{4.31}$$

$$u_i < u_j; \quad when \quad a^f_{n'i} = a^f_{n''j} \tag{4.32}$$

$$p^f_{n'i} < p^f_{n''j}; \quad when \quad u_i = u_j \tag{4.33}$$

$$i < j; \quad when \quad p^f_{n'i} = p^f_{n''j} \tag{4.34}$$

   Upon receiving a request from a player i to use a VNF f, the controller determines the position k of the player in its ordered list of that VNF. Hence, it calculates the player i's start processing time according to Eq.(4.35).

$$t^f_{n'i} = t^f_{n''(k-1)} + p^f_{n''(k-1)} \tag{4.35}$$

   Note that if, upon the arrival of the traffic of s to the VNF f, the buffer of f did not have enough capacity to store the traffic of s, that is, $\phi^\delta_f < w_s$ at $\delta = a^f_{n's}$, where ϕ^δ_f is the available buffer capacity of f at δ; the controller updates the arrival time of the traffic of s to VNF f as in Eq.(4.36) before applying the processing policy (Eq.(4.31) to Eq.(4.34)) in order to compute the start processing time of the traffic of s on f. Eq.(4.36) updates the arrival time of the traffic of s to VNF f to be equal β^f_{ns} depicting the time slot at which the buffer of f has enough capacity to store the traffic of s.

$$a^f_{n's} = \beta^f_{ns} \tag{4.36}$$

Similarly, in order to get its start transmission time on a virtual link $e' \in E'_s$, player s communicates to the controller its tentative arrival time $a_{e's}$, deadline u_s, transmission time $d_{e's}$ in addition to the physical route $L_{e's}$ to which e' is mapped. With the knowledge of all the players moves, the controller maintains an ordered list (Eq.(4.37)) of all the arrival times, deadlines, and transmission delays on the virtual links mapped to a specified physical link $l \in L$. $z^l_{e's} \in \{0,1\}$ in Eq.(4.37) depicts that e' is part of the move of player s and is mapped to the physical link $l \in (L_{e's} \subset L)$ (1) and (0) otherwise.

$$\left\{(a_{e's}, u_s, d_{e's})| \substack{\forall e' \in E'_s : z^l_{e's}=1 \\ \forall s \in S}\right\} \tag{4.37}$$

2. *Transmission Policy*

   For each physical link $l \in L$, the controller orders the players (i,j) requesting it based on the following transmission policy:

$$a_{e'i} < a_{e''j} \tag{4.38}$$

$$u_i < u_j; \quad when \quad a_{e'i} = a_{e''j} \tag{4.39}$$

$$d_{e'i} < d_{e''j}; \quad when \quad u_i = u_j \tag{4.40}$$

$$i < j; \quad when \quad d_{e'i} = d_{e''j} \tag{4.41}$$

Upon the receipt from a player i a request of its updated start transmission time on a physical route $L_{e'i}$ to which e' is mapped, the controller evaluates if all the physical links $l \in L_{e'i}$ have enough bandwidth to guarantee to s between $a_{e'i}$ and $a_{e'i} + d_{e'i}$. If any of the aforementioned physical links cannot guarantee the required bandwidth of i from the time of its arrival to the link e' until it is completely transmitted, then, the controller applies the transmission policy on each physical link $l \in L_{e'i}$ (Eq.(4.38) to Eq.(4.41)). This policy determines the earliest start transmission time of i on e' ($t_{e'i}$) at which all the links $l \in L_{e'i}$ have enough bandwidth for the traffic of i. Hence, it calculates the player i's start transmission time according to Eq.(4.42) where $t^l_{e'i}$ is the possible start transmission time of the traffic of i on l, computed based on the transmission policy, and c^δ_l represents the bandwidth available on l at time slot δ and b_i is the bandwidth demands of NS i.

$$t_{e'i} = max\ t^l_{e'i}\ :\ (c^\delta_l \geq b_i \quad {}_{\substack{\forall\ t^l_{e'i} \leq \delta < t^l_{e'i} + d_{e'i} \\ \forall\ l \in L_{e'i}}}) \tag{4.42}$$

4.4.4 Expected Utility and Nash Equilibrium

After updating its virtual graph based on the updated start processing/transmission times acquired from the controller, identifying its action set A_s and the possible completion times it can achieve, a player s has to decide on its strategy. We consider that the game is played under a mixed strategy where at each iteration, the player s needs to decide on its mixed strategy through a probability distribution over its actions, using its updated virtual graph at that iteration.

Given the set of actions A_s of a player s, let $\sigma_s : A_s(i) \rightarrow [0, 1]$ denotes the probability distribution (mixed strategy) over A_s written in terms of the actions available at each of its information sets $i \in I_s$ [73]. Therefore, in a mixed strategy σ_s of a player s, each action $a \in A_s$ is played with probability $x_{as} \in [0, 1]$. The mixed strategy game is modeled as follows:

$$max\ \gamma_s(x) = max\ -\sum_{a \in A_s} x_{as} \sum_{\sigma_{-s}} \left(\gamma_{as}^{\sigma_{-s}} \prod_{A_{s'} \in \sigma_{-s}} x_{a's'} \right)\ \forall s \in S \tag{4.43}$$

subject to:

$$\sum_{a \in A_s} x_{as} = 1\ \ \forall s \in S \tag{4.44}$$

where $\gamma_s(x)$ is the expected utility of player $s \in S$, σ_{-s} represents the mixed strategy profiles of the opponents of s, $\gamma_{as}^{\sigma_{-s}}$ is the completion time of player s when selecting action $a \in A_s$ as its strategy given those of its opponents (σ_{-s}) and $x_{a's'}$ is the probability of player $s' \in S \backslash \{s\}$ for selecting action $a' \in A_{s'}$ as its strategy. Note that the completion time of an action $a \in A_s(i)$ can be determined as the finish processing time on the last node in the path $p \in P_s$ (in the virtual graph $H'_s(N'_s, E'_s)$ of s) associated with a and can be calculated as the sum of the start processing time and the processing delay on that node. Eq.(4.43) consists of maximizing the utility (schedule length) of player s. Eq.(4.44) ensures that the probabilities over all the player's actions add to 1.

Theorem 4.1. *The LASS-Game admits a sub-game perfect Nash equilibrium and has at least one mixed strategy Nash equilibrium.*

Proof. Based on the definition of the LASS-Game in Section 4.4.2, LASS-Game is an extensive-form game where players play sequentially under a mixed strategy with the objective of maximizing their payoff through minimizing their schedule lengths, or in other words, the completion times of their mapping, routing and scheduling solution (Eq.(4.43)). In addition, the LASS-Game is a finite game given that it considers a finite set S of players, where each player $s \in S$ has a finite set of actions A_s which yields the set of paths P_s in its virtual graph $H'_s(N'_s, E'_s)$. Thus, each player admits a finite set of strategies. Further, the LASS-Game is played for a finite number of iterations. Furthermore, it is a game with perfect information as all the players are perfectly informed of all the preceding moves and the game structure ($\eth(S, \mathcal{K}(V,\rho), I, A, X, \gamma)$). In fact, while at the beginning of the game, each of the players $s \in S$ builds its virtual graph without the knowledge of the congestion level on the VNFs and on the physical links it requires from the physical network; in subsequent iterations, each of the players is communicated with its start processing/transmission time on each of its requested resources. Such start processing/transmission time is computed by a central controller which is perfectly informed of all the information set I, the actions, the moves and the payoffs of the players, in addition to the status of the network $G(K,L)$. Hence, we have shown that the LASS-Game is a finite, extensive-form game with perfect information, thus, it admits a sub-game perfect Nash equilibrium (Kuhn's theorem) [72, 74]. Finally, the LASS-Game posses an equivalent normal-form strategic game, and hence based on Nash theorem [72], it has at least one mixed strategy Nash equilibrium. ∎

In order to determine the sub-game Nash equilibrium, backward induction is carried out [75, 72]. It consists of starting at the end of the game tree $\mathcal{K}(V,\rho)$, and reasoning backward up the tree by solving for the optimal behavior at each node. In other words, it identifies the equilibrium in the bottom most tree, and adopt these as one moves up the tree. Thus, Eq.(4.45) is used at each node of the game tree to determine the path to the root, and hence, to specify the sub-game Nash equilibrium where no player has an incentive to deviate and change its strategy anymore as it will not increase its payoff. Note that in Eq.(4.45), σ_s^* is the best mixed strategy of s and σ_{-s}^* illustrates the optimal strategies of its opponents.

$$\gamma_s(\sigma_s^*, \sigma_{-s}^*) \geq \gamma_s(\sigma_s, \sigma_{-s}^*) \quad \forall s \in S \tag{4.45}$$

4.4.5 Price of Anarchy

Players of the LASS-Game behave selfishly with the objective of achieving the lowest schedule length in order to meet their deadlines and get admitted to the network. Such selfish behavior can lead to sub-optimal solutions provided by the LASS-Game in comparison to a central approach such as a MILP method (e.g., LASS-MaxAdmission (Section 4.3)). This is because the optimal solution provided by a MILP does not account for maximizing the benefit of each NS, but rather guarantees the best social welfare. Hence, in order to evaluate the performance of the LASS-Game solution, we determine the Price of Anarchy of the LASS-Game (PoA($\Game$)) by considering the latency experienced by each player on each of the resources it requires (i.e., VNFs, physical links). Hence, the PoA($\Game$) is defined as the ratio of the latency experienced by all the players at the Nash equilibrium (ζ_{eq}) over the optimal latency (ζ_{opt}) obtained by a central method (Eq.(4.46)).

$$PoA(\Game) = \frac{\zeta_{eq}}{\zeta_{opt}} \tag{4.46}$$

Thus, we evaluate the upper bound of the PoA($\Game$) by showing that the LASS-Game is a congestion game where each player $s \in S$ admits a linear latency function of the form $g_s(y) = a_s y + o_s$ where a_s and o_s are non negative real numbers.

Theorem 4.2. *The price of anarchy of the LASS-Game PoA($\Game$) is upper bounded by* $\frac{1}{2}(3 + \sqrt{5})$.

Proof. We first show that every player $s \in S$ admits a linear latency function depicting the latency it experiences at each VNF f, that we denote by ζ_{fs}, and at each virtual link e' (i.e, set of physical links to which the virtual link e' is mapped), which we represent by $\zeta_{e's}$, that it requires at the equilibrium.

Defining $\zeta_{e's}$: Let $E' = \cup_{s \in S} E'_s$ be the set of virtual links of all the players' virtual graphs ($H'_s(N'_s, E'_s)$) (Section 4.4.1). We define $\jmath(l) = \{s|l \in L\}$ to be the set of players, using at the equilibrium, a virtual link $e' \in E'$ mapped to a physical link $l \in L$. The load on $l \in L$ is depicted by $\alpha_l = \sum_{s\in\jmath(l)} b_s$ where b_s is the bandwidth required by s. Let $\mu_{e's} \in \{0,1\}$ denotes that the virtual link $e' \in E'_s$ is selected to route the traffic of s at the Nash equilibrium (1) and (0) otherwise. The expected latency experienced by player s on e' at the equilibrium can then be determined based on Eq.(4.47), where $g_l(\alpha_l) = a_l\alpha_l + o_l$ is a linear latency function of the load on the physical link $l \in L$ and a_l and o_l are real numbers.

$$\zeta_{e's} = E[\sum_{l\in L_{e's}} g_l(\alpha_l)|\mu_{e's} = 1] = \sum_{l\in L_{e's}} E[g_l(\alpha_l + (1-\mu_{e's})b_s)] \tag{4.47}$$

Defining ζ_{fs}: Let $N' = \cup_{s\in S} N'_s$ be the set of virtual nodes of all the players' virtual graphs ($H'_s(N'_s, E'_s)$). As each of the virtual nodes in N' is mapped to exactly one VNF $f \in F$, we define $\jmath(f) = \{s|f \in F\}$ to be the set of players using, at the equilibrium, the VNF $f \in F$. The load on $f \in F$ is depicted by $\alpha_f = |\jmath(f)|p_f$ where p_f is the processing capacity of f that is guaranteed to s. Let $\mu_{fs} \in \{0,1\}$ denotes that the VNF $f \in F$ is selected to process the traffic of s at the Nash equilibrium (1) and (0) otherwise. The expected latency experienced by player s on f at the Nash equilibrium can then be determined based on Eq.(4.48), where $g_f(\alpha_f) = a_f\alpha_f + o_f$ is a linear latency function of the load on the VNF $f \in F$ and a_f and o_f are real numbers.

$$\zeta_{fs} = E[g_f(\alpha_f)|\mu_{fs} = 1] = E[g_f(\alpha_f + (1-\mu_{fs})p_f)] \tag{4.48}$$

Let $F_s \subset F$ be the set of VNFs which will process the traffic of s at the equilibrium. Let $E''_s \subset E'_s$ be the set of virtual links mapped to the physical links that will route the traffic of s at the Nash equilibrium. The latency function of player s at the Nash equilibrium that we denote by ζ_s, can then be written as in Eq.(4.49) where $\zeta_{e''s}$ and

ζ_{fs} are as defined earlier. It is linear as it is the sum of two linear functions ($\zeta_{e''s}$ and ζ_{fs}).

$$\zeta_s = \sum_{f \in F_s} \zeta_{fs} + \sum_{e'' \in E''_s} \zeta_{e''s} \tag{4.49}$$

Similarly, the optimal latency function of player s denoted by ζ^*_s can be defined by Eq.(4.50) where ζ_{f^*s} and $\zeta_{e''^*s}$ respectively represent the optimal latencies experienced by s on VNF f and on the physical route to which e'' is mapped. $\zeta_{e''^*s}$ and ζ_{f^*s} can be respectively defined based on Eq.(4.47) and Eq.(4.48) by replacing e' by e''^* and $L_{e's}$ by $L_{e''*s}$ in Eq.(4.47) and replacing f by $f^* \in F^*_s$ in Eq.(4.48); where $e''^* \in E''^*_s$ denotes an optimal virtual link and $f^* \in F^*_s$ represents a VNF selected to process the traffic of s at the optimal solution.

$$\zeta^*_s = \sum_{f^* \in F^*_s} \zeta_{f^*s} + \sum_{e''^* \in E''^*_s} \zeta_{e''^*s} \tag{4.50}$$

Given that in the LASS-Game, each player acts selfishly to minimize its schedule length, the latency it experiences at the Nash equilibrium is always less than that experienced at the optimal solution where all players collaborate towards achieving the social welfare as denoted in Eq.(4.51).

$$\zeta_s \leq \zeta^*_s \tag{4.51}$$

Hence, following the derivation in [76], it can be shown that the $PoA(\eth)$ can be represented by Eq.(4.52).

$$PoA(\eth) = \frac{\sum_{\epsilon \in L \cup F} a_\epsilon E[\alpha^2_\epsilon] + o_\epsilon E[\alpha_\epsilon]}{\sum_{\epsilon \in L \cup F} (a_\epsilon \alpha^*_\epsilon + o_\epsilon)\alpha^*_\epsilon} \tag{4.52}$$

We have proved that the LASS-Game admits a mixed strategy Nash equilibrium (Theorem 4.1) and a player specific linear latency function (Eq.(4.49)), hence, its

$PoA(\eth)$ is upper bounded by $\frac{1}{2}(3+\sqrt{5})$ as explained in [76, 77]. Thus, Eq.(4.53) holds which completes the proof.

$$PoA(\eth) = \frac{\sum_{\epsilon \in L \cup F} a_\epsilon E[\alpha_\epsilon^2] + o_\epsilon E[\alpha_\epsilon]}{\sum_{\epsilon \in L \cup F} (a_\epsilon \alpha_\epsilon^* + o_\epsilon)\alpha_\epsilon^*} \leq \frac{1}{2}(3+\sqrt{5}) \tag{4.53}$$

■

4.4.6 Best Response Algorithm

The existence of a mixed strategy Nash equilibrium (Theorem 4.1) is an interesting and important result. Reaching such equilibrium requires calculating the probability distribution σ_s for each player $s \in S$ through solving the LP depicted by Eqs.((4.43), (4.44)). The complexity of this LP grows with the number of players and the number of possible actions to be taken by each player (number of nodes in $\mathcal{K}(V, \rho)$). In fact, it requires the knowledge of the completion time ($\gamma_{as}^{\sigma_{-s}}$) of player s when selecting an action a in response to the actions of its opponents. With a high number of players, possible mapping, routing and scheduling solutions for each; the combination of actions taken by the players can be very large, and hence the calculation of all the possible values of $\gamma_{as}^{\sigma_{-s}}$ is hard. This, in fact, makes the problem of calculating the mixed strategy Nash equilibrium hard to solve through the LP (Eqs.((4.43), (4.44)). Thus, in the following, we propose a time-efficient algorithm to find approximate equilibrium solutions by simplifying the calculation of the probability distribution of each player and the selection of its strategy. Fig.4.3 details the steps of the best response algorithm.

At the beginning of the game, each player $s \in S$ builds its virtual directed graph ($H'_s(N'_s, E'_s)$) by considering that any requested VNF/physical link can be guaranteed directly to it with no waiting delays. The players start by playing sequentially one

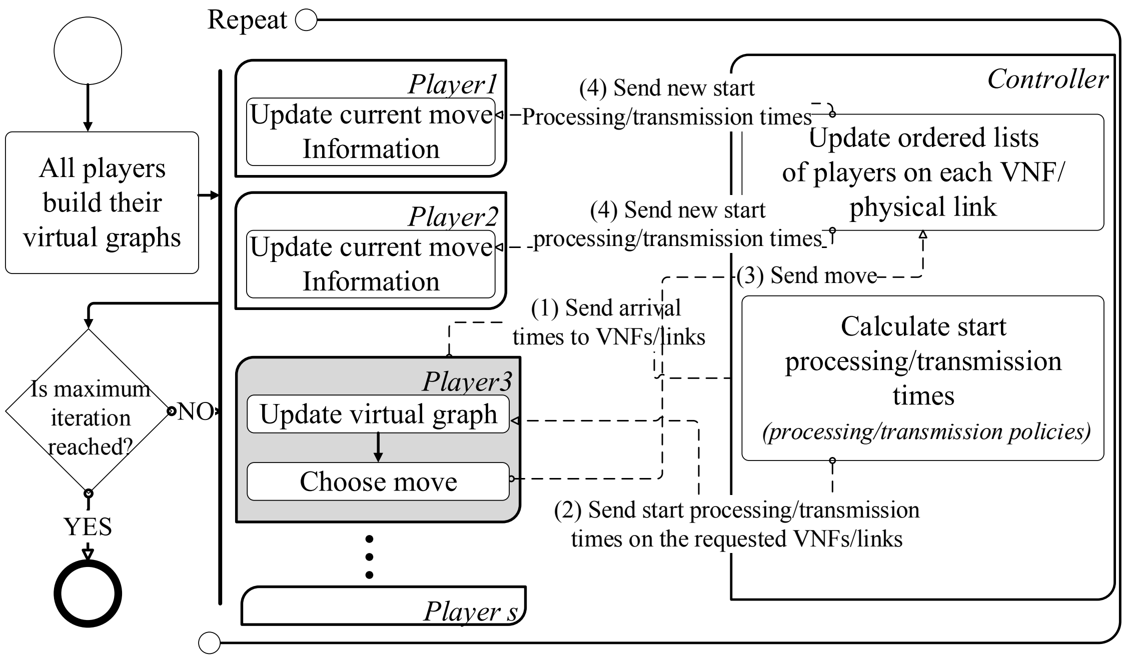

Repeat
All players build their virtual graphs
Is maximum iteration reached?
NO
YES
Player1
Update current move Information
Player2
Update current move Information
Player3
Update virtual graph
Choose move
Player s
Controller
Update ordered lists of players on each VNF/ physical link
Calculate start processing/transmission times
(processing/transmission policies)
(1) Send arrival times to VNFs/links
(2) Send start processing/transmission times on the requested VNFs/links
(3) Send move
(4) Send new start Processing/transmission times
(4) Send new start processing/transmission times

Figure 4.3: LASS-Game flowchart (player 3 is playing).

after the other. Each player $s \in S$ updates the weights of the nodes and virtual links in its virtual graph $H'_s(N'_s, E'_s)$ with the help of the controller and based on the processing and transmission policies as explained in Section 4.4.2 and Section 4.4.3. Once its virtual graph is updated, NS s chooses a move and shares it with the controller.

In order to choose its move, NS s needs to calculate its probability distribution as explained in Section 4.4.4. Given the complexity and non-scalability of the LP depicted by Eqs.((4.43), (4.44)) designated for this purpose, we develop a heuristic approach to be applied by each player in order to calculate its probability distribution (σ_s). Motivated by the fact that each player aims at minimizing its completion time, higher probability should be assigned to the actions depicting the lowest completion time. Further, given that Eq.(4.44) should be respected, each player will define the probabilities by dividing the completion time of each action over the sum of completion times of all the available ones ($\frac{\gamma_{as}}{\sum_{a \in A_s} \gamma_{as}}$). The player will then order the calculated probabilities in a descending manner and respectively assign them to the actions ordered in ascending form of their completion times. Using randomness and based on the defined probability distribution, the player s will then choose its move and sends it to the controller. Upon receiving the move of s, the controller updates the ordered list of the players on each VNF and each physical link, and thus, computes and communicates the new start processing/transmission times of the players who already played before s on each of their requested VNFs and links. These latter update the nodes and physical links information (arrival, start processing/transmission time) in their selected move accordingly. This update is of high importance in order to maintain a valid and coherent schedule for all the players at each iteration where only one player can use a VNF at a time and the physical links capacity should remain respected. The best response algorithm is terminated after a maximum number of iterations is reached. The number of iterations can be accordingly fine-tuned to provide a trade-off between the solution quality and the computation time. At the end of the game, players that were able to meet their deadlines can be identified by simply comparing their deadlines to their payoffs.

4.5 Performance Evaluation

We evaluate the performance of our LASS-MaxAdmission (Section 4.3) against our game theoretic approach LASS-Game (Section 4.4), the LASS-MinSumCT and the LASS-MinMaxCT (Section 4.3). In addition, we compare our proposed methods against the state of the art *Tabu-Search for NF Mapping and Scheduling (TS-NFMS)* [38]. To have a valid comparison, we add a shortest path routing to the TS-NFMS. Even though these methods have different objectives, it is interesting to compare them in order to understand the behavior of our hybrid approach (LASS-Game) against other centralized solutions which affect the schedule length of the NSs differently. Throughout our numerical study, we consider network slices of different sizes where the VNFs are randomly placed and have varying processing ([300-500] processing units) and buffer ([4000-6000] units) capacities. We use sets of NSs randomly generated of varying traffic ([500-1500] units) and bandwidth requirements ([300-500] bandwidth units) and demanding each varying number of NFs ([3-5] NFs). We account for different deadline settings for the generated NSs; *superTightDeadline*, *tightDeadline* and *relaxedDeadline* to be respectively equal to 4/3, 3/2 and 2 of the sum of their processing and transmission delays without considering any waiting delays. All our numerical evaluations are averaged over 5 sets and presented with 95% confidence interval. They are conducted using Cplex version 12.4 to solve the MILPs on an Intel core i7-4790 CPU at 3.60 GHZ with 16 GB RAM.

4.5.1 Offline Scheduling

4.5.1.1 LASS-Game vs Centralized Methods

We first evaluate the LASS-Game against the other mentioned methods under an offline scenario where the NSs are known a priori. Hence, we consider a slice hosted in a small test mesh network consisting of 4 physical servers hosting 7 VNFs of 5 different types. Each server is connected to a switch and the switches are interconnected by 5 links of 1000 bandwidth units each. We run our tests over sets of NSs of different size. Their deadlines are generated based on the superTightDeadline setting.

