

New control of networks / IT : Performance analysis of virtualized network functions for a programmable infrastructure

Verónica Karina Quintuna Rodriguez

▶ **To cite this version:**

Verónica Karina Quintuna Rodriguez. New control of networks / IT : Performance analysis of virtualized network functions for a programmable infrastructure. Networking and Internet Architecture [cs.NI]. Sorbonne Université, 2018. English. NNT : 2018SORUS099 . tel-02612498v2

HAL Id: tel-02612498

https://theses.hal.science/tel-02612498v2

Submitted on 19 May 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Sorbonne Université
École Doctorale EDITE de Paris
Inria - RAP

Nouvelle commande réseau / IT : Performance des fonctions virtualisées pour une infrastructure programmable

Par **Verónica QUINTUNA RODRIGUEZ**

Thèse de doctorat en
Sciences de l'Information et de la Communication
Dirigée par Philippe Robert et Fabrice Guillemin

Présentée et soutenue publiquement le 03 octobre 2018

Devant un jury composé de :

M. André-Luc Beylot	IRIT	Rapporteur
Mme. Annie Gravey	IMT Atlantique	Rapporteur
M. Fabrice Guillemin	Orange Labs	Directeur
M. Raymond Knopp	Eurecom	Examinateur
M. Paul Muhlethaler	Inria-EVA	Examinateur
M. Hervé Rivano	INSA Lyon	Rapporteur
M. Philippe Robert	Inria-RAP	Directeur
M. Pierre Sens	Inria-REGAL	Examinateur

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisors: Dr. Fabrice Guillemin and Dr. Philippe Robert, for their continuous guidance and insightful assistance. Without their precious support it would not be possible to conduct this research.

I should thank each and every one of ARC's team-members for their encouragement but also for promoting me to widen my research from various perspectives. I would particularly like to express my gratitude to Nathalie Labidurie for constantly motivating me.

A special thanks goes to my family, notably my grand parents Miguel and Elena, my aunts Ligia and Alicia, my mom Laura, and last but not least, my brother David and my little sister Tadea.

Abstract

Keywords: *Virtualization, NFV, VNF, queuing systems, resource pooling, Cloud-RAN, C-RAN, parallel programming, processor sharing, cloud computing, scheduling, service chaining.*

In the framework of Network Function Virtualization (NFV), we address in this work the performance analysis of virtualized network functions (VNFs), wherein the virtualization of the radio access network (namely, Cloud-RAN) is the driving use-case. The overarching principle of network virtualization consists of replacing network functions, which were so far running on dedicated and proprietary hardware, with open software applications running on shared general purpose servers. The complexity of virtualization is in the softwarization of low-layer network functions (namely, PHY functions) because their execution must meet strict latency requirements.

Throughout this work, we evaluate the performance of VNFs in terms of latency which considers the total amount of time that is required to process VNFs in cloud computing systems. We notably investigate the relevance of resource pooling and statistical multiplexing when available cores in a data center are shared by all active VNFs. We perform VNF modeling by means of stochastic service systems. Proposed queuing models reveal the behavior of high performance computing architectures based on parallel processing and enable dimensioning the required computing capacity in data centers.

We concretely investigate the $M^{[X]}/M/1$ queuing system with Processor-Sharing discipline, for representing the simultaneous execution of VNF's jobs through a simple model. We notably consider VNFs composed of a set of parallel runnable jobs forming batches. The execution time of the entire VNF is therefore determined by the runtime of individual jobs. The job's sojourn time distribution can then be used for dimensioning purposes in order to guarantee that, with a large probability, the service of a job is completed before some lag time.

In the case of Cloud-RAN, the sojourn time of virtualized RAN functions in the cloud must respect tight time budgets. We notably study the runtime of virtual RAN functions by using Open Air Interface (OAI), an open-source solution which implements the RAN functionality in software. In order to reduce latency, we investigate the functional and data decomposition of RAN functions, which leads to batch arrivals of parallel runnable jobs with non-deterministic runtime. To assess the required processing capacity when hosting Cloud-RAN systems, we introduce a bulk arrival queuing model, namely the $M^{[X]}/M/C$ queuing system, where the batch size follows a geometric distribution. The variability of the fronthaul delay and job's runtime are captured by the arrival and service distributions, respectively. Since the runtime of a radio subframe becomes the batch sojourn-time, we have derived the Laplace transform of this latter quantity as well as the probability of exceeding certain threshold to respect RAN deadlines. We validate by simulation the effectiveness of the $M^{[X]}/M/C$ model while considering

the behavior of a real Cloud-RAN system. For this purpose, we fed the queuing system with statistical parameters captured from the OAI-based Cloud-RAN emulation. Results provide valuable guidelines for sizing and deploying Cloud-RAN systems.

As a proof of concept, we implement an end-to-end virtualized mobile network which notably confirms the accuracy of theoretical models. Performance results highlight important gains in terms of latency. This fact particularly enables increasing the concentration level of VNFs in data centers for achieving CAPEX and OPEX reduction and, moreover, it opens the door to the cloudification of critical network functions.

Contents

Abstract ii

List of Figures x

List of Tables xi

Acronyms xii

Introduction xvii

1 Network Function Virtualization: State of the Art 1

1.1 Virtualization background . . . 1
1.1.1 Virtual Machines . . . 2
1.1.2 Containers . . . 3
1.2 Microservices as an IT paradigm . . . 4
1.2.1 Monolithic and Microservice-based applications . . . 4
1.2.2 Microservices and Component based architectures . . . 5
1.2.3 Microservices and Service Oriented Architectures . . . 5
1.2.4 Implementing microservices within containers . . . 5
1.3 Virtualized Network Functions . . . 6
1.3.1 Formal definition . . . 6
1.3.2 Architectural framework . . . 6
1.3.3 A virtual Network Service . . . 7
1.3.4 Implementing VNFs as microservices . . . 8
1.4 VNFs placement . . . 9
1.4.1 VNFs placement as an optimization problem . . . 11
1.4.2 Multi-site placement by heuristic approaches . . . 11
1.4.3 Multi-provider scenarios . . . 12
1.4.4 Multi-objective solutions . . . 12
1.4.5 Dynamic placement solutions . . . 12
1.5 Resource Sharing . . . 14
1.5.1 Fairness as the basis of resource allocation . . . 14
1.5.2 Mono-resource allocation . . . 15
1.5.3 Multi-resource allocation . . . 16
1.5.4 General Fairness Criterion . . . 19