1. *Execution Time*

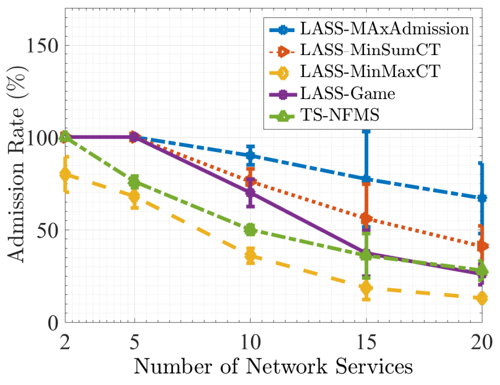
   In order to study the scalability of our proposed methods, we compare their execution times. Table 4.4 shows that when the number of NSs is small (2 NSs), all methods were able to find a solution in around 1 second. However, as the number of NSs increases, the LASS-MaxAdmission, the LASS-MinMaxCT and the LASS-MinSumCT become much harder to solve and their runtime increases exponentially to reach around 0.85 minutes, 6 and 42 hours respectively for 20 NSs. Further, it is important to note that LASS-Game is more scalable than the MILPs as it does not invoke any execution of complex mathematical formulation. However, its runtime remains higher than that of the TS-NFMS. While the TS-NFMS is a meta-heuristic which, similarly to the LASS-Game can invoke multiple iterations to reach a good solution, we notice that it remains more scalable than the LASS-Game as the latter requires building and updating the players virtual graphs at each iteration which is time consuming. However, one advantage of the LASS-Game in comparison to TS-NFMS is that the quality of the solution it provides is guaranteed and bounded (Section 4.4.5) which is shown by the higher admission rate it offers as we highlight in the following.

Number of NSs	**LASS-MaxAdmission**	**LASS-MinSumCT**	**LASS-MinMaxCT**	**LASS-Game**	**TS-NFMS**
2	1201.4	912.6	869.2	41.2	8.6
5	5811.4	4632.6	6232.4	322.6	4.2
10	18113	185015.4	663246	5949.4	5.4
15	29806	5725331.4	5346004	10070.8	7.4
20	51147.4	151764394.8	21320066.4	14312.4	69.8

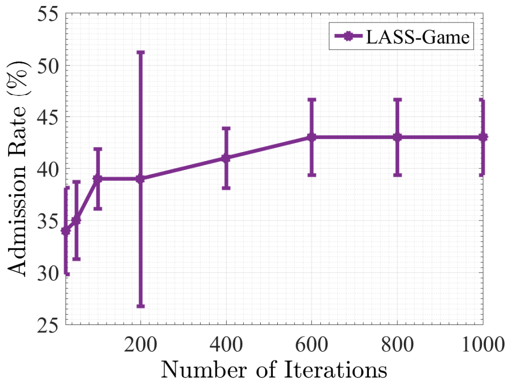
Table 4.4: Execution time (ms) per number of NSs.

2. *Admission Rate*

The admission rate is an important metric that depicts the quality of the presented mapping routing and scheduling solution as it has a direct impact on the revenue generated by the service provider. A higher admission rate yields a higher revenue. Even though the presented methods have different objectives, we observe that it is interesting to compare them in terms of admission rate. Note that NSs admitted by LASS-MinSumCT, LASS-MinMaxCT are easily identified by comparing their deadlines to their completion times provided by these MILPs. Thus, we present the admission rate of the various proposed methods in Fig.4.4.a as we vary the number of NSs.



4.4.a Admission rate per number of NSs.



4.4.b Admission rate per number of iterations.

Figure 4.4: Offline scheduling numerical evaluation.

The increase in the number of NSs increases the sharing of the VNFs. Hence, many NSs will suffer from extra waiting delays to be served by these shared VNFs which will lead to an increase in their completion time and hence, the risk of missing their deadlines. This explains the fact that the admission rate decreases with the increase of the number of NSs using the five presented methods in Fig.4.4.a. It is clear from Fig.4.4.a that LASS-MaxAdmission outperforms the other methods in terms of admission rate which is expected given its objective. Even though LASS-MinSumCT, LASS-MinMaxCT, LASS-Game and TS-NFMS do not aim at maximizing the number of admitted NSs, we notice, however, that LASS-MinSumCT depicts an average of 12.26% of gap, the LASS-Game delineates a gap of 20.2% while the TS-NFMS presents a gap of 28.86% in comparison to the optimal admission rate provided by LASS-MaxAdmssion. LASS-MinMaxCT presents the worst admission rate in comparison to all the other methods with an average gap of 43.73% in comparison to LASS-MaxAdmssion. This is related to the fact that LASS-MinMaxCT will force some NSs to sacrifice some of their desired resources (VNFs, bandwidth and time slots) to others with higher completion times. This will increase the probability of some NSs to miss their deadlines at the expense of minimizing the high completion times of other NSs. In contrast to LASS-Game where players play sequentially and selfishly to reserve the VNFs which will minimize their completion time without any consideration of the impact of their choice on the other players' (NSs) completion times and incurred delays, LASS-MinSumCT is a centralized solution where the NSs will cooperatively adjust their completion times such that their sum is minimized. Hence, the chance of a NS to meet its deadline is higher in LASS-MinSumCT which explains the reason behind having LASS-MinSumCT outperforms LASS-Game by an average of 7.93%. Further, even though the LASS-Game and the TS-NFMS both aim at minimizing the schedule length of each of the NSs, we notice that the LASS-Game outperforms the TS-NFMS in terms of admission rate by 8.66% given that the TS-NFMS considers reducing the flow time [2] through migrating the VNF with the highest flow time, while the LASS-Game explores different mapping options for all the NFs in the forwarding graph of the NS. In addition, we notice that as the number of NSs increases (15, 20 NSs), the LASS-Game presents an admission rate similar to that provided by TS-NFMS as it becomes harder to reach the equilibrium. However, our tests show that the LASS-Game can always outperform the TS-NFMS as we increase its number of iterations.

3. *Stability of the LASS-Game*

In order to study the stability of the LASS-Game, we present in Fig.4.4.b the average maximum admission rate provided by this method as we increase the number of iterations. Here, we consider 25 VNFs of 7 different types deployed in a network of 20 physical servers and 20 links connecting them of 2000 bandwidth

[2]Flow time: Time between the completion of the processing of the traffic by a preceding function and the start of its processing by the current one.

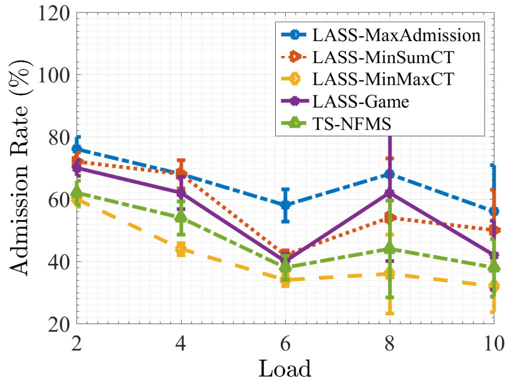
units each.We account for 10 NSs having super tight deadlines. Clearly, as we increase the number of iterations, the maximum admission rate increases given that the NSs are able to explore a wider pool of actions as they are provided by new information sets. Hence, they will be able to choose better strategies which will improve their completion time. However, we notice that after 600 iterations (Fig.4.4.b), LASS-Game starts to converge around the best solution in terms of admission rate to stabilizes at 43%. We note that the number of iterations does not highly affect the admission rate (i.e., 20.9% of admission rate increase between 25 and 600 iterations), however, our studies show that the execution time of the game increases with the number of iterations. Hence, there exists a trade-off between the quality of the solution and the computation time needed to obtain it.

4.5.2 Online Batch Scheduling

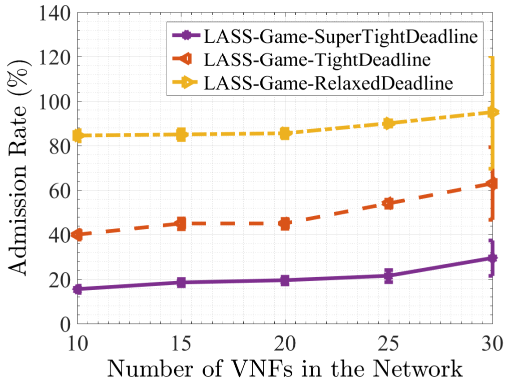
Given that in practice, the traffic is dynamic with random arrival and departure of the NSs, we study in this section the performance of the LASS-Game by considering an online batch scheduling where the evaluated methods are run periodically on a batch of NSs. Thus, in the following, we consider sets of NSs randomly generated as described earlier, following a Poisson arrival of varying rate. We fix the period length to 5 time slots which will allow the division of each set of NSs into batches of different size.

1. *Varying the arrival rate*

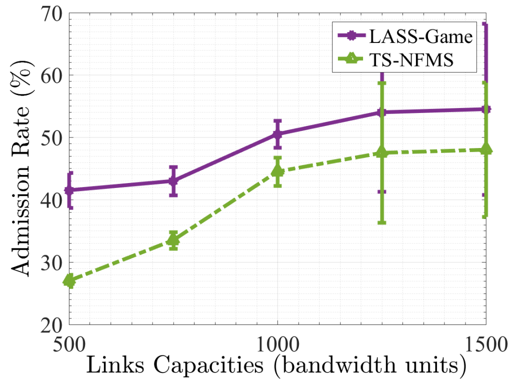
   We consider a network slice hosted on the same small test mesh network of 4 physical servers and 7 links of 1000 bandwidth units described in Section 4.5.1. We account for sets of 10 NSs of super tight deadlines. We evaluate in Fig.4.5.a the admission rate provided by the different methods under varying



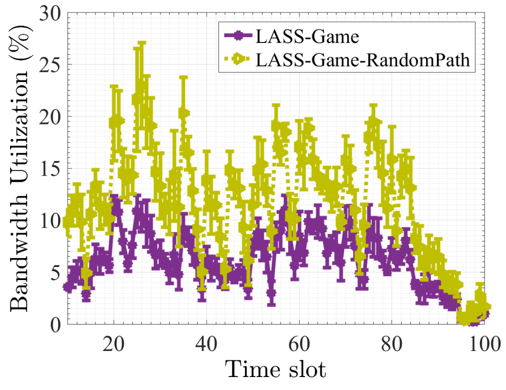
4.5.a Admission rate per varying arrival rate.



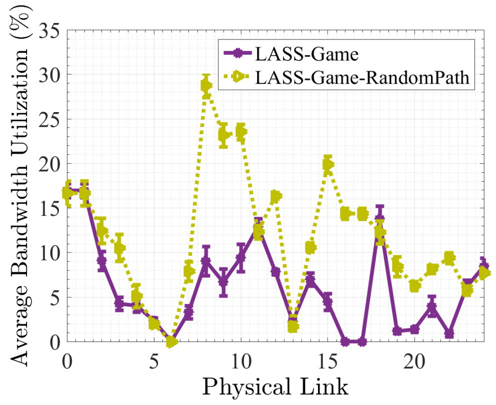
4.5.b Admission rate per varying VNF number.



4.5.c Admission rate per varying links capacity.



4.5.d Average bandwidth utilization per time slot.



4.5.e Average bandwidth utilization per link.

Figure 4.5: Online batch scheduling numerical evaluation.

arrival rate. Fig.4.5.a shows that as we increase the arrival rate, the admission rate decreases given that the number of NSs in each batch will increase. The increase of the number of NSs in a batch contributes to extra waiting delays given that more congestion will be experienced on the shared resources (i.e., VNFs, physical links). Hence, many of the NSs will miss their deadlines and get rejected from the network. However, we notice that at an arrival rate 8, the admission rate of all the methods increased in comparison to that observed at rate 6. Such increase shows that as the number of NSs in a batch increases, the utilization of the resources becomes more optimized and hence these resources can be freed earlier to be used by the NSs of the next batch, which positively impacts the admission rate. Finally, we note that the LASS-Game performed better than LASS-MinSumCT at rate 8 as the set of NSs that were admitted by the LASS-Game in the first few batches are different from those admitted by LASS-MinSumCT and achieved different schedule lengths. This, in fact varied the network state (i.e., occupied/available resources) between the batches in each of the two methods and resulted in a better resource utilization provided by the LASS-Game which leads to a higher admission.

In the remainder of our numerical evaluation, we consider a slice hosted on a larger mesh network of 20 physical servers hosting 25 VNFs (unless stated otherwise) of 7 different types. Each server is connected to a switch and the switches are interconnected by 25 links of 2000 bandwidth units (unless stated otherwise) each. We consider 40 NSs of tight deadlines following a Poisson distribution of an arrival rate of 5.

2. *Varying the number of VNFs*

   NSs sharing the same VNFs suffer from waiting delays since each VNF can process the traffic of one NS at a time. Thus, increasing the number of VNFs in the network slice will increase the strategy pool of the players as they will be provided with more mapping options for their NFs. Given their interest in reducing their completion times, NSs attempt to map their NFs to the least loaded VNFs able to provide them with the earliest service time. This reduces the sharing of VNFs and lead to an increase in the admission rate as depicted in Fig.4.5.b. Fig.4.5.b also shows the admission rate increase with the relaxation of the NSs deadlines. Such increase is expected given that the NSs can tolerate more waiting delays with the increase of their deadlines. Note that, LASS-Game-SuperTightDeadline, LASS-Game-TightDeadline and LASS-Game-RelaxedDeadline correspond to tests on NSs generated with super tight, tight and relaxed deadlines respectively.

3. *Varying the capacity of the physical links*

   Fig.4.5.c depicts the increase in the admission rate as the capacity of the physical links increases. In fact, increasing the links capacity allows more NSs to share the same link at the same time which will reduce their schedule lengths and hence, allow them to meet their deadlines. Fig.4.5.c also shows that the LASS-Game outperforms the TS-NFMS by an average of 17.65% in terms of admission rate given that it allows each NS to explore more NF mapping possibilities and hence, more diverse physical routes, thus, reducing the number of NSs contending for the same resources which will decrease the waiting delays they experience.

4. *Evaluating the bandwidth utilization*

   As the LASS-Game applies the shortest path route to transmit the traffic of the NSs between each two consecutive VNFs, we study the impact of the routing path on the network bandwidth utilization by evaluating the *LASS-Game under a random path (LASS-Game-RandomPath)* routing approach. Hence, we first compare the total average bandwidth utilized in the network at each time slot during the period between time slots [10-100]. Fig.4.5.d depicts that the LASS-Game-RandomPath utilizes more bandwidth than the LASS-Game given that the random path route often requires the traffic to pass through a higher number of physical links which will increase the consumed bandwidth. In addition, Fig.4.5.d shows that the bandwidth utilization varies between the time slots as some NSs arrive to the network while others leave it, respectively consuming and releasing the network bandwidth. Further, we present in Fig.4.5.e the average bandwidth utilized on each physical link in the network during the same aforementioned period. Fig.4.5.e shows that 14 links experience more than 10% of bandwidth utilization with the LASS-Game-RandomPath in comparison to only 4 links in LASS-Game.

4.6 Conclusion

We studied in this chapter the LASS problem to provide a mapping, routing and scheduling solution for ultra-low latency NSs requiring their traffic to be processed by a chain of NFs within a guaranteed end-to-end delay. We formulate this problem as a MILP under different objectives (i.e., maximize the admitted NSs, minimize the sum of completion times of all NSs, minimize the maximum completion time of the NSs). Given its complexity, we proposed a novel game theoretic approach, the LASS-Game to orchestrate the traffic processing and transmission schedules of the NSs. To the best of our knowledge, our non-cooperative extensive-form game based on a mixed strategy selection is the first hybrid approach to solve this problem. Our numerical evaluation shows that the LASS-Game is much more scalable than the presented MILP methods as it is able to provide a solution in the order of seconds while outperforming the LASS-MinMaxCT by 23% in terms of admission rate and being 5.7% away from the admission rate provided by the LASS-MinSumCT method.

Our studies also depict that LASS-Game is able to provide a good quality solution in very few iterations, hence, realizing a trade-off between the quality of the solution and the computation time.

Chapter 5

Low-Latency IoT Services in Multi-access Edge Computing [1]

The promise of 5G networks in enabling ultra-low latency services is faced by some challenges related to few shortcomings of IoT devices. The limited battery, computing, and storage capacities of IoT devices restrict their capability to locally process and compute the tasks of their running IoT applications enabling such services. Thus, to overcome these limitations, the MEC paradigm has been proposed to facilitate the access to advanced computing capabilities at the edge of the network, in close proximity to these end devices, thereby enabling a rich variety of latency sensitive services demanded by various emerging industry verticals. IoT devices, being highly ubiquitous and connected, can offload their computational tasks to be processed by applications hosted on MEC servers. Thus, given the heterogeneous requirements of the offloaded tasks (computing, latency requirements, etc.) and the importance of efficient resource utilization of MEC capabilities, we extend in this chapter our previous work to the network edge by studying the joint problem of deciding on the

[1] This chapter has been accepted for publication in IEEE Journal on Selected Areas in Communications - Special Issue on Network Softwarization & Enablers [78].

task offloading (tasks to application assignment) and scheduling (order of executing them) strategy. Unlike our previous work where the computing resources of the VNFs were pre-determined, we exploit in this work the scheduling problem under undetermined computing resources of the IoT applications which makes the problem much more interesting given the direct impact of the computing resources allocation on the scheduling decisions.

5.1 Introduction

5.1.1 Overview

The expectations towards a premium QoE are widely increasing with the recent advancement of 5G networks, paving the way towards a broad new set of services such as augmented/virtual reality, traffic safety, image and face recognition, etc. [14, 17]. Enabling proximity services with fast service delivery at anytime in crowded areas, comes hand-in-hand with the goal of 5G systems in providing user-centric QoE that includes but is not limited to ultra-low latency, ultra-high reliability and a support of 1000 times higher data volumes [14, 79]. The need to support higher data volumes is a result of the tremendous foreseen increase in the number of wearable IoT devices and mobile User Equipments (UEs) (e.g., smart phones, tablets, smart speakers, etc.). Although these UEs and IoT devices are being equipped with much more powerful CPU and greater storage capacity, they still fall short in satisfying the emerging applications which require huge processing capabilities for their data in a relatively short amount of time. Further, they suffer from limited battery life which restricts their users from executing these highly demanding applications locally [14, 17]. Hence, the idea of Mobile Cloud Computing (MCC) was introduced to allow mobile users to exploit the centralized cloud computing resources which can be accessed through the

Internet, thus, expanding the battery life of their UEs by allowing them to offload their tasks to be processed by the applications hosted in the cloud. However, MCC incur extra delays due to the time needed to transfer the data from the UEs to cloud servers which are far from the users [17].

Thus, in order to overcome the high latency incurred by the access to the central cloud, the paradigm of distributed edge computing has been introduced to enable access to computing resources deployed at the edge of the network, in close proximity of the user [15, 16, 17].

5.1.2 Edge Computing Related Concepts

Different concepts promoting the computation at the edge have been gaining interest in the past few years:

1. *Cloudlet*

   Cloudlet was introduced in 2009 by Satyanarayanan et al. [80] who define it as "*a mobile user exploits VM technology to rapidly instantiate customized service software on a nearby cloudlet and then uses that service over a wireless local area network; the mobile device typically functions as a thin client with respect to the service.*" In fact, cloudlet depicts servers or cluster of servers deployed at a single hop proximity of mobile UEs, thereby guaranteeing real-time interactive responses [15]. These servers host a set of VMs responsible of processing the tasks offloaded by different UEs [15]. Cloudlets, also known as computing "hotspots", are mostly enabled at business premises and accessed by mobile UEs through WiFi Access Points (APs) which limit the support of the UEs' mobility that is constrained by the local WiFi coverage [17, 81]. In this case, UEs may switch to mobile network to connect to a distant cloud which contributes to a degraded performance [15, 17]. Note that, as cloudlets are part of a three-tier hierarchy (i.e., mobile device - cloudlet - cloud), they can function in a standalone mode with total isolation from the cloud or can be connected to the latter [15]. They mainly target mobile offloading applications such as face recognition, image processing, video streaming applications.[14, 15].

2. *Fog computing*

   "*Fog Computing is a highly virtualized platform that provides compute, storage, and networking services between end devices and traditional cloud computing data centers, typically, but not exclusively located at the edge of network.*" is the definition of Fog computing as introduced by CISCO in 2012 [82]. Fog is an extension of the cloud computing paradigm from the core to the edge of the network; hence, it supposes the interaction of fog nodes with the central cloud [15]. Fog follows a multi-tiered architecture which deploys an intermediate fog platform between the device and the main cloud. Fog nodes, encompassing a large number of geographically wide spread edge nodes with processing power and storage resources such as switches, edge routers, access points, personal computers, etc. are deployed at different network tiers [14, 15, 81]. Fog computing is mainly identified to deliver new applications and services for IoT devices [14, 15, 81]. However, content delivery networks can also benefit from Fog computing [15, 81].

3. *MEC*

   MEC was first introduced by ETSI in 2014 under the name of Mobile Edge Computing to accelerate the advancements on edge computing in mobile networks and within the Radio Access Network (RAN). ETSI defined mobile edge computing as follows: "*Mobile edge computing provides IT and cloud computing capabilities within the RAN in close proximity to mobile subscribers*" [83]. The definition was then slightly modified to "*Edge Computing refers to a broad set* *of techniques designed to move computing and storage out of the remote cloud (public or private) and closer to the source of data*" [84] to accommodate various set of access technologies (e.g., WiFi, Long-Term Evolution(LTE), etc.) [85]. This definition motivated renaming mobile edge computing to multi-access edge computing. MEC servers can be deployed at various locations at the network edge such as the LTE macro base station (eNodeB) and function as standalone entities with no reported interaction with the cloud [15, 81]. MEC will enable a wide range of services and applications such as those related to IoT devices, augmented reality, virtual reality, etc. [14, 85].

While MEC, Fog computing and cloudlet promote access to computing resources at the edge of the network, they differ by the access technology used, the virtualization technology they employ (i.e, cloudlet uses VM, while Fog computing and MEC support VM and other virtualization technologies), the applications they target and their interaction with the central cloud [15]. Beside these differences, the main benefits of these edge computing paradigm rely in assisting UEs in processing the data of computation intensive applications by providing them access to real-time distributed computing with short response time [14, 85]. Further, they can help in analyzing collected data from geographically distributed sensors to assist machine to machine applications (e.g., vehicle to vehicle communication) in the quest of optimizing for example network traffic [85]. Finally, they offer ultra-low latency services with high bandwidth and real time access to radio and network analytics [14].

5.1.3 Motivation and Challenges

Giving UEs access to edge computing resources in order to process their tasks requires a careful allocation of these resources to the UEs. Given that the resources and the computation abilities of edge clouds are relatively constrained when compared to the

MCC, edge clouds can be backed by a remote cloud via the Internet. For instance, edge clouds can at anytime offload their demanding tasks to MCC servers through the Internet whenever they are overloaded. Further, multiple edge servers can collaborate and offload their tasks to each others (e.g., via a backhaul network) to provide mobile users with better service through balancing their workloads and sharing their resources [86]. For instance, a nearby edge server can choose to offload a task of a connected UE to another edge server, if it does not host the required application to process it, or if its deployed application instance is overloaded or cannot be allocated enough computing resources to process the task within its delay requirement. Hence, efficiently utilizing edge resources is necessary to guarantee its foreseen benefits which are intimately associated with solving the following challenges:

1. *The task offloading problem*

   The task offloading problem consists of determining the edge server to which each task should be offloaded. More precisely, it consists of associating each task to an application hosted on an edge server and able to process it.

2. *The application resource allocation problem*

   The application resource allocation problem determines the computing resources to be allocated to each application[2] deployed on an edge server in order to process its assigned tasks within their delay requirements.

3. *The task scheduling problem*

   The task scheduling problem decides on the order in which each offloaded task should be processed on the shared application while meeting its deadline.

While some work in the literature focused on determining the edge server to which each task should be offloaded and the computing resources it needs to be allocated [87,

[2]An IoT application can be hosted on a VM or a container deployed at the edge. The amount of resources of the VM/container is to be determined.

88, 89, 90, 91], others addressed the joint problem of task offloading and scheduling either through stochastic optimization [92, 93] or using algorithmic solutions [86, 94, 95]. The methods used in existing works are approximate solutions which do not explore the benefits that can be brought by enabling dynamic modifications of the computing resources allocated to the shared IoT applications and their impact on the scheduling of the offloaded tasks.

In this work, we consider a MEC system model and we target the QoE requirements of the rising 5G services of different business verticals by presenting a complete offloading scheme that accounts for a joint resource provisioning of IoT applications as well as a fine grained task scheduling to meet the delay sensitive requirements of these 5G services. We show that such computing resource provisioning directly affects the scheduling decisions and impacts the number of tasks that can be admitted to the network. We refer to this problem as the *Dynamic Task Offloading and Scheduling problem (DTOS)*. Addressing the latter is motivated by recent advancements on 5G targeting ultra-low latency uses cases, in addition to the emerging trend of network softwarization empowered by virtualization technologies that enable dynamic scaling of resources.

5.1.4 Novel Contributions

The main contributions of this chapter can be disclosed as follows:

1. We mathematically define and formulate the DTOS problem as MILP *(DTOS-MILP)*.

2. Given its complexity, we explore the *DTOS-LBBD*, a LBBD approach to efficiently solve the DTOS problem. To the best of our knowledge, we are the first to present a LBBD framework to solve this problem that is able to achieve

several order of magnitude of faster run times while providing the optimal solution. The DTOS-LBBD decomposes the DTOS problem into a master problem that performs the task offloading and the resource allocation; and multiple subproblems, each addresses the scheduling of tasks that are offloaded to a single IoT application.

3. Extensive numerical evaluations are carried out to examine the efficiency of the DTOS-LBBD compared to the DTOS-MILP in terms of runtime. In addition, valuable performance trends are explored to highlight the impact of task offloading and scheduling in meeting the diverse QoE requirements aligned with 5G vertical industries.

The remainder of this chapter is organized as follows. Section 5.2 highlights the latest work in the literature tackling edge computing challenges. The system model and the problem motivation are discussed in Section 5.3. Section 5.4 defines and mathematically formulates the DTOS problem. Section 5.5 presents and explains the various aspects of the DTOS-LBBD approach. Our numerical evaluation is depicted in Section 5.6. We conclude in Section 5.7.

5.2 Literature review

5.2.1 Joint Task Offloading and Resource Allocation

Sun et al. [89] solved the latency-aware workload offloading problem by mathematically formulating the task offloading problem under the objective of minimizing the average response time for mobile users and presenting an algorithmic approach to solve it efficiently. While Sun et al. [89] did not discuss the presence of IoT applications and their allocated resources; the limited resource pool in MEC in comparison to a centralized cloud computing motivated many work in the literature to jointly address the resource allocation and task offloading problem in the quest to efficiently utilize these resources [85].

The authors of [87] mathematically formulated the joint problem of task offloading and resource allocation in MEC where they do not only account for the computing resource allocation but also for the transmission power allocation of mobile users. Given the NP-hardness of the problem, they decompose it into a task offloading problem which they solve using a heuristic approach and a resource allocation problem which they address through convex optimization techniques. While the work in [87] accounted for the QoE of each mobile user which they characterized by their task completion time and energy consumption, their work fall short in accounting for tasks with strict delay requirement and overlook the sharing of computing resources between multiple UEs. Unlike [87], the work of [88] focused on minimizing the cost of the online resource allocation and task offloading in MEC under unpredictable resource prices and user mobility. The authors of [88] provide an online optimization-based algorithm to solve the resource allocation problem of mobile users at each time slot by considering adapting their allocated resources based on the optimal solution obtained at the previous one.

The work in [90, 91] accounted for mobile users requesting a determined type of IoT applications. With the objective of minimizing the average response time in terms of network and computing delays, the authors of [91] formulated and addressed the problem of placing IoT applications of different types on existing MEC while deciding on their computing resources and determining the tasks that will be offloaded to each of them. In their work, they specified a maximum allowable computing delay for each application. In contrast, Jia et al. [90] considered a NFV based MEC, where they solved the task offloading problem for a set of mobile users requesting a specific type of VNF within determined latency requirements. They solve this problem through reducing it to a series of minimum weight maximum matching problem in auxiliary bipartie graph. They further, address the dynamic changes of offloading request patterns (i.e, task size, VNF type) over time, and develop a prediction mechanism to release and/or create instances of VNFs. The work in [96] aimed at evaluating the impact of edge computing and its enabling technologies on the response time. Thus, the authors of [96] consider the special use case of a mobile gaming 3-D application and evaluate the response delay incurred by offloading the tasks to be processed on edge servers deployed at three different locations (local deployment, special-purpose cloud infrastructure and commercial public cloud). Their experimental evaluation shows that the location of edge servers and the virtualization technology used (i.e., container, Virtual Machine (VM), bar metal) highly impact the latency experienced. Other works on the task offloading and resource allocation problem have been reviewed in [15, 16].

5.2.2 Task Scheduling

An efficient resource utilization entails a smart orchestration of resource sharing which cannot be accomplished without a proficient scheduling of their utilization. Hence, to maximize the revenue of the infrastructure owner, Katsalis et al. [93] devised a Lyapunov optimization framework to address the VMs placement and scheduling problem while accounting for the Service Level Agreement (SLA) for time-critical services. Their scheduling approach considers the scheduling of the number and the type (small, medium, large) of VMs to deploy at each time slot for each mobile service operator based on the variability of its workload. Similarly, the authors of [92] employed a Lyapunov function to decide on the offloading schedules of tasks while stochastically maximizing the network utility under partial out-of-date network states information without any consideration of the tasks delay requirements. The authors of [94] jointly optimized the task offloading and scheduling problems along with the transmit power allocation problem. They were interested in minimizing the weighted sum of execution delays and service energy consumptions. They decomposed the problem into a task offloading and scheduling problem which they solve using an algorithmic approach with the objective of minimizing the makespan of all the jobs; and a transmit power allocation problem which they address using convex optimization. The work in [94] considers that all the jobs are sharing a single-CPU core at the edge server and overlooks the deadline requirements of the tasks.

The task offloading and scheduling problem was investigated by Wang et al. [95] as well, who presented an algorithmic solution to solve the problem while accounting for the deadline requirement of the tasks and for the scheduling of their transmission and computation. However, the authors of [95] did not account for different IoT applications hosted at the edge nor solved the application resource allocation problem. The authors of [86] addressed the task offloading problem while accounting for the possibility of dispatching jobs to a remote cloud as well as to a MEC. In addition, they presented a preemptive scheduling for the offloaded tasks with the objective of minimizing their weighted response time using an online algorithm.

5.3 System Model

The tremendous move towards smart cities is gaining momentum with the development of 5G networks. Smart cities will enable several services such as smart traffic management, security, energy efficiency and smart health care which make use of multiple IoT devices and applications. We consider a smart city Wide Area Network (WAN) (Fig.5.1), composed of a set S of cellular base stations which can be represented by either a macro cell (Evolved Node B (eNB)) or a small cell (Small

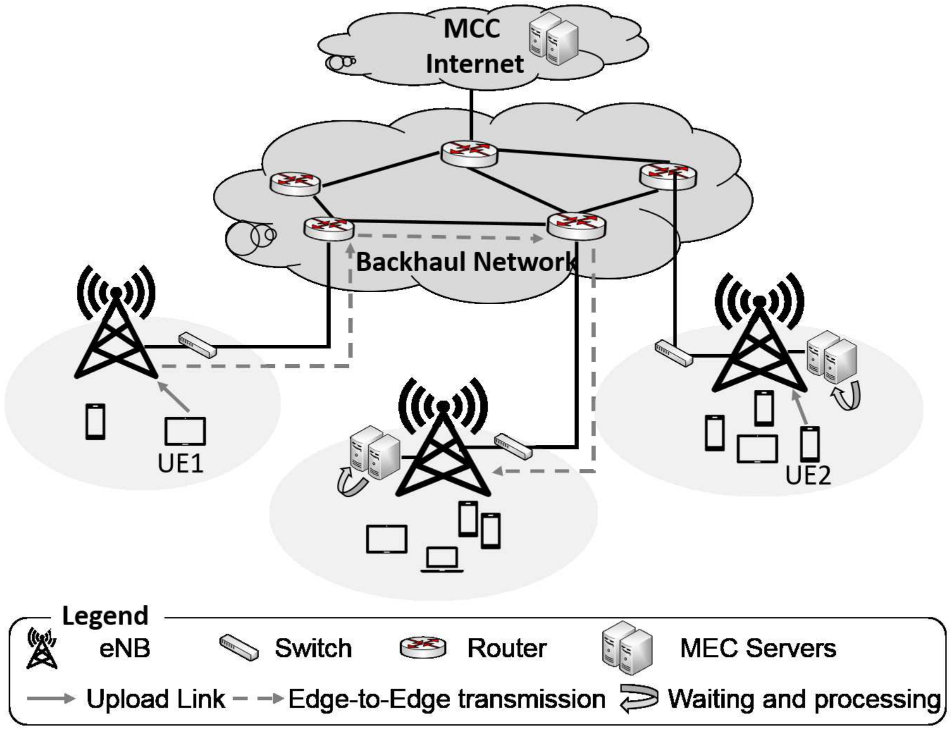


Figure 5.1: System Model.

Cell Evolved Node B (SCeNB)). For simplicity and without loss of generality, we will represent by *eNB* any type of base station. In order to enable flexible routing and communications among eNBs, we consider that the core cellular network is enabled with SDN technology consisting of an SDN controller and openFlow switches [91]. The SDN controller benefit from a global view of the network and can be used to provide some monitoring based information such as the latency experienced by a flow when transmitted between two eNBs [97]. A subset of eNBs in S are mounted with MEC servers to provide computation offloading services to the IoT users (i.e., UEs such as tablets and wearable devices) (Fig.5.1). The MEC-enabled system that we consider operates in a time-slotted structure, where we denote by $\delta \in \Delta$ a time slot.

Let M be a set of deployed MEC servers; each MEC server $m \in M$ consists of a pool of physical servers with an aggregated computing capacity c_m specified in terms of cycles/second or MHz. The MEC servers are hosting a set A of IoT applications of multiple types (e.g., face recognition, video encoding, etc.) designated to process the offloaded tasks of UEs. Each application $a \in A$ is a software which can be deployed on top of a VM or a container hosted on a MEC server $m \in M$. It is of specific type that we depict by $t_a \in T$ where T is a set of IoT application types. Like any other software, an application $a \in A$ requires some minimum system specifications to be able to operate efficiently. For simplicity and without loss of generality, we represent the minimum system requirements of an application a by the minimum processing capacity p^a_{min}, it requires to operate. However, each application can be provisioned with some computing resources p_a, that exceed its minimum requirement p^a_{min}, in order to maximize the workload it can process within a time limit. The processing capacity p_a of an application is represented in terms of cycles/second and is related to the virtual CPU (vCPU) resources (number of cores) assigned to the VM/container on top of which the application is running [98]. Further, we assume that these applications can be shared by many UEs but can process the task of one UE at a time.

5.3.1 UEs Computation Tasks

We consider a set U of UEs requesting to offload their delay-sensitive tasks to be processed by an IoT application $a \in A$ of a suitable type deployed on an edge server $m \in M$. In this work, we account for a static scenario where the set U of UEs remains unchanged during the offloading period[3]. We consider that each UE $u \in U$ has one computation task at a time. Thus, in the following we use task and UE interchangeably. We represent each task by a tuple $< t_u, \mu_u, \theta_u >$ where $t_u \in T$ depicts the type of the IoT application required to process the task of UE $u \in U$. μ_u represents the workload (cycles) required to accomplish the processing of the task of

[3]We leave the study of the dynamic scenario where mobile UEs arrive and depart dynamically during an offloading period for future work.

UE u and can be obtained by profiling the task execution [99]. θ_u denotes the latency requirements (i.e., deadline) in terms of time slots of the task of UE u. Note that, if the latter was not processed within its deadline, it will be rejected from the network.

5.3.2 Experienced Delays

Processing the offloaded tasks with respect to their latency requirements entails deciding on the MEC server to which each of the tasks should be offloaded, determining the computation resources to allocate to the IoT applications that will process the tasks, in addition to specifying the order in which the offloaded tasks should be processed by each of these applications. Solving the three aforementioned challenges highly impacts the admission of the tasks to the network as they directly affect some of the delays they experience. In the following, we summarize the delays that an offloaded task experiences.

1. *Upload delay* d^u_{up}

   The task uploading delay corresponds to the time required to transmit the task from the UE u to the serving eNB. We assume that the serving eNB $s \in S$ of each UE u is the base station with the highest received signal quality. For simplicity and without loss of generality, we assume that d^u_{up} is predefined and can be calculated based on the Signal to Interference plus Noise Ratio (SINR) as explained in [87].

2. *Edge-to-edge delay* d^u_{ee}

   Once the task of UE u is uploaded, it should be processed by an IoT application $a \in A$ of a suitable type, deployed on a MEC $m \in M$ to which the task was offloaded. It is of the best interest of u to have its task processed by the MEC attached to its serving eNB to avoid any additional network delays. However, the serving eNB may not be enabled with MEC capabilities (UE1 in Fig.5.1), or the MEC server attached to it may not be able to process the task of u within its deadline θ_u; that is, the MEC server m is not hosting an application instance a of the same type of that required by u ($t_a != t_u$); or the hosted application instance a of the same type requested by u does not have enough processing capacity p_a to meet the task's deadline; or a is overloaded and hence, the task of UE u will have to experience long waiting delay in its buffer before being processed as other tasks were scheduled on a before it (i.e, a can process the task of one UE at a time). Thus, in any of these situations, the task of UE u can be offloaded to another MEC server m' that is able to process it with respect to its QoE requirements. In this case, the serving eNB needs to transmit the task to another eNB $s \in S$ where m' is hosted. Hence, we denote by d^u_{ee}, the delay incurred for transmitting the task of a UE u from its serving eNB to the eNB connected to the MEC server where the task of u will be processed. As our SDN-based cellular core network can be used to establish a routing path between two eNBs [89], the edge-to-edge delay d^u_{ee} can be measured by the SDN controllers enabled with monitoring tools such as SLAM [97]. Thus, we define in the following a matrix $\mathcal{H}$ with elements h^m_u to represent the value of d^u_{ee} for each UE $u \in U$ to each MEC server $m \in M$. Note that $d^u_{ee} = h^m_u = 0$, if the MEC server m is attached to the serving eNB of u.

3. *Waiting delay* d^u_{wait}

   When the task of UE u reaches the MEC server m hosting the IoT application a that can process it, it may experience some waiting delays, that we denote by d^u_{wait}, in the buffer of a. Such delay depends on the scheduling order and the size of tasks assigned to a.

4. *Processing delay* d^u_{proc}

Once the task of u starts processing on its assigned application a, it will experience some processing delay, that we depict by d^u_{proc}. d^u_{proc} is the time taken by a to execute the task of UE u and is inversely proportional to the computing resources allocated to a as specified in Eq.(5.1) where μ_u and p_a are as defined above.

$$d^u_{proc} = \frac{\mu_u}{p_a} \tag{5.1}$$

5. *Download delay* d^u_{down}

   Once the execution of the task of a UE u by IoT application a is finalized, the output should be transmitted back to u. As the size of the output is usually much smaller than the initial size of the task, we assume that the download delay incurred by transferring the output to u is negligible [87]. Thus, we consider that $d^u_{down} = 0$.

In the following, we address the joint problem of task offloading, application resource allocation, and task scheduling and refer to it as the *Dynamic Task Offloading and Scheduling (DTOS)* problem.

5.4 DTOS - A Mixed Integer Linear Program (DTOS-MILP)

We define and mathematically formulate the DTOS problem as MILP.

5.4.1 Problem Definition

Let $G(N, E)$ be a physical network consisting of a set of nodes $N = R \cup M \cup S$ where R is a set of physical equipment (e.g., switches, routers, etc.) and M is a set of MEC servers attached to a set S of eNB; E is a set of links connecting them. Let A be the set of IoT applications of different types deployed on the MEC servers $m \in M$, and let U be the set of UEs requesting to offload and process their latency-sensitive tasks on these applications. The DTOS problem can be formally defined as follows:

Definition 5.1. *Given a physical network $G(N;E)$, a set U of UEs, each UE requesting to offload and process a generated task on an IoT application of the same type deployed on one of the MEC servers $m \in M$; determine the optimal assignment of the tasks generated by UEs to the set of applications $a \in A$, provision computing resources for each application a and schedule the processing of tasks assigned to each of them in order to maximize the number of admitted tasks with respect to their latency requirements.*

5.4.2 Problem Formulation

Table 5.1 and Table 5.2 respectively delineate the parameters and the decision variables used in the formulation of the DTOS-MILP problem presented below.

We define the variable $y_u^{a\delta} \in \{0,1\}$ to determine that the IoT application $a \in A$ started processing the task of UE $u \in U$ at time slot $\delta \in \Delta$.

$$y_u^{a\delta} = \begin{cases} 1 \text{ if task of UE } u \text{ started its processing on IoT application } a \text{ at time slot } \delta, \\ 0 \text{ otherwise.} \end{cases}$$

Our objective is to maximize the number of admitted tasks (Eq.(5.2)). A task of a UE $u \in U$ is admitted if it can be processed by an IoT application $a \in A$ within its specified deadline θ_u.

$$Maximize \sum_{u \in U} \sum_{a \in A} \sum_{\delta \in \Delta} y_u^{a\delta} \tag{5.2}$$

In order to meet our objective, several constraints that we elucidate in the following, have to be respected. Towards defining these constraints, we declare:

Network inputs	
$G(N, E)$	Physical network of N nodes where $N = R \cup M \cup S$ and E links connecting them.
S	Set of eNBs.
M	Set of MECs.
R	Set of physical equipment.
A	Set of IoT applications to be deployed on $m \in M$.
T	Set of IoT application types.
P	Set of processing capacities which can be assigned to an IoT application $a \in A$.
$c_m \in \mathbb{N}^+$	Processing capacity of a MEC server $m \in M$.
$x_m^a \in \{0, 1\}$	Specifies that a MEC server $m \in M$ is hosting the IoT application $a \in A$ (1), or not (0).
$t_a \in \mathbb{N}^+$	Type of IoT application $a \in A$ ($t_a \in T$).
$p_{min}^a \in \mathbb{N}^+$	Minimum processing capacity required by the IoT application $a \in A$.
UEs inputs	
U	Set of UEs.
$t_u \in \mathbb{N}^+$	Type of IoT application requested to process the task of UE $u \in U$ ($t_u \in T$).
$\theta_u \in \mathbb{N}^+$	Deadline of the task offloaded by UE $u \in U$.
$\mu_u \in \mathbb{N}^+$	Number of cycles required to process the task of UE $u \in U$.
$d_{up}^u \in \mathbb{N}^+$	Upload delay of the task of UE $u \in U$.
$h_u^m \in \mathbb{N}^+$	Edge-to-edge transmission delay of the task of a UE $u \in U$ to a MEC server $m \in M$.
Other inputs	
Δ	Set of time slots (time line).
H	Big integer number.

Table 5.1: Parameters of the DTOS-MILP.

Decision variables of DTOS-MILP	
$y_u^{a\delta} \in \{0, 1\}$	Determines that the IoT application a started processing the task of UE u at time slot δ (1) and (0) otherwise.
$p_a \in \mathbb{R}^+$	Specifies the processing capacity allocated to an IoT application a.
$n_a \in \{0, 1\}$	Depicts that an IoT application a is used (1), or not (0).
$s_{uu'}^a \in \{0, 1\}$	A of UE u started processing on IoT application a before the task of UE u' (1) and (0) otherwise.
$z_a^p \in \{0, 1\}$	IoT application a is allocated the processing capacity $p \in P$ (1) and (0) otherwise.

Table 5.2: Decision variables of the DTOS-MILP.

$p_a \in \mathbb{R}^+$ as a decision variable that determines the processing capacity allocated to an IoT application $a \in A$.

We introduce the variable $n_a \in \{0, 1\}$ to depict that an IoT application $a \in A$ is used, that is, it is processing at least one task (1), or not (0).

$$n_a = \begin{cases} 1 \text{ if IoT application } a \text{ is used,} \\ 0 \text{ otherwise.} \end{cases}$$

Further, we declare $s^a_{uu'} \in \{0, 1\}$ as a decision variable to indicate that task of UE $u \in U$ started processing on IoT application $a \in A$ before the task of UE $u' \in U$.

$$s^a_{uu'} = \begin{cases} 1 \;\text{ if task of UE } u \text{ started processing on application } a \text{ before} \\ \;\text{the task of UE } u', \\ 0 \text{ otherwise.} \end{cases}$$

In order to maximize the number of admitted tasks, we need first to decide on the computing resources to allocate to the deployed IoT applications. Hence, we define Eq.(5.3) and Eq.(5.4) to specify that an IoT application a is used if at least one task is scheduled to be processed on it, and to ensure that it is not used otherwise.