2 VNF modeling **23**
2.1 General Assumptions . . . 23
2.2 Use cases . . . 24
2.2.1 Virtualizing packet core network functions . . . 24
2.2.2 Virtualizing wireless network functions . . . 24
2.3 Model description . . . 26
2.3.1 Model settings . . . 26
2.3.2 Representation of a VNF . . . 27
2.3.3 Queuing system formulation . . . 28
2.3.4 Performance Indicators . . . 29
2.4 Scheduling algorithms . . . 29
2.4.1 Allocating the entire macro-function to a Dedicated Core . . . 29
2.4.2 Allocating sub-functions by Round Robin criterion . . . 30
2.4.3 Greedy allocation of sub-functions . . . 30
2.5 Performance Analysis . . . 30
2.5.1 Simulation settings . . . 30
2.5.2 Scheduling performance without reneging . . . 31
2.5.3 Scheduling performance when considering a deadline . . . 31
2.5.4 Analysis of results . . . 33
2.6 A queuing model based on concurrent processing . . . 34
2.6.1 General principles of concurrent computing . . . 34
2.6.2 Model settings . . . 34
2.6.3 Queuing formulation . . . 34
2.6.4 Job's sojourn time . . . 35
2.6.5 Batch's sojourn time . . . 38

3 Cloud-RAN as an NFV use-case **41**
3.1 System description . . . 41
3.1.1 Architectural framework . . . 42
3.1.2 Implementation guidelines . . . 42
3.1.3 Functional splits . . . 44
3.2 Fronthaul analysis . . . 44
3.2.1 Fronthaul size . . . 44
3.2.2 Fronthaul capacity . . . 46
3.2.3 Fronthaul transmission protocols . . . 46
3.2.4 Required capacity per Functional Split . . . 46
3.3 Runtime evaluation . . . 51
3.3.1 Service chain . . . 51
3.3.2 VNF's runtime . . . 52
3.3.3 Runtime acceleration . . . 54
3.3.4 Performance gain . . . 58
3.4 Worst-case study . . . 59
3.4.1 Worst-case definition . . . 59
3.4.2 BBU-pool analysis . . . 59
3.4.3 Cloud-RAN sizing . . . 61
3.4.4 Resource pooling performance . . . 61

4 C-RAN modeling for dimensioning purposes 65
4.1 Modeling Principles . . . 65
4.1.1 Modeling data processing . . . 65
4.1.2 Parallelism by UEs . . . 67
4.1.3 Parallelism by CBs . . . 69
4.1.4 No parallelism . . . 69
4.2 Batch model . . . 69
4.2.1 Analysis of the first case . . . 70
4.2.2 Analysis of the second case . . . 72
4.2.3 Main result . . . 72
4.3 Numerical experiments . . . 75
4.3.1 Simulation settings . . . 75
4.3.2 Model analysis . . . 75
4.3.3 Cloud-RAN dimensioning . . . 77
4.3.4 Performance evaluation . . . 77
4.3.5 Analysis of results . . . 78

5 Proof of Concept: C-RAN acceleration 81
5.1 Test-bed description . . . 81
5.2 Implementation outline . . . 83
5.2.1 Encoding function . . . 83
5.2.2 Decoding function . . . 84
5.2.3 Thread-pool . . . 84
5.2.4 Queuing principles . . . 86
5.2.5 Performance captor . . . 86
5.3 Performance evaluation . . . 87

6 Conclusions 89
6.1 Main contributions . . . 89
6.2 Major outcomes . . . 91
6.3 Research perspectives . . . 92
6.4 Publications . . . 92

Appendices 95

A Sojourn time in an $M^{[X]}/M/1$ Processor Sharing Queue 97

B Cloud-RAN applications 103

C Fronthaul capacity reduction for Functional Split I 105

D Resource Allocation Grid 107

E Scheduling Strategy 109

F Analysis of the Markov chain considering an $M^{[X]}/M/C$ system 111

Bibliography 114

List of Figures

1.1 Virtual and Physical Machine Architectures [1]. 2
1.2 Virtual Machine architectures. 3
1.3 Container architecture. 4
1.4 Monolithic and microservice-based applications. 4
1.5 NFV architectural framework [2]. 6
1.6 End-to-end network service [2]. 7
1.7 Forwarding graph of VNFs [2]. 8
1.8 An example of service chaining: Mobile Core Network. 9
1.9 Microservices involved in a end-to-end IMS service [3]. 10
1.10 Key elements in VNFs placement. 11
1.11 CDF of demand-to-slot ratio in slot-based allocations [4]. 17
1.12 An example of 'DRF' and 'Asset fairness' allocations [4]. 19
1.13 Jain's fairness index as a function of α [5]. 20

2.1 Network Function Virtualization, use-cases. 25
2.2 A VNF as a chain of sub-functions. 27
2.3 Architecture of the virtualization environment. 27
2.4 VNF modeling: Elements of the queuing system. 28
2.5 Scheduling performance considering chained sub-functions. 31
2.6 Scheduling performance considering no chained sub-functions. 32
2.7 Scheduling performance considering chained sub-functions and reneging. 32
2.8 Scheduling performance considering no chained sub-functions and reneging. . . . 33
2.9 Processor-Sharing queue for the modeling of parallelized batch service [6]. 35
2.10 Closed integration contour avoiding the real axis [6]. 36
2.11 Function $x \mapsto \mathbb{P}(W > x)$ for different values of the pair (ϱ, q) with load $\varrho^* = \varrho/(1-q)$ fixed to 0.8 [6]. 37
2.12 Function $y \mapsto \mathbb{P}(V^{(0)} > y)$ and its asymptotics for large y [6]. 38
2.13 Distribution $D_q : x \mapsto \mathbb{P}(\Omega > x)$ of the batch sojourn time and its approximation [6]. 39

3.1 Cloud-RAN architecture. 42
3.2 Cloud-RAN virtualization environment. 43
3.3 Cloud-RAN functional splits 44
3.4 HARQ process in Cloud-RAN architectures. 45
3.5 Fronthaul radio interfaces according to the various functional splits. 47
3.6 BBU functions. 51
3.7 An example of commercial radio scheduler. 52