1. *Application resource allocation constraints*

$$n_a \leq \sum_{u \in U} \sum_{\delta \in \Delta} y^{a\delta}_u \quad \forall a \in A \tag{5.3}$$

$$Hn_a \geq \sum_{u \in U} \sum_{\delta \in \Delta} y^{a\delta}_u \quad \forall a \in A \tag{5.4}$$

Eq.(5.5) guarantees that a used IoT application $a \in A$ is at least allocated the minimum computing resources p^a_{min} it requires to operate.

$$p_a \geq n_a p^a_{min} \quad \forall a \in A \tag{5.5}$$

Eq.(5.6) guarantees that the maximum computing capacity that can be assigned to an IoT application $a \in A$ cannot exceed the capacity of the MEC server $m \in M$ hosting it.

$$p_a \leq n_a \sum_{m \in M} x^a_m c_m \quad \forall a \in A \tag{5.6}$$

Note that Eq.(5.5) and Eq.(5.6) ensure that an application $a \in A$ will not be allocated any computing resources if it is not used. Eq.(5.7) guarantees that the capacity of a MEC server $m \in M$ is not violated.

$$\sum_{a \in A} x^a_m p_a \leq c_m \quad \forall m \in M \tag{5.7}$$

2. *Task offloading constraints*

   A valid task offloading suggests that the task of a UE $u \in U$ cannot be scheduled on more than one application $a \in A$ (Eq.(5.8)).

$$\sum_{a \in A} \sum_{\delta \in \Delta} y^{a\delta}_u \leq 1 \quad \forall u \in U \tag{5.8}$$

   Further, the task of a UE $u \in U$, cannot be scheduled on an IoT application $a \in A$ which is of different type $t_a \in T$ than the requested one $t_u \in T$ (Eq.(5.9)).

$$\sum_{u \in U} \sum_{a \in A:(t_a != t_u)} \sum_{\delta \in \Delta} y^{a\delta}_u \leq 0 \tag{5.9}$$

3. *Task scheduling constraints*

   In addition, Eq.(5.10) guarantees that an IoT application $a \in A$ can at most process one task in a time slot.

$$\sum_{u \in U} y_u^{a\delta} \leq 1 \quad {\scriptstyle \forall \delta \in \Delta \atop \forall a \in A} \tag{5.10}$$

As we assume a non-preemptive scheduling, we present Eq.(5.11) and Eq.(5.12) to ensure that an IoT application $a \in A$ processes the task of a UE $u \in U$ completely before starting a new task. Thus, an IoT application a cannot start processing the task of UE u' before finishing the processing of the task of a UE u; that is only if u is scheduled before u' on a (Eq.(5.11)). Similarly, an IoT application a cannot start processing the task of a UE u before finishing the processing of the task of u'; that is only if u' is scheduled before u on a (Eq.(5.12)).

$$\sum_{\delta \in \Delta} y_{u'}^{a\delta} \delta \geq \sum_{\delta \in \Delta} y_u^{a\delta} \delta + d_{proc}^{u} - H(1 - s_{uu'}^{a}) \quad {\scriptstyle \forall a \in A:(t_u = t_{u'} = t_a) \atop \forall u,u' \in U:(u! = u')} \tag{5.11}$$

$$\sum_{\delta \in \Delta} y_{u}^{a\delta} \delta \geq \sum_{\delta \in \Delta} y_{u'}^{a\delta} \delta + d_{proc}^{u'} - H(1 - s_{u'u}^{a}) \quad {\scriptstyle \forall a \in A:(t_u = t_{u'} = t_a) \atop \forall u,u' \in U:(u! = u')} \tag{5.12}$$

d_{proc}^{u} in Eq.(5.11) and Eq.(5.12) determines the processing delay experienced by the task of UE u on the IoT application processing it. It is calculated as in Eq.(5.13).

$$d_{proc}^{u} = \sum_{a \in A} \sum_{\delta \in \Delta} y_u^{a\delta} \frac{\mu_u}{p_a} \quad \forall u \in U \tag{5.13}$$

Eq.(5.14) represents the precedence constraint of the schedule of the tasks of UEs u and u' on IoT application $a \in A$.

$$s_{uu'}^{a} + s_{u'u}^{a} = \sum_{\delta \in \Delta} y_u^{a\delta} \sum_{\delta \in \Delta} y_{u'}^{a\delta} \quad {\scriptstyle \forall a \in A:(t_u = t_{u'} = t_a) \atop \forall u,u' \in U:(u! = u')} \tag{5.14}$$

An IoT application a cannot start processing the task of a UE u, unless the task is uploaded and transmitted to the application (Eq.(5.15)).

$$\sum_{a\in A}\sum_{\delta\in\Delta} y_u^{a\delta} d_{up}^u + d_{ee}^u \leq \sum_{a\in A}\sum_{\delta\in\Delta} y_u^{a\delta}\delta \quad \forall u \in U \tag{5.15}$$

d_{ee}^u in Eq.(5.15) captures the edge-to-edge delay (Section 5.3) and is determined as specified in Eq.(5.16).

$$d_{ee}^u = \sum_{m\in M}\sum_{a\in A}\sum_{\delta\in\Delta} y_u^{a\delta} x_m^a h_u^m \quad \forall u \in U \tag{5.16}$$

Finally, since we are addressing tasks with stringent deadlines, we need to ensure that the total delay experienced by a task of UE $u \in U$ should not exceed its deadline as depicted in Eq.(5.17) where d_{proc}^u is as defined in Eq.(5.13).

$$\sum_{a\in A}\sum_{\delta\in\Delta} y_u^{a\delta}\delta + d_{proc}^u \leq \theta_u \quad \forall u \in U \tag{5.17}$$

Eqs.(5.11), (5.12), and (5.17) are non linear due to the non linearity of d_{proc}^u (Eq.(5.13)). Such non linearity is related to the division by the decision variable p_a which is multiplied by another decision variable $y_u^{a\delta}$. Hence, in order to simplify its linearization, we reduce the search space by allowing p_a to take at most one specific value of a predefined set P instead of all the values in $\mathbb{R}^+$. This is determined by Eq.(5.18).

$$\sum_{p\in P} z_a^p \leq 1 \quad \forall a \in A \tag{5.18}$$

where z_a^p is a new decision variable defined as follows:

$$z_a^p = \begin{cases} 1 \text{ if } a \text{ is allocated the processing capacity } p \in P, \\ 0 \text{ otherwise.} \end{cases}$$

Thus, Eq.(5.13) can then be rewritten as in Eq.(5.19).

$$d_{proc}^u = \sum_{a \in A} \sum_{\delta \in \Delta} y_u^{a\delta} \sum_{p \in P} z_a^p \frac{\mu_u}{p} \quad \forall u \in U \tag{5.19}$$

Similarly, p_a can be replaced by $\sum_{p \in P} z_a^p p$ in constraints (5.5), (5.6) and (5.7) as detailed in Appendix D.1.1. Finally, Eqs.(5.11), (5.12), (5.14) and (5.17) are non linear and can be easily linearized as explained in Appendix D.1.2.

5.4.3 DTOS Complexity

DTOS is a MILP (DTOS-MILP) which is complex to solve. It is NP-hard given that it is a combination of three NP-hard problems which are the task offloading problem, the application resource allocation problem and the non-preemptive task scheduling problem. In fact, the task offloading problem can be proven as NP-hard via a reduction from the generalized assignment problem [100] while the application resource allocation problem can be proven as NP-hard via a reduction from the two-dimensional bin-packing problem [101] where the MEC servers are the bins and the IoT applications are the objects to pack. Similarly, the task scheduling problem can be proven as NP-hard via a reduction from the job-shop scheduling problem [102].

5.5 DTOS-LBBD: A Logic-Based Benders Decomposition

Given the complexity of the DTOS problem, we devise in the following a LBBD technique to solve it (DTOS-LBBD).

5.5.1 LBBD in a Nutshell

LBBD [103], is a row generation technique which follows the "no-good learning" strategy. It consists of decomposing the problem into a MP representing a relaxation of the original model and one or more SPs. The SP, also known as inference dual, is an optimization over the secondary variables while fixing the primary variables to values computed based on the solution of the MP. By assigning the primary variables some trial values and solving the SP, the LBBD learns about the quality of other trial solutions, which are then used to reduce the number of solutions that need to be enumerated in order to find the optimal one. More precisely, the solution of the SP is used to derive Benders' cuts which are added to the MP in order to cut infeasible solutions from its solution space. An infeasible solution or a non globally feasible solution can be defined as follows:

Definition 5.2. *An infeasible solution or a non globally feasible solution is a solution provided by the MP that is not feasible to the SP (i.e., a solution that is unlikely to satisfy the SP constraints).*

Benders' cuts are derived from the inference dual which can be defined as the problem of inferring the tightest possible bound on the optimal value of the main problem [103]. LBBD consists of iteratively solving the MP and the SPs, deriving and adding Benders' cuts from the SPs to the MP until the MP and the SPs solutions converge.

5.5.2 DTOS Decomposition Strategy

The efficiency of LBBD relies on the decomposition approach and the strength of the defined Benders' cuts. Unfortunately, no standard scheme for generating Benders' cuts exists for LBBD [104].Thus, designing an efficient LBBD with strong Benders' cuts for the DTOS problem is challenging.

Hence, our DTOS-LBBD consists of dividing the DTOS problem into a MP which solves the task offloading and the application resource allocation problems, and multiple SPs to resolve the task scheduling problem. Jointly solving the task offloading and the application resource allocation problems has a positive impact on the efficiency of our DTOS-LBBD:

1. It overlooks the granularity of scheduling the tasks with respect to their latency requirements. Instead, it considers assigning the tasks of UEs to IoT applications of suitable types while allocating each of these applications the minimum processing capacity required to meet the deadline of each of the assigned tasks. This, indeed, provides an upper bound on the number of tasks that can be scheduled as the MP represents a relaxation of the original problem.

2. Since application schedules are independent from each other, our DTOS-LBBD design allows us to benefit from a distributed scheduling scheme by devising a scheduling problem for each used IoT application. Defining a SP per IoT application may not be possible if the application resource provisioning is to be solved as part of the LBBD SP due to the MEC servers capacity constraint which controls the processing capacity allocated to its hosted applications. Finally, a distributed scheduling enables the parallel execution of the SPs, and hence, reduces the overall computation time of the DTOS-LBBD.

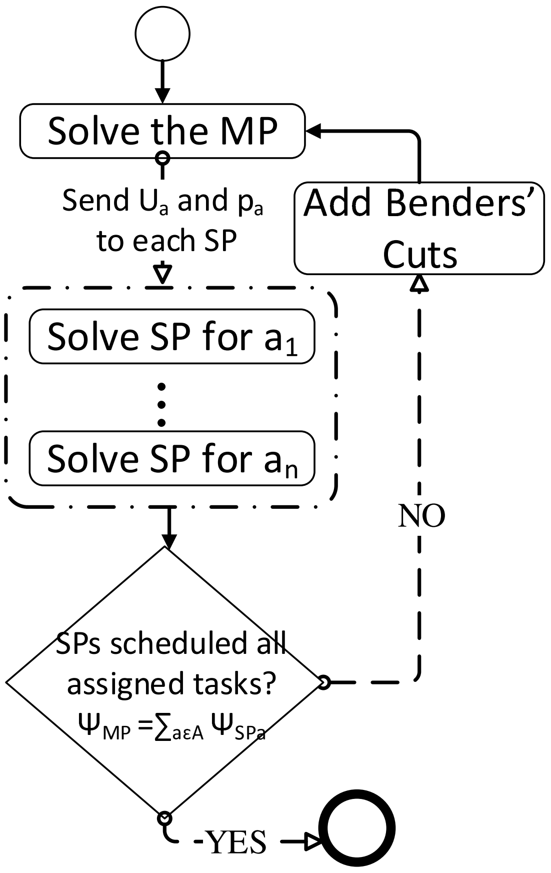

Solve the MP
Send U_a and p_a to each SP
Add Benders' Cuts
Solve SP for a_1
Solve SP for a_n
NO
SPs scheduled all assigned tasks? $\Psi_{MP} = \sum_{a \in A} \Psi_{SPa}$
-YES

Figure 5.2: DTOS-LBBD flowchart.

As depicted in Fig.5.2, the DTOS-LBBD starts by solving the MP. The MP solution yields an assignment of a subset of tasks of UEs to the hosted IoT applications and a processing capacity allocated to each of these applications. For each used application $a \in A$, a SP is defined and fed by the set of tasks of UEs assigned by the MP to a and by the processing capacity p_a allocated to a. Let ψ_{MP_a} be the total number of tasks of UEs assigned to a by the MP. Further, let ψ_{SP_a} denote the maximum number of tasks of UEs that can be scheduled within their delay requirements on a by the SP. For every used application $a \in A$, if $(\psi_{MP_a} > \psi_{SP_a})$, a Benders' cut is derived and added to the MP to guide it towards determining a better value for p_a and hence, performing a better assignment of tasks that are likely to be all scheduled by the SP. The MP problem is solved again after adding the Benders' cuts inferred from all the SPs. This process is repeated until $\sum_{a \in A} \psi_{MP_a} = \sum_{a \in A} \psi_{SP_a}$ when the optimal solution is reached.

5.5.3 Master Problem (MP)

The MP specific parameters are detailed in Table 5.3 while the remaining ones are as specified in Table 5.1.

MP inputs	
P_u^a	Set of processing capacities which each of them, if assigned to application $a \in A$, enables the task of UE u to meet its deadline ($P_u^a \subset P$).
$\sigma_u^a \in \mathbb{N}^+$	Arrival time of task of UE $u \in U$ to application $a \in A$.
$\sigma_{min}^a \in \mathbb{N}^+$	Minimum arrival time to application $a \in A$ of all tasks of UE $u \in U$ that can be processed on it.
$\theta_{max}^a \in \mathbb{N}^+$	Maximum deadline of all the tasks of UEs $u \in U$ that can be processed on $a \in A$.

Table 5.3: Parameters of the MP.

Decision variables of the MP	
$q_u^a \in \{0, 1\}$	Determines that the task of UE u is mapped to IoT application a (1) and (0) otherwise.
$n_a \in \{0, 1\}$	Specifies that an application a is used (1) and (0) otherwise.
$\beta_{ua}^j \in \{0, 1\}$	Depicts that the processing capacity $j \in P_u^a$ is selected to process the task of UE $u \in U$ on application a (1) and (0) otherwise.
$p_a \in \mathbb{R}^+$	Specifies the processing capacity allocated to an application $a \in A$.
$z_a^p \in \{0, 1\}$	Specifies that IoT application a is allocated the processing capacity $p \in P$ (1) and (0) otherwise.

Table 5.4: Decision variables of the MP.

We define the decision variable $q_u^a \in \{0, 1\}$ to determine that the task of UE $u \in U$ is mapped to IoT application $a \in A$.

$$q_u^a = \begin{cases} 1 \text{ if task of UE } u \text{ is mapped to IoT application } a, \\ 0 \text{ otherwise.} \end{cases}$$

The objective of the MP is to maximize the number of admitted tasks (Eq.(5.20)).

$$\psi_{MP} = Maximize \sum_{u \in U} \sum_{a \in A} q_u^a \tag{5.20}$$

The MP is subject to several constraints; we start by defining a decision variable $n_a \in \{0, 1\}$ to specify that an application $a \in A$ is used, that is, if a task of at least one UE is assigned to it.

$$n_a = \begin{cases} 1 \text{ if IoT application } a \text{ is used,} \\ 0 \text{ otherwise.} \end{cases}$$

We define $p_a \in \mathbb{R}^+$ as a decision variable that specifies the processing capacity allocated to an application $a \in A$.

Table 5.4 summarizes the decision variables of the MP.

1. *MP basic constraints*

   The set of constraints includes Eq.(5.21) which depicts that the task of UE $u \in U$ can be processed by at most one IoT application $a \in A$.

$$\sum_{a \in A} q_u^a \leq 1 \quad \forall u \in U \tag{5.21}$$

   Further, Eq.(5.22) is formulated to prevent the tasks of UEs to be processed on IoT applications of different type.

$$\sum_{u \in U} \sum_{a \in A : t_a != t_u} q_u^a = 0 \tag{5.22}$$

   Eq.(5.23) and Eq.(5.24) specify that an IoT application $a \in A$ is used if at least one task is offloaded to be processed on it.

$$n_a \leq \sum_{u \in U} q_u^a \quad \forall a \in A \tag{5.23}$$

$$Hn_a \geq \sum_{u \in U} q_u^a \quad \forall a \in A \tag{5.24}$$

Eq.(5.25) guarantees that an IoT application $a \in A$, if used, should at least, be allocated its minimum required processing capacity p_{min}^a.

$$p_a \geq n_a p_{min}^a \quad \forall a \in A \tag{5.25}$$

Eq.(5.26) guarantees that the maximum computing resources that can be allocated to an IoT application $a \in A$ cannot exceed those of the MEC server hosting it.

$$p_a \leq n_a \sum_{m \in M} x_m^a c_m \quad \forall a \in A \tag{5.26}$$

Eq.(5.25) and Eq.(5.26), guarantee that no computing resources can be assigned to an unused IoT application $a \in A$. We define Eq.(5.27) to ensure that the capacity of each MEC server $m \in M$ is respected.

$$\sum_{a \in A} x_m^a p_a \leq c_m \quad \forall m \in M \tag{5.27}$$

2. *Strengthening the MP formulation*

   While the above MP formulation provides an upper bound on the optimal solution (i.e., upper bound on the number of tasks that can be admitted) of the DTOS problem, our experiments have shown that it is not sufficient to efficiently solve big test instances. Therefore, we strengthen the MP formulation by adding valid inequalities to further tighten the solution space. Thus, we first add Eq.(5.28) to determine a lower bound on the processing resources to be allocated to each application $a \in A$ based on its assigned tasks. The minimum processing resources needed to process all the assigned tasks to an application a can be determined based on Eq.(5.1) by considering the maximum processing time available to them on a, and which can be calculated by deducting the earliest arrival time σ^a_{min} to a (i.e., $\sigma^a_{min} = min \quad \sigma^a_u \quad \forall u \in U : (t_u = t_a)$ where σ^a_u accounts for the upload and edge-to-edge delays of task of UE u to a) from the maximum deadline θ^a_{max} of all tasks of UEs requiring the same type of a.

$$p_a \geq \frac{\sum_{u \in U:(t_u=t_a)} \mu_u q^a_u}{\theta^a_{max} - \sigma^a_{min}} \quad \forall a \in A \tag{5.28}$$

Similarly, p_a can be lower bounded by the computing resources that, if allocated to a, allows the task of UE u to meet its deadline on a. Thus, we determine such computing resources set P^a_u by a pre-processing scheme through first applying Eq.(5.29) to calculate the minimum processing resources (p^{ua}_{min}) required on a to meet the deadline of the task of UE u. P^a_u can then be determined by adding all the processing capacities $p \in P$ that exceeds p^{ua}_{min} and do not surpass the capacity of the MEC server hosting a ($P^a_u = \{p \in P : p^{ua}_{min} \leq p \leq \sum_{m \in M} x^a_m c_m\}$).

$$p^{ua}_{min} = \frac{\mu_u}{\theta_u - \sigma^a_u} \quad \substack{\forall u \in U \\ \forall a \in A:(t_u=t_a)} \tag{5.29}$$

Hence, we define $\beta^j_{ua} \in \{0, 1\}$ as a new decision variable to determine the processing capacity $j \in P^a_u$ which is selected to process the task of UE $u \in U$ on application $a \in A$.

$$\beta^j_{ua} = \begin{cases} 1 \text{ if } j \in P^a_u \text{ is used for processing the task of UE } u \text{ on } a, \\ 0 \text{ otherwise.} \end{cases}$$

We add the inequality depicted in Eq.(5.30) as a constraint in the MP to specify that p_a should be greater than or equal to the maximum processing resources chosen to process any of the tasks of UEs $u \in U$ assigned to it.

$$p_a \geq \sum_{j \in P_u^a} j\beta_{ua}^j \ {}_{\forall a \in A}^{\forall u \in U} \tag{5.30}$$

Eq.(5.31) is added to guarantee that one processing capacity is chosen for the task of UE $u \in U$ on application $a \in A$, if and only if, it is mapped to a.

$$\sum_{j \in P_u^a} \beta_{ua}^j = q_u^a \ {}_{\forall a \in A}^{\forall u \in U} \tag{5.31}$$

As some tasks may experience high edge-to-edge delays if they were assigned to an application a of the same type hosted on a certain MEC m, the minimum processing capacity they require to meet their deadline on a may be very high and surpasses the capacity c_m of MEC m that can be provisioned to a (i.e, $P_u^a = \emptyset$). Thus, the assignment of such tasks to these applications will always lead to their rejection. Hence, we determine Eq.(5.32) to prune such assignments.

$$\sum_{u \in U:(P_u^a = \emptyset)} \sum_{a \in A: t_u = t_a} q_u^a = 0 \tag{5.32}$$

Note that as in DTOS-MILP, we replace p_a by $\sum_{p \in P} z_a^p p$ in Eq.(5.25) to Eq.(5.28) and Eq.(5.30) (as we explain in Appendix D.2.1) where $z_a^p \in \{0, 1\}$ is a decision variable (Table 5.4). We add Eq.(5.18) to guarantee that one processing capacity is allocated to an application $a \in A$.

5.5.4 The Sub-Problem (SP)

Table 5.5 depicts the parameters of the SP model. The remaining parameters are as specified in Table 5.1. The SP formulation is presented in the following.

SP inputs	
a	Used application $a \in A$ for which the SP is defined.
p_a	Processing capacity of application $a \in A$.
U_a	Subset of UEs $u \in U$; which were assigned to application a based on the solution provided by the MP.
$\sigma_u^a \in \mathbb{N}^+$	Arrival time of task of UE $u \in U$ to application $a \in A$.

Table 5.5: Parameters of the SP.

Decision variables of the SP	
$\alpha_u \in \{0, 1\}$	Determines that the task of UE $u \in U_a$ is admitted on application a (1) and (0) otherwise.
$y_u \in \mathbb{N}^+$	Determines the time slot at which the task of UE $u \in U_a$ starts processing on application a.
$s_{uu'} \in \{0, 1\}$	Indicates whether the task of UE $u \in U_a$ started processing on application a before the task of UE $u' \in U_a$ (1) and (0) otherwise.

Table 5.6: Decision variables of the SP.

We define $\alpha_u \in \{0, 1\}$ as a decision variable which determines whether the task of UE $u \in U_a$ is admitted on application a; that is, it was able to be scheduled within its latency requirements on a.

$$\alpha_u = \begin{cases} 1 \text{ task of UE } u \in U_a \text{ is admitted on } a, \\ 0 \text{ otherwise.} \end{cases}$$

We define the decision variable $y_u \in \mathbb{N}^+$ to determine the time slot at which the task of UE $u \in U_a$ starts processing on application a. Further, we declare $s_{uu'} \in \{0, 1\}$ as a decision variable to indicate whether the task of UE $u \in U_a$ started processing on application a before the task of a UE $u' \in U_a$.

$$s_{uu'} = \begin{cases} 1 \text{ task of UE } u \in U_a \text{ started processing on} \\ \text{ application } a \text{ before the task of UE } u' \in U_a, \\ 0 \text{ otherwise.} \end{cases}$$

The decision variables of the SP are highlighted in Table 5.6. The objective of the SP is to maximize the number of admitted tasks (Eq.(5.33)).

$$\psi_{SP_a} = Maximize \sum_{u \in U_a} \alpha_u \tag{5.33}$$

The SP is subject to several constraints. Eq.(5.34) is used to guarantee that the task of UE $u \in U_a$ cannot start processing on a before its arrival time, only if it is admitted.

$$y_u \geq \sigma_u^a \alpha_u \quad \forall u \in U_a \tag{5.34}$$

Further, the application a should guarantee the consecutive processing of the task of UE $u \in U_a$ during all its required processing time. Hence, Eq.(5.35) and Eq.(5.36) ensure that no two tasks can be scheduled on a at the same time.

$$y_u \geq y_{u'} + d_{proc}^{u'} \alpha_{u'} - H(1 - s_{u'u}) \quad \forall u, u' \in U_a : (u! = u') \tag{5.35}$$

$$y_{u'} \geq y_u + d_{proc}^{u} \alpha_u - H(1 - s_{uu'}) \quad \forall u, u' \in U_a : (u! = u') \tag{5.36}$$

d_{proc}^u and $d_{proc}^{u'}$ in Eq.(5.35) and Eq.(5.36) respectively determine the processing delays of u and u' calculated based on Eq.(5.1) where p_a is the processing capacity assigned by the MP to application a. Eq.(5.37) represents the precedence constraint of the schedule of the tasks of UEs u and u' on a.

$$s_{uu'} + s_{u'u} = \alpha_u \alpha_{u'} \quad \forall u, u' \in U_a : (u! = u') \tag{5.37}$$

Finally, Eq.(5.38) ensures that the total delay experienced by a task of UE $u \in U_a$ does not exceed its latency requirement.

$$y_u + d^u_{proc}\alpha_u \leq \theta_u \quad \forall u \in U_a \tag{5.38}$$

Note that, Eq.(5.37) is non linear and can be easily linearized as we explain in Appendix D.2.2.

5.5.5 Benders' Cut

When a SP fails to schedule all the tasks of UEs assigned to application a by the MP, a Benders' cut has to be generated and added to the MP to prune such solution and similar non feasible ones. In fact, the failure of the SP to schedule all of the assigned tasks is a result of an allocation of the MP of low computing resources p_a to application a. Hence, a Benders' cut can be added to guide the MP to either increase the value of p_a while keeping the same assignment of tasks, or to assign fewer tasks on the application to match those that were able to be admitted by the SP. While such cut is valid, we believe it is not strong enough as it only prunes the solutions sent by the MP without considering any other similar infeasible ones.

In order to define a stronger Benders' cut, we try to identify the solutions that are likely to be provided by the MP and will be infeasible for the SP. Thus, we graphically depict an application a as a bin of height $h = p_a$ and of width $w = \theta^a_{max} - \sigma^a_{min}$ which indicates the time horizon during which the tasks of UEs $u \in U_a$ have to be scheduled and processed on a (Fig.5.3.b). Each task $u \in U_a$ can be seen as a rectangle of height

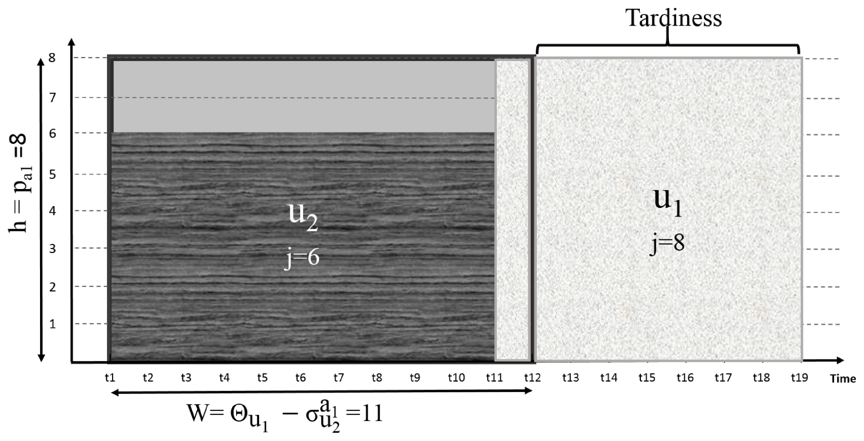
SP inputs

Application a_1

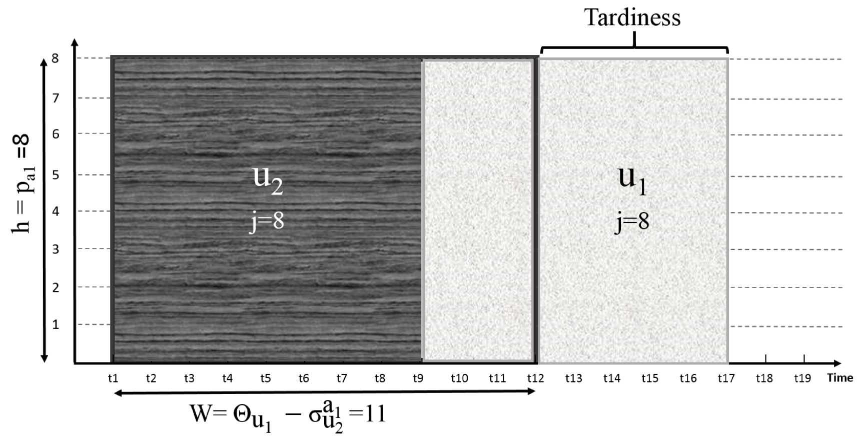
p_{a_1}= 8 cycles/time slot

UEs	Arrival time $\sigma_{\boldsymbol{u}}^{a_1}$	Cycles $\mu_{\boldsymbol{u}}$	Deadline Θ_u	$\boldsymbol{P}_{\boldsymbol{u}}^{a_1}$
u_1	t4	61	t12	{6,7,8,9,10}
u_2	t1	60	t11	{6,7,8,9,10}

5.3.a SP inputs.


Tardiness
$h = p_{a1}$ =8
8
7
6
5
4
3
2
1
u_2
j=6
u_1
j=8
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15 t16 t17 t18 t19
Time
W= $\Theta_{u_1} - \sigma_{u_2}^{a_1}$ =11

5.3.b Scheduling with $j = 6$ for u_2.


Tardiness
$h = p_{a1}$ =8
8
7
6
5
4
3
2
1
u_2
j=8
u_1
j=8
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15 t16 t17 t18 t19
Time
W= $\Theta_{u_1} - \sigma_{u_2}^{a_1}$ =11

5.3.c Scheduling with $j = 8$ for u_2.

Benders' cut

$$\beta^{6}_{u_1 a_1} + \beta^{7}_{u_1 a_1} + \beta^{8}_{u_1 a_1} + \beta^{6}_{u_2 a_1} + \beta^{7}_{u_2 a_1} + \beta^{8}_{u_2 a_1} \leq 1$$

5.3.d Benders' cut.

Figure 5.3: Benders' cut insights.

$h_u = j$ where j is the processing capacity assigned to it by the MP ($j \in P_u^a : \beta_{ua}^j = 1$) and width $w_u = d_{proc}^u$ representing the processing time of u on a when assigned the processing capacity j. The scheduling problem can then be abstracted to a bin-packing problem where a is the bin and the tasks are the objects to place in a. The geometrical size of the task $u \in U_a$ can increase by its height if we increase j (i.e., more processing capacity) or by its width if we decrease j (i.e., extend its completion time) (Eq.(5.1)).

To elaborate, we consider the example shown in Fig.5.3 where we assume two tasks u_1 and u_2, to be scheduled on application a_1 (Fig.5.3.a). a_1 can be presented as a bin, and u_1 and u_2 are the objects to be placed in it (Fig.5.3.b and Fig.5.3.c). In Fig.5.3.b, we assign the task u_2 computing resources $j = 6$ *cycles/time slot* which yields a processing delay of 10 time slots to finish at t_{11}. Increasing the computing resources to $j = 8$ *cylces/time slot* for the processing of task u_2 decreases its processing time to 8 time slots to finish at t_9 (Fig.5.3.c). However, in both cases it was not possible to admit u_1 on a_1 and satisfy its deadline.

Hence, with a processing capacity $p_{a_1} = 8$ *cylces/time slot* assigned to application a_1, at most one of both tasks u_1 or u_2 can be admitted on a_1 when assigned any computing resources $j \leq p_{a_1}$. More precisely, varying the processing capacity j assigned to u_1 or u_2 will result in the same infeasible solution. Thus, we conclude that, if the set of tasks $u \in U_a$ were not able to be scheduled by the SP with $p_a \in P$, then for sure they will not be admitted with any other value $(p'_a < p_a) \in P$. Therefore, we define a cut (Eq.(5.39)) to prune such infeasible solutions where p_a is the processing capacity allocated to application a by the MP at the previous iteration, $\tilde{U}_a$ is the set of tasks of UEs that were admitted by the SP on a and $\hat{U}_a$ is the set of UEs whose tasks were rejected by the SP. Note that Eq.(5.39) guides the MP towards assigning to application a at most the same number of tasks $|\tilde{U}_a|$ that were admitted by the

SP from the set of tasks in $(\tilde{U}_a \cup u'_{\in \hat{U}_a})$. Such guidance is only applicable in the case where the tasks in $(\tilde{U}_a \cup u'_{\in \hat{U}_a})$ were assigned computing resources $j \leq p_a$. The cut (Eq.(5.39)) is added for every task rejected by the SP. Fig.5.3.d depicts the cut that needs to be added for the example in Fig.5.3.

$$\underbrace{\sum_{u \in \tilde{U}_a} \sum_{j \in P^a_u:(j \leq p_a)} \beta^j_{ua}}_{Admitted\ tasks\ by\ SP} + \underbrace{\sum_{j \in P^a_{u'}:(j \leq p_a)} \beta^j_{u'a}}_{One\ rejected\ task\ by\ SP} \leq \underbrace{|\tilde{U}_a|}_{number\ of\ admitted\ tasks\ by\ SP} \quad \substack{\forall u' \in \hat{U}_a \\ \forall a \in A} \tag{5.39}$$

To guarantee that the DTOS-LBBD converges to an optimal solution, we need to prove that Eq.(5.39) is a valid Benders' cut [105]. A Benders' cut is valid if it satisfies the following two conditions [105]:

Condition 5.1. *The cut must exclude the current MP solution if it is not globally feasible.*

Condition 5.2. *The cut must not remove any global feasible solutions composed of any combination of tasks that were selected by the MP at a previous iteration and requiring a processing capacity $j \leq p_a$.*

Chu and Xia [105] show that Condition 5.1 guarantees finite convergence if the MP variables have finite domains, and that Condition 5.2 guarantees optimality since the cuts never cut feasible solutions.

Theorem 5.1. *Benders'cut in Eq.(5.39) is valid.*

Proof. To prove the validity of our proposed cut, we need to show that Condition 5.1 and Condition 5.2 are satisfied.

We first prove that Condition 5.1 is satisfied. To show that Eq.(5.39) cuts off infeasible solutions provided by the MP, we will show that Eq.(5.39) will not be satisfied if the same set of tasks were admitted again by the MP on application a.

Thus, we let $U_a^{(i)}$ be the set tasks which were admitted by the MP on application a at iteration i, hence resulting in a MP solution which is globally infeasible (as found by the SP). Further, let $\tilde{U}_a^{(i)}$ be the set tasks which were admitted by the SP at iteration i ($\tilde{U}_a^{(i)} \subset U_a^{(i)}$). This will result in the cut depicted in Eq.(5.40)

$$\sum_{u \in \tilde{U}_a^{(i)}} \sum_{j \in P_u^a : (j \leq p_a^{(i)})} \beta_{ua}^j + \sum_{j \in P_{u'}^a : (j \leq p_a^{(i)})} \beta_{u'a}^j \leq |\tilde{U}_a^{(i)}| \ {}_{\forall a \in A}^{\forall u' \in \hat{U}_a^{(i)}} \tag{5.40}$$

If at a subsequent iteration $k > i$, the same set of tasks U_a is admitted by the MP, the left hand side in Eq.(5.40) will be equal to $|\tilde{U}_a^{(i)}|+1$ as shown in Eq.(5.41) where $\beta_{ua}^{j(k)}$ depicts the MP solution at iteration k.

$$\underbrace{\sum_{u \in \tilde{U}_a^{(i)}} \sum_{j \in P_u^a : (j \leq p_a^{(i)})} \beta_{ua}^{j(k)}}_{|\tilde{U}_a^{(i)}|} + \underbrace{\sum_{j \in P_{u'}^a : (j \leq p_a^{(i)})} \beta_{u'a}^{j(k)}}_{1} = |\tilde{U}_a^{(i)}|+1 \ {}_{\forall a \in A}^{\forall u' \in \hat{U}_a^{(i)}} \tag{5.41}$$

The equality in Eq.(5.41) results from the following. Based on Eq.(5.31), a constraint of the MP that should be valid for its provided solution ($\beta_{ua}^{j(k)}$), we derive that $\sum_{j \in P_u^a} \beta_{ua}^{j(k)} = 1$. In addition, by accounting for the validity of Eq.(5.30), we obtain $\sum_{j \in P_u^a : (j \leq p_a^{(i)})} \beta_{ua}^{j(k)} = 1$. Hence, $\sum_{u \in \tilde{U}_a^{(i)}} \sum_{j \in P_u^a : (j \leq p_a^{(i)})} \beta_{ua}^{j(k)} = |\tilde{U}_a^{(i)}|$. Similarly, $\sum_{j \in P_{u'}^a : (j \leq p_a^{(i)})} \beta_{u'a}^{j(k)} = 1$. This explains the equality depicted in Eq.(5.41) which indeed shows that the MP solution violates the cut presented in Eq.(5.40). This proves that Condition 5.1 is satisfied.

Next, we prove that Condition 5.2 is satisfied. As we need to show that the cut (Eq.(5.39)) does not cut any feasible solution, we will provide a proof by contradiction where we consider a globally feasible solution W removed by the cut, and we show that such solution can not be feasible; where the contradiction resides. Hence, we first consider the legitimate infeasible solution I provided at iteration i and which resulted in the cut shown in Eq.(5.42), where not all the tasks assigned by the MP were admitted by the SP with a determined processing capacity $p_a^{(i)}$.

$$\sum_{u \in \tilde{U}_a^{(i)}} \sum_{j \in P_u^a : (j \leq p_a^{(i)})} \beta_{ua}^j + \sum_{j \in P_{u'}^a : (j \leq p_a^{(i)})} \beta_{u'a}^j \leq |\tilde{U}_a^{(i)}| \quad {}_{\forall a \in A}^{\forall u' \in \hat{U}_a^{(i)}} \tag{5.42}$$

The cut in (Eq.(5.42)) is designed to remove any infeasible solution composed of a subset of tasks from those in $U_a^{(i)}$ for any processing capacity less or equal than $p_a^{(i)}$ assigned to application a, and should not remove any feasible solutions. To prove this by contradiction, we consider a globally feasible solution W found at iteration $w > i$ that was removed by the cut (Eq.(5.42)). That is, in W, the tasks in $\tilde{U}_a^{(i)}$ admitted in I in addition to one or more tasks in $\hat{U}_a^{(i)}$ (that were rejected in I) are admitted in W with a processing capacity $p_a^{(w)} \leq p_a^{(i)}$. Therefore, the opposite of the cut in Eq.(5.42) which is presented by Eq.(5.43) is valid for W.

$$\sum_{u \in \tilde{U}_a^{(i)}} \sum_{j \in P_u^a : (j \leq p_a^{(i)})} \beta_{ua}^{j(w)} + \sum_{j \in P_{u'}^a : (j \leq p_a^{(i)})} \beta_{u'a}^{j(w)} > |\tilde{U}_a^{(i)}| \tag{5.43}$$

As the tasks in W are assigned a processing capacity $p_a^{(w)} \leq p_a^{(i)}$, their processing time on application a will increase in comparison to that observed with $p_a^{(i)}$, and hence the total schedule length of all tasks will be greater or equal to that obtained with $p_a^{(i)}$. As such solution W is feasible, any other solution where the tasks experience less processing delay than that observed with $p_a^{(w)}$ should also be feasible (i.e., tasks meet their deadlines). For the tasks to experience less processing delays, they need to be assigned a processing capacity higher than $p_a^{(w)}$ which is the case of solution I which is infeasible. Hence, W can not be feasible which completes the proof. ∎

5.6 Performance Evaluation

We carry out an extensive empirical study to evaluate the performance of our DTOS-LBBD approach against the DTOS-MILP. Further, we explore the engineering impact of the DTOS problem under varying system parameters and QoE requirements. We highlight the influence of the different problems solved (i.e., task offloading, application resource allocation and task scheduling) on serving multiple vertical industries while analyzing the effectiveness of our proposed DTOS-LBBD framework.

Industry Vertical	**Allowable latency (ms)**	**Applied latency (θ_u) (ms)**
Tactile Internet	1 - 10	7
Factory Automation	0.25 - 10	10
Smart Grid	3 - 20	20
Intelligent transportation Systems (ITS)	10 - 100	50
Tele Surgery	$\leq$ 250	110

Table 5.7: Latency requirements of different industry verticals [2, 3].

5.6.1 Experimental Setup

In our numerical study, we consider networks of different sizes with varying number of MEC servers, each having a capacity of $c_m = 20\ Ghz$ [87]. We account for $|T| = 5$ different types of varying number of IoT applications that belong to the same industry vertical (unless stated otherwise). Each IoT application requires minimum computing resources (p^a_{min}) randomly generated between $[2 - 5]\ Ghz$. The IoT applications are randomly placed on the MEC servers. We assume multiple UEs offloading tasks belonging to different industry verticals and hence, are of varying QoE requirements. Thus, we depict in Table 5.7 the different industry verticals accounted for in our tests, and present the range of their latency requirements in addition to the ones

	Execution Time (ms)		**Admission Rate (%)**	
Nb. of UEs ($\|U\|$)	*DTOS-MILP*	*DTOS-LBBD*	*DTOS-MILP*	*DTOS-LBBD*
5	922	56	92	92
10	2359	116.4	84	84
15	15512.6	1214	76	76
20	251218.4	10077.4	67	67
25	3014109.8	21602.6	62.4	62.4

Table 5.8: DTOS-MILP versus DTOS-LBBD.

used in our tests. We consider that the number of cycles (μ_u) demanded by UEs are randomly generated between $[20 - 100]$ *cycles*. The upload and edge-to-edge delays of the offloaded tasks are randomly generated between $[1 - 2]$ ms and $[1 - 3]$ ms [106] respectively. All our numerical evaluations are averaged over 5 sets. They are conducted using Cplex version 12.4 to solve the MIPs on an Intel core i7-4790 CPU at 3.60 GHZ with 16 GB RAM.

5.6.2 DTOS-MILP vs. DTOS-LBBD

We start by evaluating the performance of DTOS-LBBD against the DTOS-MILP in terms of execution time as we vary the number of UEs' offloaded tasks. Increasing the number of offloaded tasks makes the problem harder to solve given the limited computing resources. Hence, we also look at the the impact of such increase on the admission rate. Thus, we consider a network composed of $|M|= 3$ MEC servers and $|A|= 15$ IoT applications of $|T|= 5$ different types representing multiple industry verticals. The deadlines of the offloaded tasks are randomly generated between $[5 - 20]$ ms. Our results are presented in Table 5.8.

1. *Admission Rate*

   LBBD is an exact method which is able to provide the optimal solution as shown in Table 5.8, where the admission rates of the DTOS-MILP and DTOS-LBBD are equal. The same table depicts that as the number of UEs increases the admission rate decreases. Such decrease is expected as more tasks are contending for the same amount of computing resources, hence, some of them will be suffering from high waiting delays on some IoT applications, waiting for them to be freed. This will negatively impact their experienced latency which will lead to miss their deadlines and get rejected from the network.

2. *Execution Time*

   We evaluate the scalability of the DTOS-LBBD against the DTOS-MILP. Our results shown in Table 5.8 clearly depict that the DTOS-LBBD is much more scalable than the DTOS-MILP. In fact, it is able to provide the optimal solution on an average of 95% faster than the DTOS-MILP. This is because the LBBD learns from the quality of the solution generated at each iteration to cut off similar infeasible ones from the solution space. This will restrict the search space as the number of iterations increases and hence, will help reaching the optimal solution faster than the DTOS-MILP. In addition, the decomposition of the problem into multiple SPs helps in reducing the execution time of DTOS-LBBD especially that multiple scheduling SPs are run in parallel using threads.

5.6.3 Evaluation of DTOS-LBBD

We evaluate the performance of DTOS-LBBD under different system parameters while studying the engineering impact of the DTOS problem.

1. *DTOS-LBBD convergence*

   In order to evaluate the performance of DTOS-LBBD, we account for a single test instance and we plot in Fig.5.4, the number of admitted tasks at each iteration as determined by the MP and the SPs. We consider a network of $|M|= 10$ MEC servers hosting $|A|= 15$ IoT applications. We account for $|U|= 30$ UEs' tasks belonging to a factory automation industry vertical ($\theta_u = 10ms$).

Fig.5.4 depicts that the objective of the MP represents an upper bound on

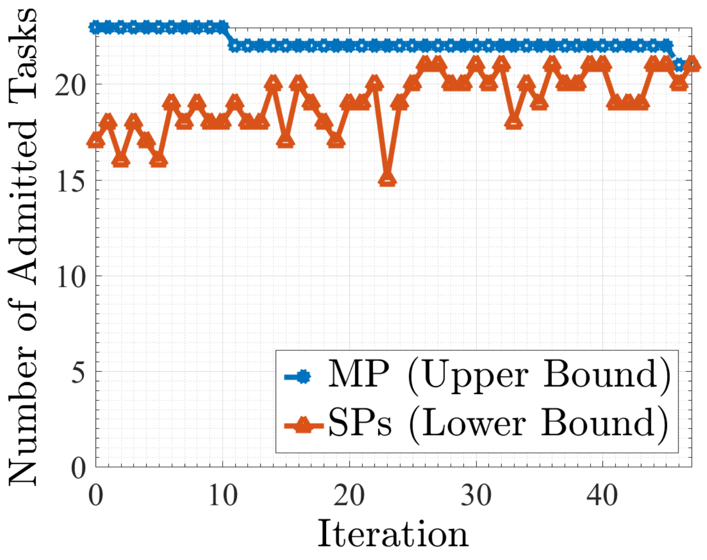


Figure 5.4: DTOS-LBBD convergence.

the optimal objective value while the number of tasks admitted by the SPs represents a lower bound. As the number of iterations increases, the objective value of the MP decreases given that more Benders' cuts are added to it. In contrast, the number of tasks admitted by the SPs varies between the iterations depending on the requirements of the tasks (i.e., number of cycles, arrival time) sent by the MP at each of them. However, it is important to note that the optimal objective value always lies between the maximum lower bound and the minimum upper bound attained so far. Further, the variance of the gap existing between the upper and lower bound provides the option to terminate the DTOS-LBBD at anytime based on the desired solution quality and runtime. For instance, one may terminate the DTOS-LBBD at iteration 14 with a gap of 9% between the upper and lower bound, scarifying little in the quality of the solution while gaining about 75.4% in terms of runtime. If a better solution quality is desired, one can stop the DTOS-LBBD at iteration 26 where the gap reaches 4.5%; however, the gain in terms of computation time is about 53%.

2. *Trade-off between optimality gap and runtime*

   To further emphasize the fact that the LBBD approach represents an anytime algorithm that can be stopped at any iteration while providing a feasible solution, we show in Table 5.9 an averaged runtime of the DTOS-LBBD using the same network settings mentioned in the previous paragraph. The results

	DTOS-LBBD Execution Time (ms)		
Nb. of UEs ($\|U\|$)	*Optimal Solution*	*Optimality Gap*$< 10\%$	$10\% <$*Optimality Gap*$< 20\%$
20	15474.8	2816.6	634.2
30	295859	30607	8085
40	1473334.6	516176.8	27890
50	1760259	419640.3333	38563.33333

Table 5.9: DTOS-LBBD execution time (ms).

   reported in Table 5.9 depict the runtime of the DTOS-LBBD at the optimal solution and at the first occurrence of an optimality gap which is either less than 10% or between 10% and 20%. It is clear that for a determined number of UEs, the runtime increases with the decrease of the optimality gap. In fact, the runtime of DTOS-LBBD increases with the increase of the number of iterations. In this case, more Benders' cuts are added to the MP tightening its solution space, hence, better locating the optimal solution which is likely to decrease the gap between the upper bound provided by the MP and the lower bound given by the SPs. Thus, stopping the DTOS-LBBD at a certain tolerable gap can lead to high gains in terms of computation time. For instance, when $|U|= 30$, 97.26% of gain in runtime is depicted when the gap is between 10% and 20%, while 89.65% is obtained with a gap less than 10%. Finally, it is worth noting that as the number of UEs increases, the runtime of the DTOS-LBBD increases as the size of the problem grows. Hence, the problem becomes harder to solve.

3. *Impact of varying the number of UEs*

We vary the number of UEs and evaluate its impact for different industry verticals. Thus, we consider a network of $|U|= 10$ MEC servers hosting $|A|= 15$ IoT applications.

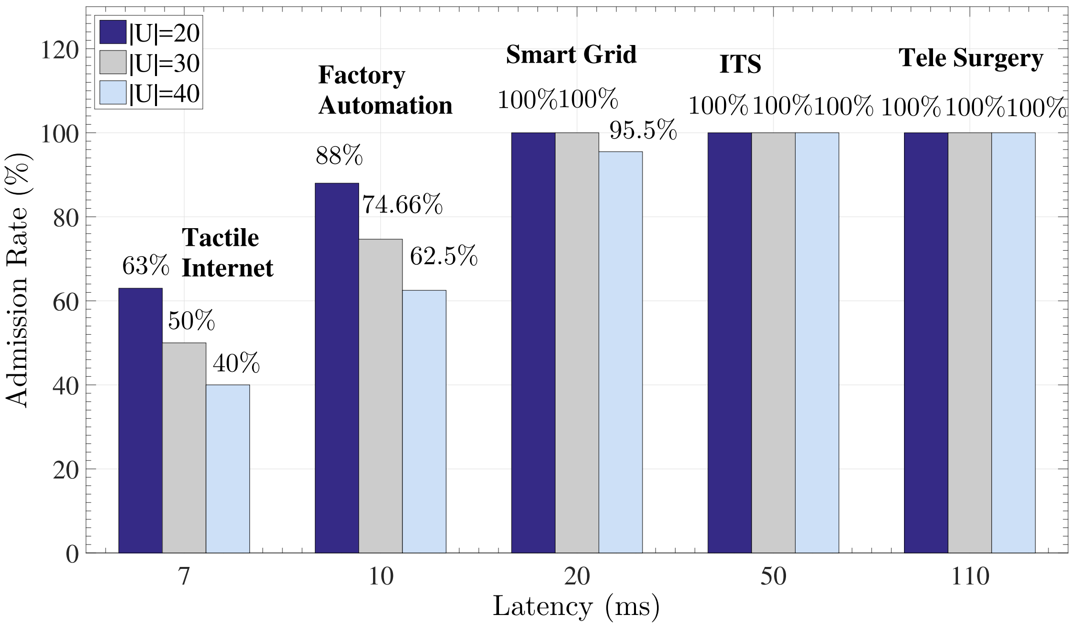


Figure 5.5: Admission rate per varying number of UEs.