3.8 Emulated radio resource grid. . . . 53
3.9 Runtime of virtual BBU functions (PHY layer, uplink). . . . 54
3.10 Runtime of virtual BBU functions (PHY layer, downlink). . . . 54
3.11 Functional and data decomposition of BBU functions: Multi-threading model. . . 56
3.12 Radio data units. . . . 57
3.13 PHY threading model. . . . 57
3.14 MIMO threading model. . . . 57
3.15 Channel decoding performance in a single eNB, $C = 6$. . . . 58
3.16 Cloud-RAN processing, worst-case analysis. . . . 59
3.17 Channel coding runtime as a function of the MCS, 100 RBs. . . . 60
3.18 Downlink workload of a BBU-pool, worst-case scenario. . . . 60
3.19 Uplink workload of a BBU-pool, worst-case scenario. . . . 61
3.20 BBU-pool performance (10 eNBs), worst-case scenario. . . . 62
3.21 Probability density function of channel coding runtime, 10 eNBs, $C = 24$. . . . 63
3.22 Cloud-RAN performance (UL+DL) for real traffic conditions, 10 eNBs. . . . 63
3.23 Decoding performance (UL), 10 eNBs. . . . 64
3.24 Encoding performance (DL), 10 eNBs. . . . 64

4.1 Parallel processing models [7]. . . . 66
4.2 Performance of parallel processing systems [7]. . . . 67
4.3 Stochastic service system for Cloud-RAN. . . . 68
4.4 Two cases upon the arrival of a batch. . . . 70
4.5 Statistical parameters of Cloud-RAN. . . . 76
4.6 $M^{[x]}/M/C$ behavior. . . . 77
4.7 C-RAN sizing when using the $M^{[X]}/M/C$ model. . . . 78
4.8 Cloud-RAN performance, 100 eNBs, $C = 151$. . . . 79
4.9 CDF of the sojourn time of radio subframes. . . . 79

5.1 Test-bed architecture. . . . 82
5.2 Block diagram of encoding function. . . . 83
5.3 Block diagram of decoding function. . . . 84
5.4 Multi-threading implementation. . . . 85
5.5 Decoding runtime (test-bed). . . . 88
5.6 Encoding runtime (test-bed). . . . 88

E.1 Global scheduler. . . . 110

List of Tables

1.1 Example of Dominant Resource Fairness (DRF) resource allocation. 18
1.2 Generalized α-fair resource allocation in wireless networks: Relation among different α- fair allocation for three utility functions. 21

2.1 Scheduling performance with chained sub-functions. 32
2.2 Scheduling performance with no chained sub-functions. 33

3.1 Required fronthaul capacity in a Cloud-RAN system. 48
3.2 List of symbols. 49
3.3 Loss rate of subframes in a BBU-pool of 10 eNBs. 62

C.1 Useful RAN bandwidth. 105

D.1 Key Features of Modulation and Coding when $N_{RB} = 100$. 107
D.2 MCS and TBS correlation. 107
D.3 An example of TBS as a function of N_{RB}. 108

Acronyms

ABI	Application Binary Interface.
ADC	Analogic-Digital Converter.
AGC	Automatic Gain Control.
API	Application Programing Interface.
ATM	Asynchronous transfer mode.
BBU	Base Band Unit.
BER	Bit Error Rate.
C-EPC	Cloud-EPC.
CB	Code Block.
CBR	Constant Bit Rate.
CC	Channel Coding.
CCDU	Channel Coding Data Unit.
CCO	Core Central Office.
CDC	Centralized Data Center.
CDN	Content Delivery Network.
CO	Central Office.
CoMP	Coordinated Multi-point.
COTS	Commercial off-the-shelf.
CP	Cyclic Prefix.
CPRI	Common Public Radio Interface.
CPU	Central Processing Unit.
CQI	Channel Quality Indicator.
CRC	Cyclic Redundancy Check.
CU	Central Unit.
DC	Dedicated Core.
Diah	Diameter handler.
DL	downlink.
DRF	Dominant Resource Fairness.
DRFH	DRF in Heterogeneous environments.
DU	Distributed Unit.
e2e	end-to-end.
eCPRI	evolved CPRI.

EM	Element Management.
eMBB	enhanced Mobile Broad-Band.
eNB	Evolved NodeB.
EPC	Evolved Packet Core.
EPCaaS	EPC as a Service.
ETSI	European Telecommunications Standards Institute.
EUTRAN	Evolved Universal Terrestrial Radio Access Network.
FCFS	First-come, First-served.
FDD	Frequency Division Duplex.
FFT	Fast Fourier Transform.
FG	Forwarding Graph.
FIFO	First In Firs Out.
FS	Functional Split.
G	Greedy.
gNB	next-Generation Node B.
GPP	General Purpose Processor.
GPU	Graphics Processing Unit.
HARQ	Hybrid Automatic Repeat-Request.
ICIC	Inter-Cell Interference Coordination.
IFFT	Inverse Fast Fourier Transform.
ILP	Integer Linear Programming.
IMS	IP Multimedia Subsystem.
IoT	Internet of Things.
IPC	Inter process communication.
IQ	In-Phase Quadrature.
ISA	Instruction Set Architecture.
ISI	Inter-symbol interference.
ISP	Internet Service Provider.
IT	Information Technology.
JVM	Java Virtual Machine.
KPI	Key Performance Indicator.
LLR	Log-Likelihood Ratio.
LTE	Long Term Evolution.
MANO	Management and Orchestration.
MCO	Main Central Office.
MCS	Modulation and Coding Scheme.
MEC	Multi-access Edge Computing.
MILP	Mixed Integer Linear Programming.
MIQCP	Mixed Integer Quadratically Constrained Program.
MNO	Mobile Network Operator.
NAT	Network Address Translation.

NBS	Nash Bargaining Solution.
NF	Network Function.
NFV	Network Function Virtualization.
NFVI	NFV Infrastructure.
NFVO	NFV Orchestrator.
OAI	Open Air Interface.
OFDM	Orthogonal Frequency Division Multiplexing.
ONAP	Open Networking Automation Platform.
OS	Operating System.
OTT	over-the-top.
PAPR	Peak-to-average power ratio.
PDCP	Packet Data Convergence Protocol.
PM	Physical Machine.
PMD	Polarization Mode Dispersion.
PoP	Point of Presence.
PRB	Physical Resource Blocks.
QCI	QoS Channel Indicator.
QoE	Quality of Experience.
QoS	Quality of Service.
RAM	Random Access Memory.
RAN	Radio Access Network.
RANaaS	RAN as a Service.
RAT	Radio Access Technologie.
RB	Resource Block.
RE	Resource Element.
REST	Representational State Transfer.
RLC	Radio Link Control.
RoE	Radio over Ethernet.
RR	Round Robin.
RRH	Radio Remote Head.
RSC	Recursive Systematic Convolutional.
RTT	Round Trip Time.
SC-FDMA	Single Carrier Frequency Division Multiple Access.
SINR	Signal-to Interference Noise Ratio.
SISO	Single Input Single Output.
SLA	Service Level Agreement.
SNR	Signal Noise Ratio.
SOA	Service Oriented Architecture.
TB	Transport Block.
TBS	Transport Block Size.
TDD	Time Division Duplex.
TTI	Transmission Time Interval.
UE	User Equipment.

UL	uplink.
URLLC	Ultra-Reliable Low-Latency Communications.
vBBU	virtualized BBU.
vEPC	virtual Evolved Packet Core.
VIM	Virtualised Infrastructure Manager.
VM	Virtual Machine.
VMM	Virtual Machine Monitor.
VNF	Virtualized Network Function.
VNF FG	VNF Forwarding Graph.
VNFC	VNF Component.
VNFM	VNF Manager.
VNO	Virtual mobile Network Operator.
VRRM	Virtual Radio Resource Management.
WFQ	Weighted Fair Queuing.

Introduction

Problem Statement

The great diversity of network services and applications are pushing network operators to constantly update and upgrade their infrastructures. These evolutions entail continual growth of CAPEX and OPEX. The required fast deployment is not compliant with the current network infrastructure based on proprietary hardware. Network infrastructures require today complex operations and manual configurations making them difficult to update and maintain. Inter-working between diverse network environments is also an important weakness of current infrastructures. These issues cannot be easily solved without renewing network architectures. Network Function Virtualization (NFV) [2] offers a new way of designing, deploying and managing networking services.

NFV precisely consists of replacing network functions, which were so far running on dedicated and proprietary hardware, with open software applications running on shared Commercial off-the-shelf (COTS) servers in cloud platforms. NFV promises great economic savings and time-to-market acceleration, as well as more flexible and accurate management of resources. However, deploying on-demand networks and upgrading services on the fly requires the adoption of modern IT solutions including adapted development patterns and optimized software models.

The overarching principle of virtualization is to host network functions on one or more virtual units (Virtual Machines (VMs) or containers). Virtualized Network Functions (VNFs) are deployed on top of a virtualized infrastructure, which may span over more than one physical location and even over a cloud computing infrastructure. Ideally, VNFs should be located where they are the most efficient in terms of performance and cost. VNFs can be placed in data centers, network nodes or even in end-user devices depending on the required performance (notably latency) and resources (bandwidth, storage and computing). The cloudification and commoditization of network functions, however, brings new challenges, especially when virtualizing wireless access networks.

The performance evaluation of VNFs is essential for designing software-based networks as performance models determine engineering rules for the deployment of future virtualized infrastructures. The challenge is to develop models for designing infrastructures that combine both software defined networks and cloud platforms.

Main objectives

The main goal of the present PhD thesis is to highlight performance models for virtualized network functions in order to develop engineering rules for their deployment. This includes:

- Determining a driving use case.
- Identifying Key Performance Indicators (KPIs) that reflect the behavior of virtualized network functions.
- Defining KPI-based slicing rules in order to decompose a global network function into elementary components.
- Evaluating the performance of virtualized network functions taking into account the scheduling strategy of internal resources.
- Validating theoretical models by means of a use case implementation.

Thesis outline and document structure

This thesis begins with a thorough survey of the state of the art presented in Chapter 1. This survey considers both academic and industrial studies in terms of Network Function Virtualization (NFV) and resource allocation. We present in Chapter 1, a deep analysis of various virtualization technologies while considering both container- (e.g., Docker) and VM-based systems (e.g., OpenStack). Modern Information Technology (IT) paradigms, namely microservices-based applications, are particularly introduced for gathering relevant implementation guidelines. Beyond the study of the virtualization concept and the formal definition of VNFs, we discus in this chapter, the problem of VNFs placement which basically determines where and how the building blocks of a virtualized network service are instantiated. Finally, we examine various resource allocation and scheduling strategies in the aim of abstracting fair sharing tenets.

In Chapter 2, we study the VNF modeling as a chain of components (or microservices) in the perspective of deriving execution principles on multi-core platforms. VNFs are executed on the top of the virtualization layer while sharing the available computing resources. A global scheduler is in charge of allocating the capacity of servers. In order to evaluate the VNF performance in terms of latency, we concretely address the evaluation of various scheduling strategies. At the end of this chapter, we pay special attention to the 'Processor Sharing' discipline; we concretely model the simultaneous execution of VNFs' components by a queuing system with batch arrivals.

We define the virtualization and the cloudification of Radio Access Network functions (Cloud-RAN) of mobile networks as the driving use-case of this study. Cloud-RAN aims centralizing the base-band processing of radio signals while keeping distributed antennas. To achieve the performance evaluation of Cloud-RAN systems, and notably numerical experiments, we use Open Air Interface (OAI), an open-source solution which implements the RAN functionality in software.

A thorough study of Cloud-RAN systems is carried out in Chapter 3. Since radio access networks must meet strict latency constraints, we particularly analyze the runtime of software-based RAN functions in the aim of identifying bottlenecks. We notably propose in this chapter, a parallel processing model to reduce latency when executing virtualized RAN functions in the cloud. Finally, to gather a first approach for Cloud-RAN sizing (i.e., determining the required computing capacity to host the RAN functionality of a given number of base stations) we carry out a worst-case analysis.

The problem of dimensioning Cloud-RAN infrastructures is concretely addressed in Chapter 4. We introduce a batch queuing model, namely the $M^{[X]}/M/C$ multi-service system, to assess the needed processing capacity in a data center while meeting the RAN latency requirements. We specially study two scheduling strategies of parallel runnable RAN jobs.