Our results presented in Fig.5.5 show that as the number of UEs increases the admission rate decreases for each vertical industry as more tasks are contending the same computing resources (IoT applications) which become overloaded and hence, fail to meet the delay requirements of all the UEs requesting their service. In fact, some tasks will suffer from extra waiting delays which will lead them to miss their deadlines and thus, get rejected from the network. However, such waiting delays can be tolerated if the latency requirements increased. For instance, one can note the tactile Internet vertical where the number of UEs increased from $|U|= 20$ to $|U|= 40$ while the admission rate decreased by 23% as the limited computing resources failed to cope with such increase. In contrast, for $|U|= 20$, the admission rate increased to 100% for less latency sensitive tasks such as those belonging to Tele surgery industry. Further, as Intelligent Transportation Systems (ITS) and Tele surgery vertical industries possess relatively high delay requirements, the admission rate of UEs requiring such types of services was not affected by the increase of the number of tasks and was kept constant to 100%.

4. *Impact of varying the number of MEC servers*

   We study in Fig.5.6 the impact of the increase of the computing resources for different vertical industries on the admission rate. Hence, we consider a network of varying number of MEC servers hosting $|A|= 15$ IoT applications. We account for $|U|= 30$ UEs offloading tasks of varying latency requirements. Fig.5.6

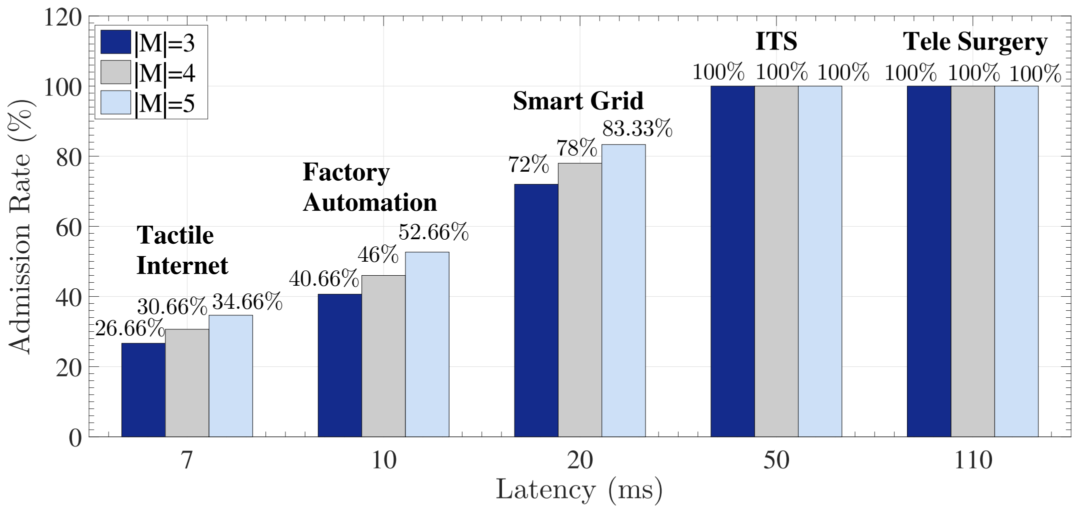


Figure 5.6: Admission rate per varying number of MEC servers.

shows that adding more MEC servers in the network increases the amount of computing resources available. This allows the hosted IoT applications to be provisioned more processing capacity, which will reduce the processing time of the assigned tasks. Thus, as tasks will be processed faster by the applications, others, waiting for the same resource to be freed will experience less waiting delays, and hence, their chances in meeting their deadlines and be admitted to the network will increase. In addition, one can note that for a fixed number of MEC servers, the admission rate increases with the increase of the latency requirements; as less-sensitive tasks can tolerate more waiting delay on the shared IoT applications. For instance, with $|M|= 3$ MEC servers, the admission rate increased by 73.34% as the deadline of the tasks increased from 7 ms for tactile Internet to 110 ms for Tele surgery. Further, the 100% admission rate depicted for ITS and Tele surgery vertical industries for the varying number of MEC servers depicts that $|M|= 3$ MEC servers were enough to admit all the assigned tasks given their relaxed latency demands.

5. *Impact of varying the edge-to-edge delay*

   To explore the impact of the edge-to-edge delay on the admission rate, we consider a network of $|M|= 5$ MEC servers hosting $|A|= 15$ IoT applications. We account for $|U|= 25$ UEs belonging to a smart grid industry vertical ($\theta_u = 20$ ms) and we fix the edge-to-edge delays for all the tasks to a defined value. Our results presented in Fig.5.7 show that the edge-to-edge delay in-

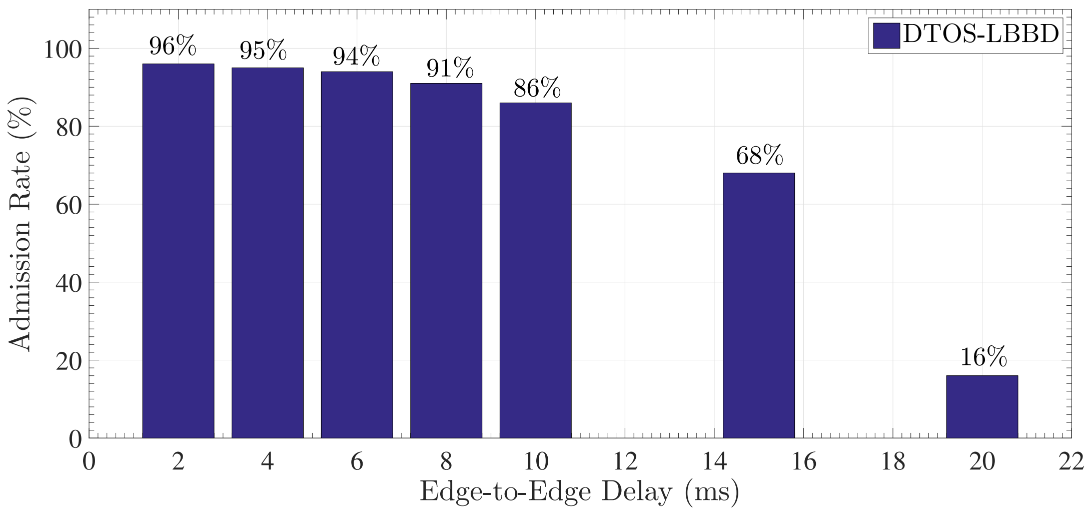


Figure 5.7: Admission rate per varying edge-to-edge delay.

crease becomes prohibitive in allowing the admission of the tasks. In fact, it is of the best interest of each task to be processed on a MEC server attached to its serving eNB in order to overcome the edge-to-edge delay. However, as the number of IoT applications is fixed in the network, some MEC servers may not be hosting certain types, further, some of their deployed applications may be overloaded. This will force the tasks served by eNBs attached to those MEC servers to travel through the network to be processed on an IoT application hosted on another MEC server. This will make these tasks suffer from high edge-to-edge delay. With their latency-sensitive requirements, the mentioned tasks will be left with very little processing time which the IoT application on which they are assigned might fail to meet, hence, leading to their rejection from the network.

5.7 Conclusion

In this chapter, we motivated and studied the DTOS problem which jointly addresses the task offloading, application resource allocation in addition to the task scheduling problems in a MEC network. We alleviate virtualization technologies capabilities through determining the computing resources to be allocated to the IoT applications based on the requirements of their scheduled tasks. To the best of our knowledge, we are the first to study the task scheduling problem under undetermined IoT applications' computing resources. Given the complexity of DTOS, we presented a novel decomposition strategy implementing the LBBD technique. Our novel DTOS-LBBD method decomposes the problem into a MP which solves the task offloading and application resource allocation problems; and multiple SPs, each addressing the scheduling of tasks on a single used IoT application. DTOS-LBBD is an exact method characterized by an anytime algorithm providing the opportunity to be terminated at any iteration, hence, realizing the trade-off between the solution quality and the computation time. Through extensive simulations, we show that the DTOS-LBBD can achieve more than 140 order of magnitude improvement in terms of runtime compared to the DTOS-MILP and can serve as a benchmark algorithm to compare against other methods. Further, we explored the interleaving dependence and implications of the aforementioned DTOS problems on different vertical industries with variable latency requirements.

Chapter 6

Conclusion and Future Research Directions

This chapter concludes the presented thesis and highlights future research directions.

6.1 Conclusion

The concept of smart living has emerged in recent years and continue to gain significant interest towards improving our quality of life through enabling innovative services leveraged by advanced information and communication technologies. Services ranging from smart health care, smart traffic management, self-driving cars, smart city infrastructure management (i.e., electricity, water supply, etc.) continue to evolve with the advancements of many IoT devices that are not limited to smart phones and tablets but also include sensors, smart meters among others. Towards supporting the heterogeneous QoS requirements of these emerging services enabled by the increasing number of IoT devices, network operators and research communities continue their investigations and investments in optimizing their technological infrastructure.

Throughout this thesis, we addressed several challenges related to providing efficient network and service management empowered by the emerging trend of network softwarization. Thus, we first highlighted the shortcomings of existing networks in coping with the heterogeneous requirements of the new innovative services and presented the different promising technologies, mainly, 5G, NFV, SDN and MEC, that will assist network operators in unleashing the power of their networks to support these services.

We presented in chapter 2 the various challenges in NFV, mainly the NF mapping, the traffic routing and the NS scheduling problems, that were the main focus of this thesis. We reviewed the work in the literature addressing these problems and highlighted the relationship existing between them. Motivated by the lessons learned from the literature, we studied in chapter 3 the interleaving relationship between these problems and we observed that the NF mapping and the traffic routing impact the schedule of NSs. Indeed, we noticed that there exists a trade-off between the resource utilization and the schedule length of the different NSs. Given this trade-off, we jointly addressed the aforementioned problems with the objective of minimizing the total schedule length of the different NSs. We mathematically formulated this problem and presented a novel CG decomposition approach to efficiently solve it. We concluded the chapter with different gap and performance analysis showing that the proposed CG approach is much more scalable than the MILP formulation of the problem and can serve as a benchmark algorithm for evaluating the performance of any low complexity method addressing the same problem.

Driven by the ultra-low latency requirements of 5G services and the different observations and results obtained in Chapter 3, we revisited in Chapter 4 the same joint problem of NF mapping, traffic routing and NS scheduling, however, we considered the fixed deadlines of the different NSs. We evaluated the impact of such fixed latency requirements on the admission of the NSs to the network by studying the mentioned joint problem under different objectives; mainly, maximizing the number of admitted NSs, minimizing the total schedule length and minimizing the sum of completion times of the different scheduled NSs. While we presented different MILPs with these objectives, we compared them against a novel game theoretic technique that we developed. Our game theoretic approach provides NSs the freedom to decide on their own mapping, routing and scheduling solution while orchestrating their schedule decisions through a centralized controller. We leveraged the relationship between the three addressed problems by evaluating the proposed game theoretic approach under two different traffic routing strategies (i.e., shortest path route, random route) and show their direct impact on the network utilization.

With the emergence of the MEC paradigm enabled by network softwarization towards fulfilling the special low-latency and high computational requirements of the IoT applications, and given the high impact of NS scheduling on meeting the requested response times, we apply our previous studies to a MEC enabled environment. Thus, we tackle in Chapter 5 the joint task offloading and scheduling problem jointly with the IoT application resource allocation problem. As the processing time of a task is highly dependent on the computing resources allocated to the application executing it, and given that the processing time is an integral part of the task scheduling problem and affects its admission to the network, we studied and evaluated the influence of the computing resources allocated to the IoT application on the scheduling and offloading decisions. This study comes in line with the objective of network softwarization in providing efficient network resource management by leveraging virtualization technologies such as VMs and containers which allocated resources can be automatically adjusted and scaled based on changing network demands. Hence, we mathematically formulated the mentioned joint problem as a MILP. Given its complexity, we presented a novel LBBD approach that solves it to optimality and yields much more scalable than the presented MILP. The chapter is concluded with valuable performance trends highlighting the impact of task offloading and scheduling in meeting the diverse QoE requirements of different 5G vertical industries.

Finally, this thesis presented significant contributions in the area of network and service management by tackling different challenges in NFV and MEC that play an intrinsic role in enabling elastic, scalable and cost-efficient networks.

6.2 Future Research Directions

While we addressed several research challenges in NFV and MEC, there still exist many future research directions that need to be tackled.

6.2.1 Online Resource Allocation and Scheduling in MEC

As the MEC computing resources are limited in comparison to those existing in a centralized cloud, and given the main purpose of MEC in enabling ultra-low latency services, efficient online resource allocation and scheduling play an important role in meeting the promised advantages of MEC. While in Chapter 5 we tackled the joint problem of task offloading and scheduling along with IoT resource allocation in an offline scenario where tasks are considered to be known a priori and the computing resources to allocate to each IoT application, once decided remains unchanged; we believe that it would be interesting to study the online aspect of the problem. In an online setting where tasks arrive and depart from the MEC dynamically, the load on each of the deployed IoT application changes over time, and hence, an efficient resource utilization of MEC servers entails dynamically adapting the amount of computing resources allocated to each IoT application in compliance to the latency requirements of the already scheduled tasks on this application that are currently being processed or waiting to be executed; and the newly offloaded tasks at a subsequent time.

6.2.2 Scalable Resource Allocation in MEC

One of the shortcomings of LBBD presented in Chapter 5 is its scalability. Thus, carrying out the resource allocation and scheduling problem in MEC-enabled networks through a scalable distributed approach is an interesting research direction that can be pursued.

6.2.3 Reliability Guarantees for Ultra-low Latency Services in MEC

While ultra-low latency is one of the main requirements of 5G services, high reliability yields another important metric to be guaranteed. Thus, an interesting research direction in the area of 5G and MEC, is to study the problem of reliability guarantee of the offloaded tasks while respecting their latency requirements. Here, each MEC server has a certain reliability that should be accounted for when selecting it to process the offloaded task. Given that one MEC server may not be able to guarantee the requested reliability, the task will need to be replicated [14] and offloaded to multiple MEC servers which jointly can provide it with its demanded reliability.

Bibliography

[1] Rajendra Chayapathi, Syed F Hassan, and Paresh Shah. *Network Functions Virtualization (NFV) with a Touch of SDN*. Addison-Wesley Professional, 2016.

[2] Maria A Lema, Andres Laya, Toktam Mahmoodi, Maria Cuevas, Joachim Sachs, Jan Markendahl, and Mischa Dohler. Business case and technology analysis for 5g low latency applications. *IEEE Access*, 5:5917–5935, 2017.

[3] Philipp Schulz, Maximilian Matthe, Henrik Klessig, Meryem Simsek, Gerhard Fettweis, Junaid Ansari, Shehzad Ali Ashraf, Bjoern Almeroth, Jens Voigt, Ines Riedel, et al. Latency critical iot applications in 5g: Perspective on the design of radio interface and network architecture. *IEEE Communications Magazine*, 55(2):70–78, 2017.

[4] Lopa J Vora. Evolution of mobile generation technology: 1g to 5g and review of upcoming wireless technology 5g. *International Journal of Modern Trends in Engineering and Research*, 2(10):281–290, 2015.

[5] Nidhi Shah. The evolution of mobile apps – 1994 through 2016. `https://arkenea.com/blog/evolution-of-mobile-apps/`, 2018.

[6] Faqir Zarrar Yousaf, Michael Bredel, Sibylle Schaller, and Fabian Schneider. Nfv and sdn-key technology enablers for 5g networks. *IEEE Journal on Selected Areas in Communications*, 35(11):2468–2478, 2017.

[7] Manuel Silverio-Fernández, Suresh Renukappa, and Subashini Suresh. What is a smart device?-a conceptualisation within the paradigm of the internet of things. *Visualization in Engineering*, 6(1):3, 2018.

[8] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1(2011):1–11, 2011.

[9] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262, 2016.

[10] Tarik Taleb, Adlen Ksentini, and Riku Jantti. " anything as a service" for 5g mobile systems. *IEEE Network*, 30(6):84–91, 2016.

[11] Ian F Akyildiz, Shuai Nie, Shih-Chun Lin, and Manoj Chandrasekaran. 5g roadmap: 10 key enabling technologies. *Computer Networks*, 106:17–48, 2016.

[12] Imtiaz Parvez, Ali Rahmati, Ismail Guvenc, Arif I Sarwat, and Huaiyu Dai. A survey on low latency towards 5g: Ran, core network and caching solutions. *IEEE Communications Surveys & Tutorials*, 2018.

[13] M Series. Imt vision–framework and overall objectives of the future development of imt for 2020 and beyond. *Recommendation ITU*, pages 2083–0, 2015.

[14] Tarik Taleb, Konstantinos Samdanis, Badr Mada, Hannu Flinck, Sunny Dutta, and Dario Sabella. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681, 2017.

[15] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H Glitho, Monique J

Morrow, and Paul A Polakos. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 20(1):416–464, 2017.

[16] Pawani Porambage, Jude Okwuibe, Madhusanka Liyanage, Mika Ylianttila, and Tarik Taleb. Survey on multi-access edge computing for internet of things realization. *arXiv preprint arXiv:1805.06695*, 2018.

[17] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *arXiv preprint arXiv:1702.05309*, 2017.

[18] Xin Li and Chen Qian. A survey of network function placement. In *Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual*, pages 948–953. IEEE, 2016.

[19] Justine Sherry, Sylvia Ratnasamy, and Justine Sherry At. A survey of enterprise middlebox deployments. 2012.

[20] Ruozhou Yu, Guoliang Xue, Vishnu Teja Kilari, and Xiang Zhang. Network function virtualization in the multi-tenant cloud. *IEEE Network*, 29(3):42–47, 2015.

[21] NFVISG ETSI. Network functions virtualisation (nfv); terminology for main concepts in nfv. *Group Specification*, 2014.

[22] Juliver Gil Herrera and Juan Felipe Botero. Resource allocation in nfv: A comprehensive survey. *IEEE Transactions on Network and Service Management*, 13(3):518–532, 2016.

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

[24] Attila Hegyi, Hannu Flinck, Istvan Ketyko, Pekka Kuure, Csaba Nemes, and Lajos Pinter. Application orchestration in mobile edge cloud: placing of iot applications to the edge. In *Foundations and Applications of Self* Systems, IEEE International Workshops on*, pages 230–235. IEEE, 2016.

[25] Md Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, and Raouf Boutaba. On orchestrating virtual network functions. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 50–56. IEEE, 2015.

[26] GSNFV ETSI. Network functions virtualisation (nfv): Architectural framework. *ETsI Gs NFV*, 2(2):V1, 2013.

[27] Jrgen Quittek, P Bauskar, T BenMeriem, A Bennett, M Besson, and A Et. Network functions virtualisation (nfv)-management and orchestration. *ETSI NFV ISG, White Paper*, 2014.

[28] Rashid Mijumbi, Joan Serrat, Juan-luis Gorricho, Steven Latre, Marinos Charalambides, and Diego Lopez. Management and orchestration challenges in network functions virtualization. *IEEE Communications Magazine*, 54(1):98–105, 2016.

[29] NM Mosharaf Kabir Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Virtual network embedding with coordinated node and link mapping. In *INFOCOM 2009, IEEE*, pages 783–791. IEEE, 2009.

[30] Maryam Jalalitabar, Guangchun Luo, Chenguang Kong, and Xiaojun Cao. Service function graph design and mapping for nfv with priority dependence. In *Global Communications Conference (GLOBECOM), 2016 IEEE*, pages 1–5. IEEE, 2016.

[31] Long Qu, Chadi Assi, and Khaled Shaban. Delay-aware scheduling and resource optimization with network function virtualization. *IEEE Transactions on Communications*, 64(9):3746–3758, 2016.

[32] Xin Li and Chen Qian. The virtual network function placement problem. In *Computer Communications Workshops (INFOCOM WKSHPS), 2015 IEEE Conference on*, pages 69–70. IEEE, 2015.

[33] Sara Ayoubi, Samir Sebbah, and Chadi Assi. A cut-and-solve based approach for the vnf assignment problem. *IEEE Transactions on Cloud Computing*, 2017.

[34] Tachun Lin, Zhili Zhou, Massimo Tornatore, and Biswanath Mukherjee. Demand-aware network function placement. *Journal of Lightwave Technology*, 34(11):2590–2600, 2016.

[35] Bernardetta Addis, Dallal Belabed, Mathieu Bouet, and Stefano Secci. Virtual network functions placement and routing optimization. In *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, pages 171–177. IEEE, 2015.

[36] Li Erran Li, Vahid Liaghat, Hongze Zhao, MohammadTaghi Hajiaghayi, Dan Li, Gordon Wilfong, Y Richard Yang, and Chuanxiong Guo. Pace: Policy-aware application cloud embedding. In *INFOCOM, 2013 Proceedings IEEE*, pages 638–646. IEEE, 2013.

[37] Marcelo Caggiani Luizelli, Leonardo Richter Bays, Luciana Salete Buriol, Marinho Pilla Barcellos, and Luciano Paschoal Gaspary. Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 98–106. IEEE, 2015.

[38] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Steven Davy. Design and evaluation of algorithms for mapping and scheduling of virtual network functions. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–9. IEEE, 2015.

[39] Tung-Wei Kuo, Bang-Heng Liou, Kate Ching-Ju Lin, and Ming-Jer Tsai. Deploying chains of virtual network functions: On the relation between link and server usage. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE, 2016.

[40] Xin Li, Haotian Wu, Don Gruenbacher, Caterina Scoglio, and Tricha Anjali. Efficient routing for middlebox policy enforcement in software-defined networking. *Computer Networks*, 110:243–252, 2016.

[41] Aaron Gember, Anand Krishnamurthy, Saul St John, Robert Grandl, Xiaoyang Gao, Ashok Anand, Theophilus Benson, Aditya Akella, and Vyas Sekar. Stratos: A network-aware orchestration layer for middleboxes in the cloud. Technical report, Technical Report, 2013.

[42] Sameer G Kulkarni, Wei Zhang, Jinho Hwang, Shriram Rajagopalan, KK Ramakrishnan, Timothy Wood, Mayutan Arumaithurai, and Xiaoming Fu. Nfvnice: Dynamic backpressure and scheduling for nfv service chains. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 71–84. ACM, 2017.

[43] Jordi Ferrer Riera, Xavier Hesselbach, Eduard Escalona, Joan A Garcia-Espin, and Eduard Grasa. On the complex scheduling formulation of virtual network functions over optical networks. In *Transparent Optical Networks (ICTON), 2014 16th International Conference on*, pages 1–5. IEEE, 2014.

[44] Christian Artigues, Sophie Demassey, and Emmanuel Neron. *Resource-constrained project scheduling: models, algorithms, extensions and applications.* John Wiley & Sons, 2013.

[45] Shohreh Ahvar, Hnin Pann Phyu, Sachham Man Buddhacharya, Ehsan Ahvar, Noel Crespi, and Roch Glitho. Ccvp: Cost-efficient centrality-based vnf placement and chaining algorithm for network service provisioning. In *Network Softwarization (NetSoft), 2017 IEEE Conference on*, pages 1–9. IEEE, 2017.