As a proof of concept, we have implemented the proposed models in an OAI-based test-bed platform. Performance results as well as implementation principles are presented in Chapter 5. Conclusions and main contributions are finally summarized in Chapter 6.

Chapter 1

Network Function Virtualization: State of the Art

Contents

1.1 Virtualization background 1
1.2 Microservices as an IT paradigm 4
1.3 Virtualized Network Functions 6
1.4 VNFs placement 9
1.5 Resource Sharing 14

The emergence of virtualization technology plays a crucial role in the evolution of telecommunications network architectures, notably by enabling the virtualization of network functions. This is clearly a groundbreaking evolution in the design of future networks and IT infrastructures, which can eventually be completely merged.

The convergence of IT and Telecom involves fundamental transformations in the way that telcos conceive, produce and operate their services. Virtualization incites network operators to redefine their business models by taking inspiration from modern IT solutions such as virtualized and microcroservices-based applications. These emerging concepts are being adopted by over-the-top (OTT) players such as Amazon or Google.

The separation of software from hardware gives rise to resource sharing mechanisms where centralized hardware can carry out diverse network functions according to network traffic and customer needs. Following a formal study of emerging IT paradigms as well as of the framework of NFV, we address in this chapter both the VNF's placement problem and the relevance of resource pooling.

1.1 Virtualization background

In a strict sense, virtualization is the process of creating a virtual version of a physical machine. This concept is specially relevant in cloud computing markets where data centers offer the possibility of reserving computing and storage resources to individual or business customers.

The virtualization technology brings multiple benefits, the major advantage is notably the efficient utilization of resources (memory, network, computing) by means of resource sharing [1],

[1] Virtualization is not necessarily a time-sharing system, although it can be [8].

i.e., physical resources are shared among the active virtual machines. Thus, the resource pooling of physical infrastructures dynamically enables the allocation of the capacity of servers according to needs instead of dedicating resources even when they are not used.

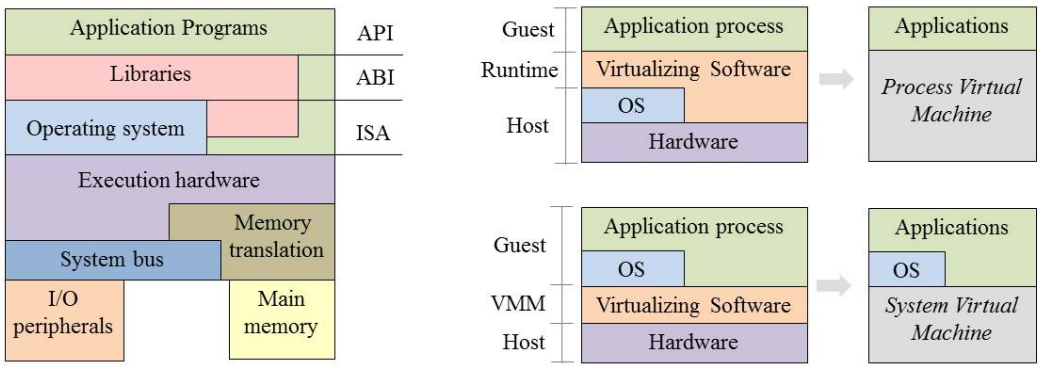
Beyond cost savings in capital and operation expenditures, virtualization enables agility, portability, and flexibility when deploying software applications. Switching over to a virtualized environment shall provide more efficient IT operations, specially when installing and maintaining software applications. Another great advantage of virtualization is the business continuity. Cloud customers can access their data and applications anywhere. In addition, virtualization makes disaster recovery much more accurate due to the facility of restoring services by means of system images whose backups are stored in the cloud.

1.1.1 Virtual Machines

Virtual Machines (VMs) were originally defined in 1974 by Popek and Goldberg [8]. They define a VM as 'an efficient isolated duplicate of a real machine'. They perform these notions by means of the so-called Virtual Machine Monitor (VMM) which provides the following key features: (*i*) 'essentially identical' environment, where any program running in a virtual machine should exhibit the same behavior as that performed on the physical machine; (*ii*) 'efficiency', where applications running in a virtual environment show at worst only minor performance reductions in terms of latency; and (*iii*) 'resource control', where the VMM is assumed to have complete control of system resources (memory, peripherals, computing, storage). The authors finally consider, a VM as the environment created by the VMM.

Virtual Machine Architecture

The VM's architecture can be defined from a process and system perspective depending on the virtualization level. While a 'process VM' simulates the programming environment for the execution of individual processes, the 'system VM' emulates the hardware and supports the execution of an operating system. See Figure 1.1 for an illustration.



(a) Physical Machine (b) Virtual Machines

Figure 1.1: Virtual and Physical Machine Architectures [1].

Process VM: From the perspective of a process, a virtual machine consists of an address space, and user-level instructions that allow the execution of a single user process. The only way that a process can interact with the I/O peripherals is through the operating system. Thus, in a 'process VM', virtualizing software translates a set of OS instructions and composes a

virtual embedded platform where Application Binary Interfaces (ABIs) enable the interaction between the process and the virtual machine. A 'process VM' exists only to support a process, it is created when the process begins and is deleted when the process terminates [1]. The most popular example of a process VM is the Java Virtual Machine (JVM).

System VM: When considering a 'system virtual machine' it supports the entire system where various processes can run simultaneously. Smith and Nair define in [1] a *system VM* as a persistent system environment that supports an Operating Systems (OSs) and various system processes. As shown in Figure 1.1, it provides the access to virtual hardware resources including I/O peripherals, storage and computing by means of a 'guest Operating System (OS)'. The VMM emulates the hardware via the Instruction Set Architecture (ISA), i.e., provides virtualized resources.

The process or system that runs on a VM is known in the literature as 'guest', while the underlying platform that supports the VM is the 'host' [1].

System Virtual machines running on a shared physical hardware need to be managed and monitored by a specialized entity, namely the Virtual Machine Monitor (VMM) (also known in the literature as 'hypervisor'). As shown in Figure 1.2, there are two kinds of hypervisors, native and hosted ones. While the first one runs directly in the bare-metal machine, the hosted hypervisor runs as an application on the top of the operating system while VMs are seen as individual processes [9].

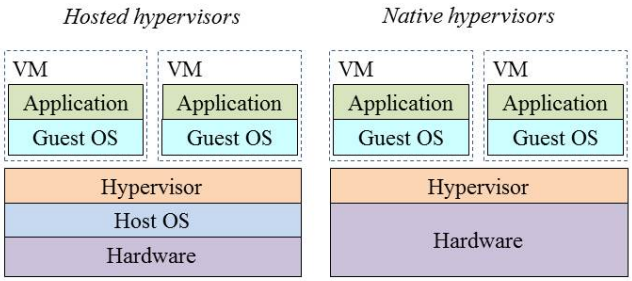


Figure 1.2: Virtual Machine architectures.

Bare-metal hypervisors: They are booted as a machine operating system and perform the complete control of physical resources. The various VMs run as system processes and maintain each of them their owner guest OS [9]. Examples of native hypervisors are Xen and VMware ESX.

Hosted hypervisors: They run on top of operating systems and are themselves a system process. As in bare-metal hypervisors, VMs maintains their owner guest OS, providing the illusion of a private hardware environment. KVM is an example of this type of VM, however, it maintains certain kernel modules that convert the host OS to a bare-metal hypervisor [9].

1.1.2 Containers

Containers were originally created in 2000*s* for security purposes, as a way of separating environments that could be shared by multiple users. In fact, the first Linux systems maintained isolated partitions or subsystems (also know as jails) [10] where the access to file systems and networking is virtualized.

Advancements in Linux 'namespaces' provided the next step for containers. In fact, user namespaces allow the isolation of Linux environments where users and groups may have privileges for certain operations inside a given container and do not outside it [11].

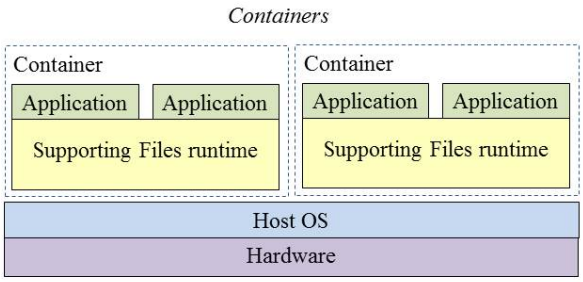


Figure 1.3: Container architecture.

Containers can then be described as Operating System (OS) level virtualisation, where the OS itself creates a Virtual Environment that provides all needed resources and files to support the processes. While VMs require a complete OS for running processes, containers require only the associated libraries needed by that process [10]. By providing an image that contains all applications' dependencies, containers are portable and consistent [11]. As shown in Figure 1.3, various containers can share a single operating system, then applications and services stay lightweight and run swiftly in parallel [11].

1.2 Microservices as an IT paradigm

Historically, there have been several trends that have incited application architects to find better ways to build systems. Software applications have been designed using objects, components, libraries, and services. Several patterns like Model View Controller, multi-tier architectures, and the widely used client-server architectures have improved the performance in execution time as well as in deployment time. Some efforts have been made in software industry, and "Microservices" were born as a way of designing software applications as suites of independently deployable services each running in its own process and communicating with lightweight mechanisms. Microservices architecture shall provide agility, robustness, and scalability to software-based network functions [12].

1.2.1 Monolithic and Microservice-based applications

While a monolithic application puts all its functionality into a single process and scales by replicating the monolith on multiple servers, a microservice-based one puts each element of functionality into a separate service and scales by distributing these services across servers, replicating as needed [12], see Figure 1.4 for an illustration.

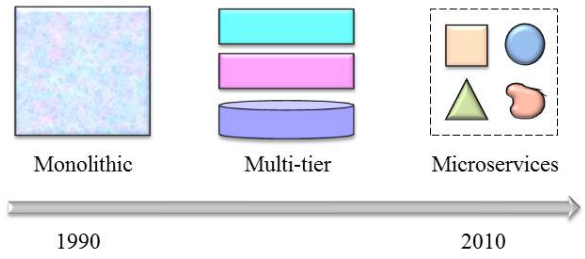


Figure 1.4: Monolithic and microservice-based applications.

Main features of microservices

- A microservice need to be treated as a product, it must be small enough to be focused on a single task. In most cases, each task represents a small business capability, e.g., a network function or sub-function [13].

- Microservices are loosely coupled and have a bounded context, i.e., they ignore everything about implementation or architecture of other microservices [13].

- Microservices can be developed in any programming language. They communicate with each other by means of neutral-language Application Programing Interfaces (APIs), e.g., Representational State Transfer (REST) protocols [13].

- Microservices maintain high availability while providing isolation and scalability. The uptime of each service contributes to the overall availability of applications [14].

1.2.2 Microservices and Component based architectures

The idea of breaking monolithic applications into components were presented several years ago. At one point objects have been the substitutes of components and then objects have come back to components again. "A component is something independently replaceable and independently upgradeable" [12]. In terms of software we can see components in two ways, libraries and services. Libraries can be defined as components that are linked into a program and called using in-memory functions while a service is a different kind of component that is running in its own process [12]. When building communication structures between the various components or services, communication approaches put significant 'smart' into the communication mechanism itself, e.g., the Enterprise Service Bus (ESB). In a microservice-based architecture, requests and responses use simple REST protocols rather than complex protocols managed by a central tool [12, 14].

1.2.3 Microservices and Service Oriented Architectures

The microservice-based architecture is a kind of Service Oriented Architecture (SOA) [15]. Although both deals with services, unlike SOA, microservices are not focused on re-usability. Microservices are built in the aim of enabling the continuous evolution of services and of making systems easier to manage.

1.2.4 Implementing microservices within containers

As presented in [14], the convergence of the evolution of both software architectures and computing infrastructures gives rise to the microservices-based paradigm. The goal of emerging trends as virtual machines or containers has always been to minimize the consumption of physical resources while enabling scalability by means of the replication of these units (VMs, or Containers). Container-based and notably unikernel-based applications abstract the host OS and implement only the needed dependencies and libraries to function, which enable granular scalability and notably the implementation of microservices. Microservices are already being proposed by cloud service providers (e.g.,'Iron.io') by means of containers-based solutions [14].