[46] Gupta Abhishek, Jaumard Brigitte, Massimo Tornatore, Mukherjee Biswanath, et al. Multiple service chain placement and routing in a network-enabled cloud. In *IEEE International Conference on Advanced Networks and Telecommunications Systems*, pages 1–3, 2016.

[47] Abhishek Gupta, Brigittte Jaumard, Massimo Tornatore, and Biswanath Mukherjee. Service chain (sc) mapping with multiple sc instances in a wide area network. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–6. IEEE, 2017.

[48] Mathis Obadia, Jean-Louis Rougier, Luigi Iannone, Vania Conan, and Mathieu Brouet. Revisiting nfv orchestration with routing games. In *Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on*, pages 107–113. IEEE, 2016.

[49] Salvatore D'Oro, Laura Galluccio, Sergio Palazzo, and Giovanni Schembra. Exploiting congestion games to achieve distributed service chaining in nfv networks. *IEEE Journal on Selected Areas in Communications*, 35(2):407–420, 2017.

[50] Aris Leivadeas, George Kesidis, Matthias Falkner, and Ioannis Lambadaris.

A graph partitioning game theoretical approach for the vnf service chaining problem. *IEEE Transactions on Network and Service Management*, 14(4):890–903, 2017.

[51] Xiaoliang Chen, Zuqing Zhu, Jiannan Guo, Sheng Kang, Roberto Proietti, Alberto Castro, and SJB Yoo. Leveraging mixed-strategy gaming to realize incentive-driven vnf service chain provisioning in broker-based elastic optical inter-datacenter networks. *IEEE/OSA Journal of Optical Communications and Networking*, 10(2):A232–A240, 2018.

[52] Jordi Ferrer Riera, Eduard Escalona, Josep Batalle, Eduard Grasa, and Joan A Garcia-Espin. Virtual network function scheduling: Concept and challenges. In *Smart Communications in Network Technologies (SaCoNeT), 2014 International Conference on*, pages 1–5. IEEE, 2014.

[53] Mark Shifrin, Erez Biton, and Omer Gurewitz. Optimal control of vnf deployment and scheduling. In *Science of Electrical Engineering (ICSEE), IEEE International Conference on the*, pages 1–5. IEEE, 2016.

[54] Yujie Liu, Yong Li, Marco Canini, Yue Wang, and Jian Yuan. Scheduling multi-flow network updates in software-defined nfv systems. In *Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on*, pages 548–553. IEEE, 2016.

[55] Hyame Assem Alameddine, Samir Sebbah, and Chadi Assi. On the interplay between network function mapping and scheduling in vnf-based networks: A column generation approach. *IEEE Transactions on Network and Service Management*, 14(4):860–874, 2017.

[56] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, 2015.

[57] Marco E Lübbecke and Jacques Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

[58] Hatem Ben Amor, Jacques Desrosiers, and José Manuel Valério de Carvalho. Dual-optimal inequalities for stabilized column generation. *Operations Research*, 54(3):454–463, 2006.

[59] Mokhtar S Bazaraa, John J Jarvis, and Hanif D Sherali. *Linear programming and network flows*. John Wiley & Sons, 2011.

[60] Leon S Lasdon. *Optimization theory for large systems*. Courier Corporation, 2002.

[61] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Towards predictable datacenter networks. In *ACM SIGCOMM computer communication review*, volume 41, pages 242–253. ACM, 2011.

[62] Jeffrey C Mogul and Lucian Popa. What we talk about when we talk about cloud network performance. *ACM SIGCOMM Computer Communication Review*, 42(5):44–48, 2012.

[63] Jeongkeun Lee, Yoshio Turner, Myungjin Lee, Lucian Popa, Sujata Banerjee, Joon-Myung Kang, and Puneet Sharma. Application-driven bandwidth guarantees in datacenters. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 467–478. ACM, 2014.

[64] Hyame Assem Alameddine, Sara Ayoubi, and Chadi Assi. Offering resilient and bandwidth guaranteed services in multi-tenant cloud networks: Harnessing

the sharing opportunities. In *Teletraffic Congress (ITC 28), 2016 28th International*, volume 1, pages 1–9. IEEE, 2016.

[65] Hyame Assem Alameddine, Sara Ayoubi, and Chadi Assi. Protection plan design for cloud tenants with bandwidth guarantees. In *Design of Reliable Communication Networks (DRCN), 2016 12th International Conference on the*, pages 115–122. IEEE, 2016.

[66] Dariush Ebrahimi, Samir Sebbah, and Chadi Assi. A column generation method for constructing and scheduling multiple forwarding trees in wireless sensor networks. *IEEE Transactions on Wireless Communications*, 15(9):6513–6523, 2016.

[67] Hyame Assem Alameddine, Mosaddek Hossain Kamal Tushar, and Chadi Assi. Scheduling of low latency services in softwarized networks (under review). *IEEE Transactions on Cloud Computing*, 2019.

[68] Dario Bega, Marco Gramaglia, Albert Banchs, Vincenzo Sciancalepore, Konstantinos Samdanis, and Xavier Costa-Perez. Optimising 5g infrastructure markets: The business of network slicing. In *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*, pages 1–9. IEEE, 2017.

[69] Peter Rost, Christian Mannweiler, Diomidis S Michalopoulos, Cinzia Sartori, Vincenzo Sciancalepore, Nishanth Sastry, Oliver Holland, Shreya Tayade, Bin Han, Dario Bega, et al. Network slicing to enable scalability and flexibility in 5g mobile networks. *IEEE Communications Magazine*, 55(5):72–79, 2017.

[70] Spyridon Vassilaras, Lazaros Gkatzikis, Nikolaos Liakopoulos, Ioannis N Stiakogiannakis, Meiyu Qi, Lei Shi, Liu Liu, Merouane Debbah, and Georgios S

Paschos. The algorithmic aspects of network slicing. *IEEE Communications Magazine*, 55(8):112–119, 2017.

[71] Sönke Hartmann and Dirk Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of operational research*, 207(1):1–14, 2010.

[72] Giacomo Bonanno. *Game Theory*. CreateSpace Independent Publishing Platform, 2nd edition, 2015.

[73] Jim Ratliff. Strategies in extensive-form games. `http://www.virtualperfection.com/gametheory/4.2.StrategiesInExtensiveFormGames.1.0.pdf`, 1997.

[74] Omer Tamus. Lecture notes on game theory". `http://tamuz.caltech.edu/teaching/ss201b/lectures.pdf`, 2018.

[75] Jonathan Levin. Extensive form games. `https://web.stanford.edu/~jdlevin/Econ%20203/ExtensiveForm.pdf`, 2002.

[76] Martin Gairing, Burkhard Monien, and Karsten Tiemann. Routing (un-) splittable flow in games with player-specific linear latency functions. In *International Colloquium on Automata, Languages, and Programming*, pages 501–512. Springer, 2006.

[77] Martin Gairing, Burkhard Monien, and Karsten Tiemann. Routing (un-) splittable flow in games with player-specific affine latency functions. *ACM Transactions on Algorithms (TALG)*, 7(3):31, 2011.

[78] Hyame Assem Alameddine, Sannaa Sharafeddine, Samir Sebbah, Sara Ayoubi, and Chadi Assi. Dynamic task offloading and scheduling for low-latency iot

services in multi-access edge computing. *IEEE Journal on Selected Areas in Communications - Special Issue on Network Softwarization & Enablers*, 2019.

[79] 5G Infrastructure PPP Association et al. 5g vision-the 5g infrastructure public private partnership: the next generation of communication networks and services. *White Paper, February*, 2015.

[80] Mahadev Satyanarayanan, Victor Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 2009.

[81] Shuo Wang, Xing Zhang, Yan Zhang, Lin Wang, Juwo Yang, and Wenbo Wang. A survey on mobile edge networks: Convergence of computing, caching and communications. *IEEE Access*, 5:6757–6779, 2017.

[82] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[83] Milan Patel, B Naughton, C Chan, N Sprecher, S Abeta, A Neal, et al. Mobile-edge computing introductory technical white paper. *White Paper, Mobile-edge Computing (MEC) industry initiative*, 2014.

[84] Alex Reznik, Rohit Arora, Mark Cannon, Luca Cominardi, Walter Featherstone, Rui Frazao, Fabio Giust, Sami Kekki, Alice Li, Dario Sabella, et al. Developing software for multi-access edge computing. *ETSI, White Paper*, (20), 2017.

[85] Hiroyuki Tanaka, Masahiro Yoshida, Koya Mori, and Noriyuki Takahashi. Multi-access edge computing: A survey. *Journal of Information Processing*, 26:87–97, 2018.

[86] Haisheng Tan, Zhenhua Han, Xiang-Yang Li, and Francis CM Lau. Online job dispatching and scheduling in edge-clouds. In *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*, pages 1–9. IEEE, 2017.

[87] Tuyen X Tran and Dario Pompili. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *arXiv preprint arXiv:1705.00704*, 2017.

[88] Lin Wang, Lei Jiao, Jun Li, and Max Mühlhäuser. Online resource allocation for arbitrary user mobility in distributed edge clouds. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 1281–1290. IEEE, 2017.

[89] Xiang Sun and Nirwan Ansari. Latency aware workload offloading in the cloudlet network. *IEEE Communications Letters*, 21(7):1481–1484, 2017.

[90] Mike Jia, Weifa Liang, and Zichuan Xu. Qos-aware task offloading in distributed cloudlets with virtual network function services. In *Proceedings of the 20th ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems*, pages 109–116. ACM, 2017.

[91] Qiang Fan and Nirwan Ansari. Application aware workload allocation for edge computing-based iot. *IEEE Internet of Things Journal*, 5(3):2146–2153, 2018.

[92] Xinchen Lyu, Wei Ni, Hui Tian, Ren Ping Liu, Xin Wang, Georgios B Giannakis, and Arogyaswami Paulraj. Optimal schedule of mobile edge computing for internet of things using partial information. *IEEE Journal on Selected Areas in Communications*, 35(11):2606–2615, 2017.

[93] Kostas Katsalis, Thanasis G Papaioannou, Navid Nikaein, and Leandros Tassiulas. Sla-driven vm scheduling in mobile edge computing. In *Cloud Computing*

(CLOUD), 2016 IEEE 9th International Conference on, pages 750–757. IEEE, 2016.

[94] Yuyi Mao, Jun Zhang, and Khaled B Letaief. Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems. In *Wireless Communications and Networking Conference (WCNC), 2017 IEEE*, pages 1–6. IEEE, 2017.

[95] Lin Wang, Lei Jiao, Dzmitry Kliazovich, and Pascal Bouvry. Reconciling task assignment and scheduling in mobile edge clouds. In *Network Protocols (ICNP), 2016 IEEE 24th International Conference on*, pages 1–6. IEEE, 2016.

[96] Gopika Premsankar, Mario Di Francesco, and Tarik Taleb. Edge computing for the internet of things: a case study. *IEEE Internet of Things Journal*, 5(2):1275–1284, 2018.

[97] Curtis Yu, Cristian Lumezanu, Abhishek Sharma, Qiang Xu, Guofei Jiang, and Harsha V Madhyastha. Software-defined latency monitoring in data center networks. In *International Conference on Passive and Active Network Measurement*, pages 360–372. Springer, 2015.

[98] Azure windows vm sizes - compute optimized. `https://docs.microsoft.com/fi-fi/azure/virtual-machines/windows/sizes-compute`.

[99] Lei Yang, Jiannong Cao, Hui Cheng, and Yusheng Ji. Multi-user computation partitioning for latency sensitive mobile cloud applications. *IEEE Transactions on Computers*, 64(8):2253–2266, 2015.

[100] Mutsunori Yagiura and Toshihide Ibaraki. The generalized assignment problem and its generalizations.

[101] David Pisinger and Mikkel Sigurd. Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. *INFORMS Journal on Computing*, 19(1):36–51, 2007.

[102] Yu N Sotskov and Natalia V Shakhlevich. Np-hardness of shop-scheduling problems with three jobs. *Discrete Applied Mathematics*, 59(3):237–266, 1995.

[103] John N Hooker and Greger Ottosson. Logic-based benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.

[104] John N Hooker. Planning and scheduling by logic-based benders decomposition. *Operations Research*, 55(3):588–602, 2007.

[105] Yingyi Chu and Quanshi Xia. Generating benders cuts for a general class of integer programming problems. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 127–141. Springer, 2004.

[106] Ruozhou Yu, Guoliang Xue, and Xiang Zhang. Application provisioning in fog computing-enabled internet-of-things: A network perspective. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 783–791. IEEE, 2018.

[107] Stephen J. Wright. Optimization. `https://www.britannica.com/science/optimization`, Oct 2016.

[108] Frederick S. Hillier and Gerald J. Lieberman. *Introduction to Operations Research*. McGraw-Hill, New York, NY, USA, seventh edition, 2001.

[109] Jacques Desrosiers and Marco E Lübbecke. A primer in column generation. In *Column generation*, pages 1–32. Springer, 2005.

[110] Toby O Davies, Graeme Gange, and Peter J Stuckey. Automatic logic-based benders decomposition with minizinc. In *AAAI*, pages 787–793, 2017.

[111] Vahid Roshanaei, Curtiss Luong, Dionne M Aleman, and David Urbach. Propagating logic-based benders' decomposition approaches for distributed operating room scheduling. *European Journal of Operational Research*, 257(2):439–455, 2017.

[112] Theodore L Turocy. game theory. *Bernhard von Stengel, London School of Economics "Game Theory" CDAM Research Report (October 2001)*, 2001.

Appendix A

Optimization and Game Theoretic Techniques

Multiple techniques including optimization and game theoretic methods are used throughout this thesis and serve as tools to solve the tackled problems. Thus, for completeness, we provide in the following a brief overview on the aforementioned techniques. For a detailed overview, interested readers are referred to [59, 60] and to [72] for more information and explanation about the different optimization and game theoretic techniques respectively.

A.1 Optimization Methods

Optimization methods have been widely used to solve multiple problems in different disciplines such as transportation, aviation, economics, etc. Mathematical programming includes the study of the mathematical structure of an optimization problem, the definition of methods to solve them and the implementation of these methods using the computer [107].

Mathematical models allow the reformulation of an optimization problem in a mathematical structure that is convenient for analysis [108]. Hence, an optimization problem can be expressed as a mathematical model composed of an objective function subject to several constraints expressed by a set of decision variables and parameters. More precisely, quantifiable decisions that are to be taken in order to solve a desired problem are expressed in terms of decision variables (i.e., x_1, x_2, ..., x_n) whose values are to be determined. These decision variables are used to express a mathematical function that represents the desired measure of performance or profit of the problem which is known as the objective function (i.e., $P = 3x_1 + 2x_2 + ... + 5x_n$). Any restrictions on the values that can be taken by the determined decision variables can be expressed in terms of mathematical expressions (i.e., equations ($3x_1 + 5x_4 \leq 4$)) known as constraints of the problem. Constants in the constraints and the objective function represents the parameters of the problem [108]. Finally, the values of the decision variables can be determined in order to maximize or to minimize the objective function.

Multiple forms of mathematical programming models exist and are elucidated in the following.

A.1.1 Linear and Non Linear Programs

Linear programming consists of optimizing (minimizing or maximizing) a linear function while satisfying a set of linear constraints. A minimization linear programming problem can be stated as follows:

$$\begin{aligned} Minimize \sum_{j=1}^{n} c_j x_j \\ subject\ to\ \sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1...m \\ x_j \geq 0, \quad j = 1...n \end{aligned} \tag{A.1}$$

where $x_1, x_2, ...x_n$ depict the decision variables of the problem, c_j, a_{ij}, b_i are constants that represent the parameters of the problem. $\sum_{j=1}^{n} c_j x_j$ is the objective function, $\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1...m$ are the functional constraints of the problem and $x_j \geq 0, \quad j = 1...n$, are the non-negativity constraints [108].

When the objective function and the constraints of the mathematical model are non-linear, the problem is named Non-Linear Program (NLP).

While there exists several methods to solve LPs (i.e., Simplex method [108, 59]), NLPs are more sophisticated to deal with. One efficient way to handle NLPs is to transform them into LPs. In this thesis, we have linearized many of the non-linear constraints of the presented mathematical models.

A.1.2 Integer Linear Programs

In the optimization problem (A.1), if all the decision variables are integers, the problem is called ILP. However, if only some of the variables are integers, the problem is referred to as MILP. Nonetheless, if all the decision variables are binary, that is, they are restricted to 0-1 values, the problem is then depicted as a Binary Linear Integer Program (BILP) [108].

Given that ILPs have fewer solutions to be considered than LPs as some feasible solutions (non-integer ones) are removed from the solution space, one may think that these problems are easier to solve than LPs, which is not the case. In fact, ILPs are usually harder to solve than LPs [108]. While IPs with a bounded feasible region are guaranteed to have a finite number of feasible solutions, this number can grow exponentially. For instance, a problem with n decision variables have 2^n solutions to be considered. Thus, adding one decision variable will double the number of possible solutions [108]. Moreover, the existence of feasible (non-integer) solutions in the solution space is a key for the efficiency of the Simplex method used to solve LPs

[108]. Thus, many methods are identified to solve ILPs. In this thesis, we used CG [57] and LBBD [103] to address this kind of problems.

A.1.3 Column Generation

To solve ILPs with a bounded solution space and finite set of feasible solutions, one can think of employing an enumeration procedure to identify the optimal solution [108]. However, as the number of possible solutions can be exponentially large, employing an enumeration method does not yield an efficient approach to solve the problem. Hence, another possible way to address the problem is to explore a clever enumeration procedure that consists of enumerating a subset of solutions only.

CG is a primal-dual decomposition approach that adopts such strategy. In fact, CG yields a classical technique that was mainly introduced to solve LPs, however, it gained success when used to solve large scale integer programs [57]. Usually, ILPs are well structured in the sense that subsets of their variables and constraints can appear in independent groups or subsystems that are linked by a distinct set of variables and/or constraints [57]. Decomposition paradigm seeks at algorithmically exploiting this specific structure of the ILP by treating the linking variables/constraints at a superior, coordinating level and independently solving each of the identified subsystems at a subordinated level [57]. CG adopts such strategy by decomposing the problem into a LP MP composed of a set of general constraints representing the predefined superior level and an (integer) LP pricing SP composed of more specific constraints and depicting the aforementioned subsystem [57, 59]. Deciding on such decomposition is not straight forward and depends on the structure of each problem. However, it allows the MP and the SP to exchange information until an optimal LP solution of the original problem is found [59]. More precisely, the MP passes revised cost coefficients or prices to the SP in order to guide it through selecting a set of non-basic variables which represents a column to be added to the MP [59].

In order to further explain the idea behind CG, we consider the following LP MP.

$$\begin{aligned} z^{\star}_{MP} &= min \sum_{j \in J} c_j \lambda_j \\ subject\ to \ &\sum_{j \in J} a_j \lambda_j \geq b \\ &\lambda_j \geq 0, \forall j \in J \end{aligned} \tag{A.2}$$

Solving the MP (A.2) using the Simplex method [59] requires identifying at each iteration of the latter a non-basic variable to enter the basis. That is, given a non-negative vector ϕ of dual variables, we seek at finding $a \quad (j \in J)$ with the minimum reduced cost ($\bar{c_j} = c_j - \phi^t a_j$). Finding such variable is costly when $|J|$ is huge. Thus, CG consists of identifying a small subset $J' \subseteq J$ of columns. Once J' is recognized, the MP is solved over the available columns in $J' \in J$ and it is called the Restricted Master Problem (RMP) which yields easier to solve given that $|J'|\leq |J|$ [109]. Thus, the RMP can be represented as follows:

$$\begin{aligned} z^{\star}_{RMP} &= min \sum_{j \in J} c_j \lambda_j \\ subject\ to \ &\sum_{j \in J} a_j \lambda_j \geq b \\ &\lambda_j \geq 0, \forall j \in J' \end{aligned} \tag{A.3}$$

The pricing SP helps in identifying the set $|J'|$ given the dual optimal solution of the current RMP. Hence, let $\phi^{\star}$ be the dual optimal solution of the RMP at a certain iteration. The objective of the pricing SP at that iteration can be defined as in Eq.(A.4) where $\phi^{\star t}$ is the transposed of vector $\phi^{\star}$, $a_j \ j \in J$ are given as elements of a set A and the cost coefficient c_j is computed from a_j via a function c.

$$c^{\star} = min\{c(a) - \phi^{\star t} a| \;\; a \in A\} \tag{A.4}$$

After solving the pricing SP, the value of $c^{\star}$ is evaluated. If $c^{\star} \geq 0$, then there exists no negative reduced cost $\bar{c}_j, \; j \in J$, thus, the RMP optimal solution obtained at the considered iteration is also optimal for the MP and the CG terminates. However, if $c^{\star} < 0$, the column derived from the SP optimal solution is added to the RMP and the process of re-optimizing the latter is repeated. Note that to initiate the CG method, an initial feasible solution to the RMP is required in order to ensure that proper dual information is passed to the pricing SP [109].

A.1.4 Logic-Based Benders Decomposition

While CG yields a column generation technique, LBBD is identified as a row generation approach that can provide the ILP optimal solution of the original problem [103]. It can be applied on any type of problem as it exploits the logical relation between its different components.

As any other decomposition approach, LBBD consists of decomposing the problem into a MP and one or many SPs [103]. Unlike CG where the MP has to be a continuous LP, the LBBD MP can take on any form (i.e., MILP). The idea behind LBBD relies in generating cuts or constraints to gradually reduce the solution space of the relaxed MP [103]. However, these cuts should not disregard any feasible solutions [103, 105]. As the MP and the SP can take on any form, LBBD does not use the linear programming dual to generate cuts, but instead, it introduces the concept of "inference dual" which can be defined as an optimization problem that finds the best possible bound implied by a set of MP variables. More precisely, the optimal solution of the MP is sent to the

SP which represents the inference dual. The SP verifies the feasibility of the provided MP optimal solution. If the latter is not feasible to the SP, the SP generates a cut and add it to the MP to eliminate this solution [103, 110]. LBBD is an iterative approach that consists of solving the MP and the SP(s) at each iteration, adding Benders' cuts from the SP(s) to the MP until the MP converges to the SP solutions [111].

The difficulty of LBBD relies in choosing a relaxed MP. For instance, if the MP does not include any important constraints, its provided solution might be too optimistic and the method converges slowly. Conversely, if the constraints are not substantially relaxed, solving the MP becomes harder. In addition, designing efficient cuts from the infeasible SP is not trivial and highly affects the performance of the LBBD [110].

A.2 Game Theory

A.2.1 Overview

Game theory was first introduced to solve strategic problems in economics, then it was extended to tackle several other fields such as computer science, biology, political science and many others. It consists of modeling, studying and analyzing strategic interactions between different entities called "players" where the latter take actions that affect each others. The players are assumed to be rational and interested in maximizing their outcomes [72]. While game theory is a formal study of decision-making in a strategic situation, a game is a formal representation of such situation [112]. Game theory is divided in two main branches:

1. *Cooperative games*

Cooperative games entail that players can communicate, cooperate, form coalitions and sign binding agreements. Cooperative games has been used for example to analyze voting behavior [72].

2. *Non-cooperative games*

   Non-cooperative games assume that players either cannot communicate or can communicate but cannot sign binding agreements. As an example, interaction between firms in competition which do not trust each others will not reach an agreement concerning prices, production, etc. [72]. Non-cooperative games can take on two different forms, a strategic form where players play simultaneously and an extensive-form where players play sequentially one after the other [72, 112].

In this thesis, we are interested in extensive-form games which we used to address the joint problem of NF mapping, traffic routing and NS scheduling (Chapter 4). Thus, we present and explain in the following extensive-form games and their aspects.