1.3 Virtualized Network Functions

1.3.1 Formal definition

The NFV concept was introduced by a group of network service providers in 2012 and has been promoted by the European Telecommunications Standards Institute (ETSI) Industry Specification Group for Network Function Virtualization [16].

NFV enables decoupling network functions from integrated hardware and moving them to virtual servers. A VNF is nothing but a software application, which can be instantiated on the fly and executed in general purpose computers. In this way, network functions are not embedded in hardware but designed as applications. These applications could run on virtualized or physical environments made of off-the-shelf hardware. The functional decoupling allows a separated evolution of hardware and software and a faster deployment of new services over an already installed physical infrastructure.

1.3.2 Architectural framework

The operation of Virtualized Network Functions (VNFs) requires a dynamic architectural framework which provides access to shared hardware and virtual resources. The NFV Framework is defined in [2] and illustrated in Figure 1.5.

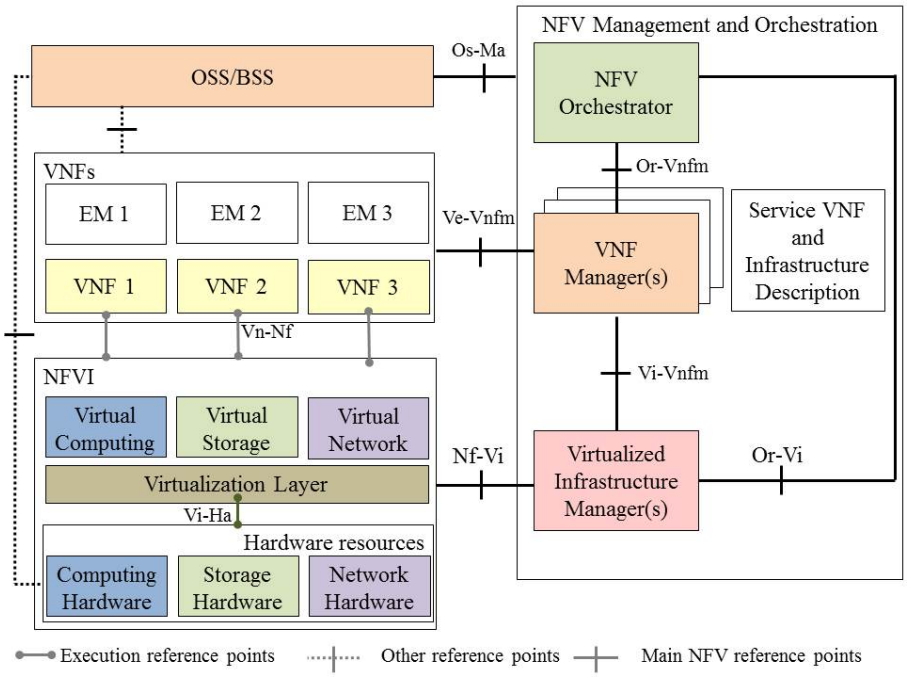


Figure 1.5: NFV architectural framework [2].

VNFs are decoupled from hardware infrastructure by the virtualization layer. Physical resources notably computing, networking and storage are shared by all software-based network functions which run in the top layer of the virtualization platform.

As presented in [2], the VNF life-cycle is managed by the Management and Orchestration (MANO) domain, which also performs the orchestration of physical and virtual resources residing in the so-called NFV Infrastructure (NFVI). To be more specific, the NFV Orchestrator (NFVO) manages the virtualisation infrastructure and carries out end-to-end (e2e) network services while multiple VNF Managers (VNFMs) perform the instantiation, update and scaling of

individual VNFs in interaction with their Element Managements (EMs). The Virtualised Infrastructure Manager (VIM) is in charge of both resource allocation procedures (e.g., hypervisors) and the deployment of virtualisation enablers (e.g., VMs onto hypervisors). Meta-data, service descriptors and deployment templates of VNFs reside in the 'Service, VNF and Infrastructure Description' entity.

1.3.3 A virtual Network Service

Service chaining

A virtualized end-to-end network service can be conceived as a Forwarding Graph (FG)' [2] of Network Function (NF) which is also known in the literature as 'Service Chaining'. The performance of the whole service is given by the aggregation of individual performances. In other words, a network service is nothing but a chain of virtual network functions interconnected between them by logical links. See Figure 1.6 for an illustration.

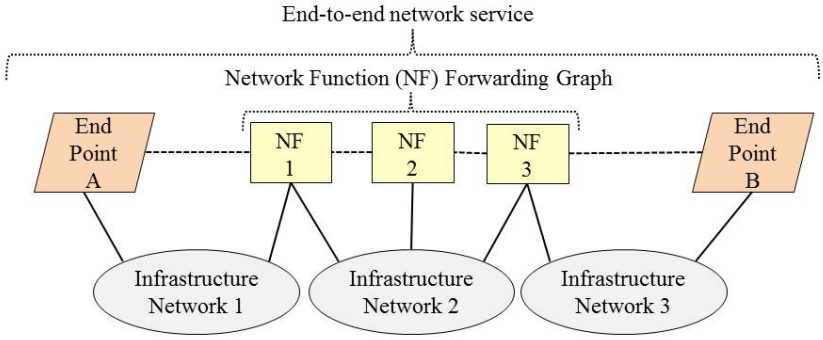


Figure 1.6: End-to-end network service [2].

The granularity of functional blocks composing the Forwarding Graph (FG) of VNFs, as well as, the geographic location of each VNF, need to be determined by network architects according to the service and performance requirements. Figure 1.7 illustrates an e2e service formed by a serial chain of VNFs ($VNF-1, VNF-2, VNF-3$) where the second one is itself decomposed in three functional blocks denoted as $VNF-2A, VNF-2B, VNF-2C$.

To be more specific, a VNF can be instantiated in a single VM or by means of multiple centralized or distributed virtual servers. A VNF can be formed by various internal components, where each single element can be executed on a VM. In the same way, a single VM can host multiple VNF's components or even various VNFs. As a consequence, Virtual Network Functions can be spread across several network sites.

In this context, VNFs composing the end-to-end service can be distributed along different physical locations (namely, Point of Presences (PoPs) of network operators). The ensemble of PoPs constitutes then the NFVI, i.e., 'the totality of hardware and software components which build up the environment in which VNFs are deployed, managed and executed' [2].