A.2.2 Extensive-Form Games

A.2.2.1 Definition

Extensive-form games, also known as dynamic games, are used to model the sequential interactions between different players (i.e., chess is an example of an extensive-form game) [72]. They are classified into perfect and imperfect information games. In perfect information games, players are aware of each others moves at any point in time, however, in imperfect information games, player will have to make a decision with only partial information about the previous moves of other players [72, 112]. In this thesis, we are interested in perfect information games.

Thus, a finite extensive-form game with perfect information can be formally defined as a tuple $\partial(P, \mathcal{K}(V, \rho), I, A, \gamma)$ where:

1. P: A finite set of players or decision-makers in the game.

2. $\mathcal{K}(V, \rho)$: Finite rooted decision tree where $V = \{v_0\} \bigcup D \bigcup T$ consists of a set V of nodes depicted by a root node v_0, a set D of decision or strategic nodes, each assigned to a player, and a set T of terminal nodes. ρ is an immediate predecessor function $\rho : V \to D$. The decision tree provides a complete description of how the game is played over time [112].

3. I: Information set of the game which includes the information (i.e., order of the players in taking actions, previous actions, etc.) that players have at the time when they must take action. I includes the information set of each player in the game which is available at each node $d \in D$ dedicated for that player [72, 112, 75].

4. A: Set of actions available during the game.

5. $\gamma = (\gamma_p)_{p \in P} : T \to \mathbb{N}^P$ is a payoff function that assigns payoffs to players as a function of the terminal node reached. A payoff, also called utility, reflects the desirability of a player to an outcome [112, 75].

As an example, we consider the decision tree depicted in Fig.A.1 which represents a finite extensive-form game with perfect information of two players, player 1 and player 2, who decided to dissolve a business partnership which assets have been valued to 100 000\$. Player 1 is the senior partner who can make an offer to divide the assets. Thus, player 1 has two different actions representing the offers he/she can make, which are a $50 - 50$ or a $70 - 30$ split of assets. Player 2, the junior partner, can respond to player 1 offer by choosing one of two possible actions, accept or reject player 1's offer.

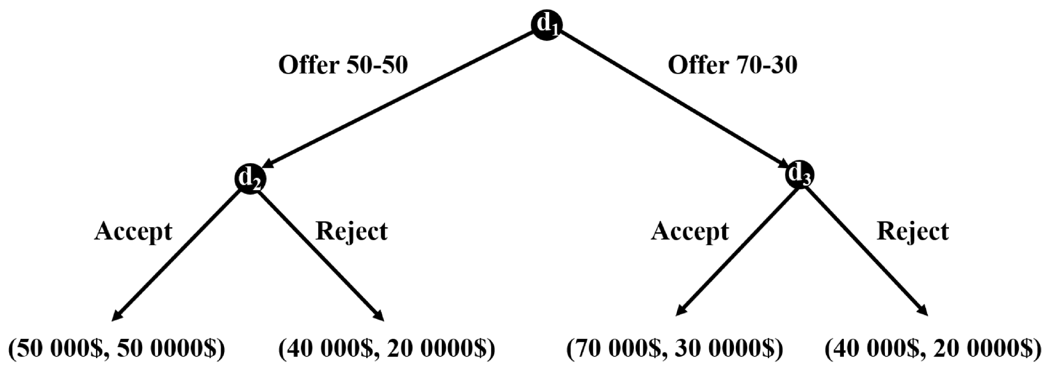


Figure A.1: Extensive-form game decision tree.

Hence, the strategies of player 1 are {offer $50-50$, offer $70-30$}. The strategies of player 2 are {{Accept, Accept}, {Accept, Reject}, {Reject, Accept}, {Reject, Reject}}. Note that each terminal node in Fig.A.1, is mounted with two values (x, y) where x represents the payoff of player 1 and y is the payoff of player 2. For instance, if player 1 makes an offer $50-50$ to player 2 who rejects it, player 1 will receive 40000\$ while player 2 will be granted 20000\$ by the court (legal fees were deducted from the total amount of the assets) [72].

A.2.2.2 Strategies

When playing, each player needs to decide on its strategy. A game can be played with either a pure strategy or a mixed strategy.

1. *Pure strategy*

   In a strategic game, a player strategy is one of its possible actions. However, in an extensive-form game, a strategy is a complete plan of actions, detailing the behavior of a player in every situation [112, 75].

2. *Mixed strategy*

   A mixed strategy for a player $p \in P$ is a probability distribution over the set of its pure strategies such that all probabilities over the player's strategy set add to 1. That is, a player may randomly select a strategy among its pure strategies with a certain probability [72, 75]. In the example in Fig A.1, player 1 might choose to play a mixed strategy with a probability q to play the offer $50 - 50$ and a probability $(1 - q)$ to choose to offer $70 - 30$.

A.2.2.3 Nash Equilibrium

Usually, each player in the game is interested in maximizing its payoff. Hence, each player will use the strategy that is the best response to the strategies selected by its opponents. The equilibrium depicts that none of the players has incentive to deviate from its selected strategy [72]. Note that a game in strategic form may not always have a pure strategy Nash equilibrium in which each player makes a deterministic choice of his strategies (i.e., coin flipping game). Thus, players may choose to randomly select a strategy from their strategy set with a certain probability. Note that John Nash showed that every finite game in strategic form has at least one mixed strategy Nash equilibrium [112]. Further, it is worth noting that an extensive-form game can be converted to a strategic form game.

Definition A.1. *A strategy profile $s^\star$ is a Nash equilibrium if for all $p \in P$ and strategy s_p of player p it holds that:*

$$\gamma(s_p, s^\star_{-p}) \leq \gamma(s^\star_p, s^\star_{-p}) \quad \forall p \in P \tag{A.5}$$

where $s^\star_p$ is the best strategy played by player p in response to the best strategies of its opponents ($s^\star_{-p}$) [72, 74].

The equilibrium notion for extensive-form game is the *Sub-game Perfect Nash Equilibrium.* A sub-game of an extensive-form game is a portion of the game that could be treated as a game itself. It starts from a decision node $d \in D$ (different than the root node), whose information set consists of the node d only and enclose in an oval node d and all its successors [72, 74].

Definition A.2. *A strategy profile $s^\star$ is a Sub-game Perfect Nash equilibrium in game ∂ if for any sub-game ∂' of ∂, $s^\star|\partial'$ is a Nash equilibrium of ∂' [72, 74].*

A.2.2.4 Backward Induction

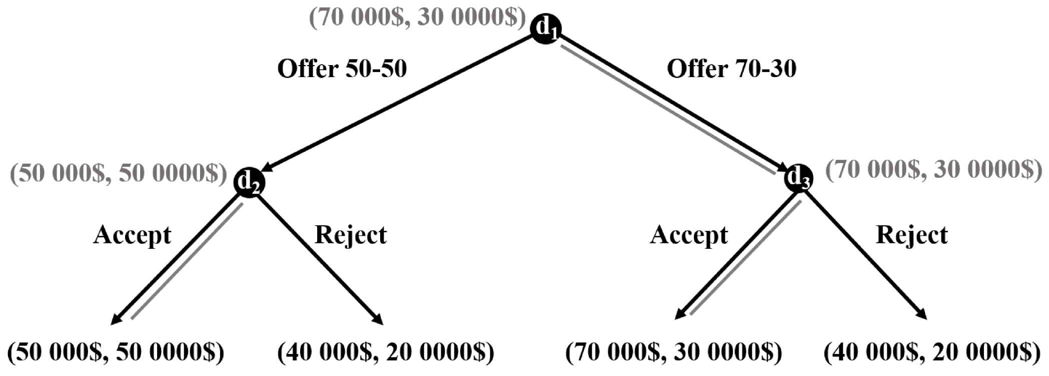


Figure A.2: Representation of backward induction reasoning.

Extensive-form games can be analyzed using backward induction. Backward induction is a technique used to solve games with perfect information. It allows to identify the sub-game perfect Nash equilibrium by starting at the end of the game tree $\mathcal{K}(V, \rho)$, and reasoning backward up the tree while solving for the optimal behavior at each node. In other words, it identifies the equilibrium in the bottom most tree, and adopt these as one moves up the tree.

Hence, in order to apply the backward induction reasoning in the example depicted in Fig.A.1, we start by reasoning at node d_2 of player 2 where player 2 will choose to accept the offer given that it will provide it with higher payoff (50 000$) than rejecting it (20 000$). Hence, we mark the node d_2 with the payoff vector $(50\ 000\$, 50\ 000\$)$ corresponding to the selected choice (Fig.A.2). Similarly, at node d_3, player 2 will choose to accept the offer as well. Hence, we mark the node d_3 with the payoff vector $(70\,000\$, 30\,000\$)$. Considering the payoff vectors at node d_2 and d_3, player 1 (at node d_1) will select to make an offer of $70 - 30$ split as it will maximize its payoff. Thus, we conclude that player 1 will make an offer of $70 - 30$ split which will be accepted by player 2. Note that the choices selected by backward induction are highlighted by doubling the corresponding edges (Fig.A.2).

Appendix B

Linearization of SFCS Problem

We present in the following the linearization details for the non linear constraints (Eq.(3.14) and Eq.(3.18)) in the SFCS problem formulation (Chapter 3, Section 3.3).

Thus, to linearize Eq.(3.14) we replace it with the following equations:

$$h_{ns}^{k} \leq q_{ns}^{k} \quad \substack{\forall k \in K_p \\ \forall n \in N_s \\ \forall s \in S} \tag{B.1}$$

$$h_{ns}^{k} \leq q_{(n+1)s}^{k} \quad \substack{\forall k \in K_p \\ \forall n,(n+1) \in N_s \\ \forall s \in S} \tag{B.2}$$

$$h_{ns}^{k} \geq q_{ns}^{k} + q_{(n+1)s}^{k} - 1 \quad \substack{\forall k \in K_p \\ \forall n,(n+1) \in N_s \\ \forall s \in S} \tag{B.3}$$

Eq.(3.18) can be linearized by declaring a new binary decision variable $g_{ij}^{s\delta e} \in \{0,1\}$ as follows:

$$g_{ij}^{s\delta e} = l_{ij}^{e} \hat{\theta}_{s}^{\delta e} \quad \substack{\forall \delta \in \Delta \\ \forall (ij) \in L \\ \forall e \in E_s \\ \forall s \in S} \tag{B.4}$$

Eq.(3.18) can then be replaced by the following set of equations:

$$g_{ij}^{s\delta e} \leq l_{ij}^{e} \quad \substack{\forall \delta \in \Delta \\ \forall (ij) \in L \\ \forall e \in E_s \\ \forall s \in S} \tag{B.5}$$

$$g_{ij}^{s\delta e} \leq \hat{\theta}_s^{\delta e} \quad \substack{\forall \delta \in \Delta \\ \forall (ij) \in L \\ \forall e \in E_s \\ \forall s \in S} \tag{B.6}$$

$$g_{ij}^{s\delta e} \geq l_{ij}^{e} + \hat{\theta}_s^{\delta e} - 1 \quad \substack{\forall \delta \in \Delta \\ \forall (ij) \in L \\ \forall e \in E_s \\ \forall s \in S} \tag{B.7}$$

$$\sum_{s \in S} \sum_{e \in E_s} g_{ij}^{s\delta e} b_s \leq c_{ij} \quad \substack{\forall \delta \in \Delta \\ \forall (ij) \in L} \tag{B.8}$$

Appendix C

Linearization of the LASS Problem

In the following, we provide the linearization details for the non linear constraints in LASS-MaxAdmission (Chapter 4 Section 4.3).

Hence, to linearize Eq.(4.9) we replace it with the following three equations:

$$h^k_{ns} \leq q^k_{ns} \quad \substack{\forall k \in K_p \\ \forall n \in N_s \\ \forall s \in S} \tag{C.1}$$

$$h^k_{ns} \leq q^k_{(n+1)s} \quad \substack{\forall k \in K_p \\ \forall n,(n+1) \in N_s \\ \forall s \in S} \tag{C.2}$$

$$h^k_{ns} \geq q^k_{ns} + q^k_{(n+1)s} - 1 \quad \substack{\forall k \in K_p \\ \forall n,(n+1) \in N_s \\ \forall s \in S} \tag{C.3}$$

Eq.(4.10) is non linear and can be linearized by replacing it with the following three equations:

$$\sum_{\delta \in \Delta} \theta^{\delta e}_s \leq a_s \quad \substack{\forall e \in E_s \\ \forall s \in S} \tag{C.4}$$

$$\sum_{\delta \in \Delta} \theta^{\delta e}_s \leq 1 - \sum_{k \in K_p} h^k_{o(e)s} \quad \substack{\forall e \in E_s \\ \forall s \in S} \tag{C.5}$$

$$\sum_{\delta \in \Delta} \theta^{\delta e}_s \geq a_s - \sum_{k \in K_p} h^k_{o(e)s} \quad \substack{\forall e \in E_s \\ \forall s \in S} \tag{C.6}$$

Eq.(4.17) is non linear and can be linearized by declaring a new decision variable

$r_{ns}^{f\delta} \in \{0,1\}$ such that:

$$r_{d(e)s}^{f\delta'} = \theta_s^{\delta' e} \sum_{\delta'' \in \Delta} y_{d(e)s}^{f\delta''} \quad \substack{\forall f \in F \\ \forall \delta' \in \Delta \\ \forall e \in E_s \\ \forall s \in S} \tag{C.7}$$

Eq.(4.17) can then be replaced by the following equations:

$$r_{d(e)s}^{f\delta'} \leq \theta_s^{\delta' e} \quad \substack{\forall f \in F \\ \forall \delta' \in \Delta \\ \forall e \in E_s \\ \forall s \in S} \tag{C.8}$$

$$r_{d(e)s}^{f\delta'} \leq \sum_{\delta'' \in \Delta} y_{d(e)s}^{f\delta''} \quad \substack{\forall f \in F \\ \forall \delta' \in \Delta \\ \forall e \in E_s \\ \forall s \in S} \tag{C.9}$$

$$r_{d(e)s}^{f\delta'} \geq \theta_s^{\delta' e} + \sum_{\delta'' \in \Delta} y_{d(e)s}^{f\delta''} - 1 \quad \substack{\forall f \in F \\ \forall \delta' \in \Delta \\ \forall e \in E_s \\ \forall s \in S} \tag{C.10}$$

$$\psi_s^{f\delta} \leq 1 - r_{d(e)s}^{f\delta'} \quad \substack{\forall \delta, \delta' \in \Delta : \delta < \delta' + \frac{w_s}{b_s} \\ \forall e \in E_s \\ \forall f \in F \\ \forall s \in S} \tag{C.11}$$

Eq.(4.23) is non linear and can be linearized by declaring a new binary decision variable $g_{ij}^{s\delta e} \in \{0,1\}$ as follows:

$$g_{ij}^{s\delta e} = l_{ij}^e \hat{\theta}_s^{\delta e} \quad \substack{\forall \delta \in \Delta \\ \forall (ij) \in L \\ \forall e \in E_s \\ \forall s \in S} \tag{C.12}$$

Eq.(4.23) can then be replaced by the following equations:

$$g_{ij}^{s\delta e} \leq l_{ij}^e \quad \substack{\forall \delta \in \Delta \\ \forall (ij) \in L \\ \forall e \in E_s \\ \forall s \in S} \tag{C.13}$$

$$g_{ij}^{s\delta e} \leq \hat{\theta}_s^{\delta e} \quad \substack{\forall \delta \in \Delta \\ \forall (ij) \in L \\ \forall e \in E_s \\ \forall s \in S} \tag{C.14}$$

$$g_{ij}^{s\delta e} \geq l_{ij}^e + \hat{\theta}_s^{\delta e} - 1 \quad \substack{\forall \delta \in \Delta \\ \forall (ij) \in L \\ \forall e \in E_s \\ \forall s \in S} \tag{C.15}$$

$$\sum_{s \in S} \sum_{e \in E_s} g_{ij}^{s\delta e} b_s \leq c_{ij} \quad \substack{\forall \delta \in \Delta \\ \forall (ij) \in L} \tag{C.16}$$

Appendix D

Formulation Details of the DTOS Problem

We provide, in the following, additional details about the DTOS-MILP (Section 5.4) and DTOS-LBBD (Section 5.5) formulations presented in Chapter 5.

D.1 DTOS-MILP

D.1.1 Constraints Reformulation

As we replace p_a by $\sum_{p \in P} z_a^p p$ in constraints (5.5), (5.6) and (5.7), we show below how these constraints can be rewritten after this change.

Eq.(5.5) can be rewritten as specified in Eq.(D.1).

$$\sum_{p \in P} z_a^p p \geq n_a p_{min}^a \quad \forall a \in A \tag{D.1}$$

Eq.(5.6) is rewritten as in Eq.(D.2).

$$\sum_{p\in P} z_a^p p \leq n_a \sum_{m\in M} x_m^a c_m \quad \forall a \in A \tag{D.2}$$

Eq.(5.7) is reformulated as in Eq.(D.3).

$$\sum_{a\in A} x_m^a \sum_{p\in P} z_a^p p \leq c_m \quad {\scriptstyle \forall m\in M} \tag{D.3}$$

D.1.2 Linearization Details

We explain below the linearization details of the non linear constraints of the DTOS problem. The non linearity of Eqs.(5.11), (5.12) and (5.17) is due to the term d^u_{proc} (Eq.(5.19)). Hence, Eq.(5.19) can be linearized by declaring a new decision variable $w_{pu}^{a\delta} \in \{0,1\}$ such that:

$$w_{pu}^{a\delta} = y_u^{a\delta} z_a^p \quad \substack{\forall p\in P\\ \forall a\in A\\ \forall \delta\in\Delta\\ \forall u\in U} \tag{D.4}$$

Eq.(5.19) can then be replaced by the following equations:

$$w_{pu}^{a\delta} \leq y_u^{a\delta} \quad \substack{\forall p\in P\\ \forall a\in A\\ \forall \delta\in\Delta\\ \forall u\in U} \tag{D.5}$$

$$w_{pu}^{a\delta} \leq z_a^p \quad \substack{\forall p\in P\\ \forall a\in A\\ \forall \delta\in\Delta\\ \forall u\in U} \tag{D.6}$$

$$w_{pu}^{a\delta} \geq y_u^{a\delta} + z_a^p - 1 \quad \substack{\forall p\in P\\ \forall a\in A\\ \forall \delta\in\Delta\\ \forall u\in U} \tag{D.7}$$

$$d^u_{proc} = \sum_{p\in P}\sum_{a\in A}\sum_{\delta\in\Delta} w_{pu}^{a\delta}\frac{\mu_u}{p} \quad \forall u \in U \tag{D.8}$$

Thus, Eq.(5.11) can be written in a linearized form by replacing $d^u_{process}$ by Eq.(D.8) as depicted in Eq.(D.9).

$$\sum_{\delta \in \Delta} y^{a\delta}_{u'} \delta \geq \sum_{\delta \in \Delta} y^{a\delta}_{u} \delta + \sum_{p \in P} \sum_{a \in A} \sum_{\delta \in \Delta} w^{a\delta}_{pu} \frac{\mu_u}{p} - H(1 - s^a_{uu'}) \quad {\scriptstyle \forall a \in A:(t_u = t_{u'} = t_a) \atop \forall u,u' \in U:(u! = u')} \tag{D.9}$$

Similarly, Eq.(5.12) can be written in a linearized form as shown in Eq.(D.10).

$$\sum_{\delta \in \Delta} y^{a\delta}_{u} \delta \geq \sum_{\delta \in \Delta} y^{a\delta}_{u'} \delta + \sum_{p \in P} \sum_{a \in A} \sum_{\delta \in \Delta} w^{a\delta}_{pu'} \frac{\mu_{u'}}{p} - H(1 - s^a_{u'u}) \quad {\scriptstyle \forall a \in A:(t_u = t_{u'} = t_a) \atop \forall u,u' \in U:(u! = u')} \tag{D.10}$$

In a similar manner, Eq.(5.17) can be written in a linearized form as specified in Eq.(D.11)

$$\sum_{a \in A} \sum_{\delta \in \Delta} y^{a\delta}_{u} \delta + \sum_{p \in P} \sum_{a \in A} \sum_{\delta \in \Delta} w^{a\delta}_{pu} \frac{\mu_u}{p} \leq \theta_u \quad \forall u \in U \tag{D.11}$$

Eq.(5.14) is non linear and can be linearized by declaring a new decision variable $\rho^a_{uu'} \in \{0, 1\}$ such that:

$$\rho^a_{uu'} = \sum_{\delta \in \Delta} y^{a\delta}_{u} \sum_{\delta \in \Delta} y^{a\delta}_{u'} \quad {\scriptstyle \forall a \in A:(t_u = t_{u'} = t_a) \atop \forall u,u' \in U:(u! = u')} \tag{D.12}$$

Eq.(5.14) can then be replaced by the following equations:

$$\rho^a_{uu'} \leq \sum_{\delta \in \Delta} y^{a\delta}_{u} \quad {\scriptstyle \forall a \in A:(t_u = t_{u'} = t_a) \atop \forall u,u' \in U:(u! = u')} \tag{D.13}$$

$$\rho^a_{uu'} \leq \sum_{\delta \in \Delta} y^{a\delta}_{u'} \quad {\scriptstyle \forall a \in A:(t_u = t_{u'} = t_a) \atop \forall u,u' \in U:(u! = u')} \tag{D.14}$$

$$\rho^a_{uu'} \geq \sum_{\delta \in \Delta} y^{a\delta}_{u} + \sum_{\delta \in \Delta} y^{a\delta}_{u'} - 1 \quad {\scriptstyle \forall a \in A:(t_u = t_{u'} = t_a) \atop \forall u,u' \in U:(u! = u')} \tag{D.15}$$

$$s^a_{uu'} + s^a_{u'u} = \rho^a_{uu'} \quad {\scriptstyle \forall a \in A:(t_u = t_{u'} = t_a) \atop \forall u,u' \in U:(u! = u')} \tag{D.16}$$

D.2 DTOS-LBBD

D.2.1 Constraints Reformulation

Given that we replace p_a by $\sum_{p\in P} z_a^p p$ in the formulation of the MP, Eq.(5.25), Eq.(5.26), Eq.5.27, Eq.(5.28) and Eq.(5.30) can be respectively rewritten as in Eq.(D.1), Eq.(D.2), Eq.(D.3), Eq.D.17 and Eq.D.18.

$$\sum_{p\in P} z_a^p p \geq \frac{\sum_{u\in U:(t_u=t_a)} \mu_u q_u^a}{\theta_{max}^a - \sigma_{min}^a} \quad \forall a \in A \tag{D.17}$$

$$\sum_{p\in P} z_a^p p \geq \sum_{j\in P_u^a} j\beta_{ua}^j \ {}_{\forall a\in A}^{\forall u\in U} \tag{D.18}$$

D.2.2 Linearization Details

We explain in the following the linearization details of the non linear constraints of the DTOS-LBBD SP.

Eq.(5.37) is non linear and can be linearized by declaring a new decision variable $\varrho_{uu'} \in \{0,1\}$ such that:

$$\varrho_{uu'} = \alpha_{u'}\alpha_u \quad \forall u, u' \in U_a : (u! = u') \tag{D.19}$$

Eq.(5.37) can then be replaced by the following equations:

$$\varrho_{uu'} \leq \alpha_{u'} \quad \forall u, u' \in U_a : (u! = u') \tag{D.20}$$

$$\varrho_{uu'} \leq \alpha_u \quad \forall u, u' \in U_a : (u! = u') \tag{D.21}$$

$$\varrho_{uu'} \geq \alpha_u + \alpha_{u'} - 1 \quad \forall u, u' \in U_a : (u! = u') \tag{D.22}$$

$$s_{uu'} + s_{u'u} = \varrho_{uu'} \quad \forall u, u' \in U_a : (u! = u') \tag{D.23}$$