An example of service chaining

The service chaining of the 5G Mobile Core Network (namely, Evolved Packet Core (EPC) in 4G) is a good candidate for virtualization. As shown in Figure 1.8, several network functions such as user authentication, user authorization, context management, session management, among others, can be executed as a chain of components in general purpose servers, i.e., in a data center. Other examples of service chaining are the path of functional blocks of a web service (firewalls,

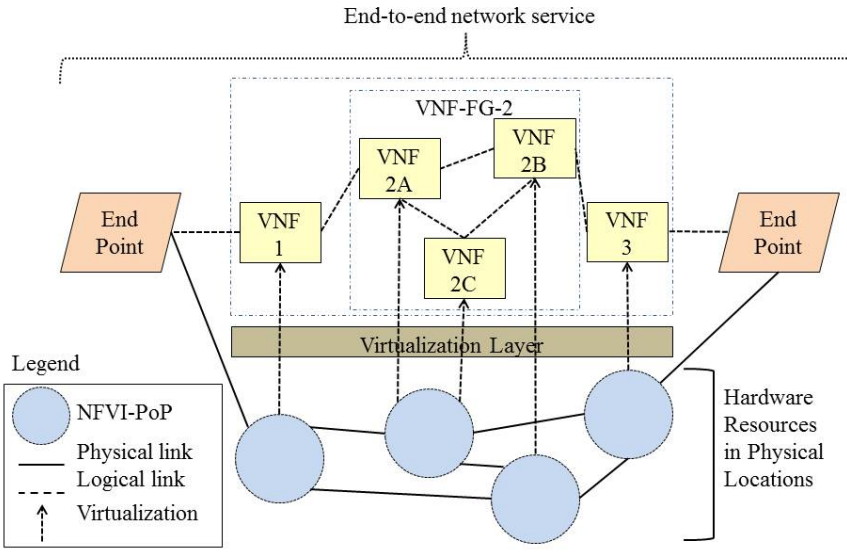


Figure 1.7: Forwarding graph of VNFs [2].

Network Address Translation (NAT) servers, and load balancers), the existing network functions within Content Delivery Networks (CDNs), IP Multimedia Subsystems (IMSs), or even the chain of base-band processing in Radio Access Networks (RANs).

1.3.4 Implementing VNFs as microservices

From a modeling point of view, a Microservice can be typically mapped to a mono-VNF or even to a single VNF Component (VNFC), which can be hosted either in a dedicated light-weighted virtualization environment e.g., a container, or in a shared virtualized system where physical resources are common with other microservices.

The first approach (dedicated) does not have any significant impact on the NFV architectural framework, while the second one (shared) requires separating the life-cycle management of VNF instances from the life-cycle of the virtualization environment. In both cases, the mono-VNF nature of microservices might reduce or eliminate the role of the VNFM. The major advantage of microservices is that they make possible the promises of NFV notably in terms of agility, scalability and robustness.

A network service as a chain of microservices

An end-to-end network service can be seen as a chain of microservices. Network functions and sub-functions can be either fully distributed into a cloud-based computing system or centralized in a node-based infrastructure. Potvin et al. propose in [3], a distributed software architecture enabling the deployment of a single software version of a microservice on multiple heterogeneous cloud platforms. Thus, microservices can be distributed and combined without location restrictions on VMs in order to efficiently use the available resources.

This is an advantage compared to a Node-based deployment where functionality of microservices is bound to specific physical resources (dedicated hardware or VM). The authors employ as driving use-case the service chaining of an IP Multimedia Subsystem (IMS), which is shown in Figure 1.9. The first microservice involved in a service setup scenario is the SIP Handler, which implements the session initiation functions of the call session control function. It uses the Node selector service for determining the VNF placement, i.e., where the call session microservice

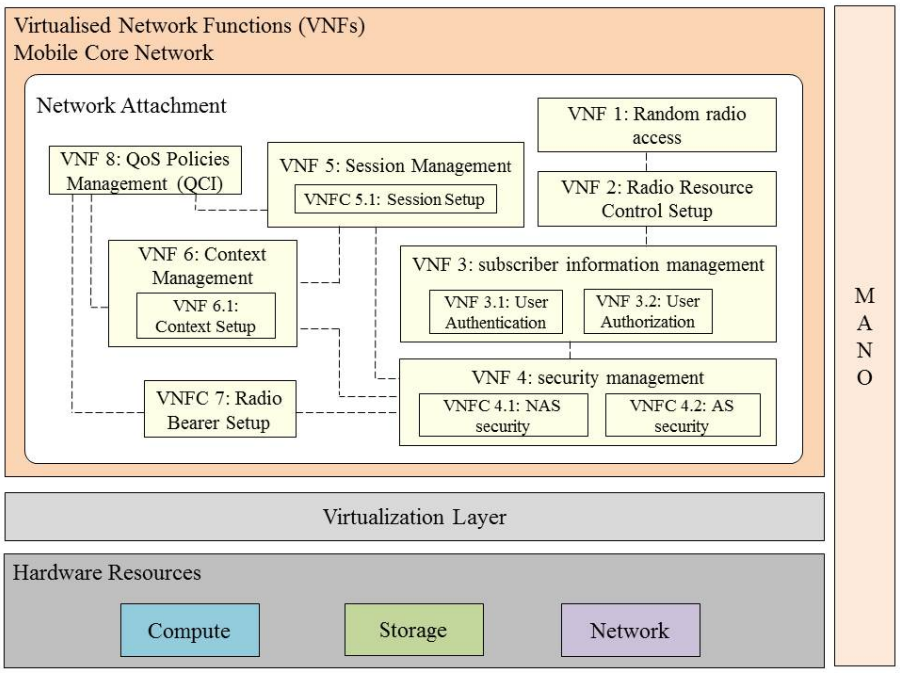


Figure 1.8: An example of service chaining: Mobile Core Network.

must be instantiated.

The microservice denoted by 'C' performs the functionally of a call session control function. It builds the appropriate service chain to provide the requested service. The C unit is instantiated on-demand and is terminated when the service is completed; during a call, it remains active until the SIP bye message is acknowledged.

The microservice 'H' is used for getting the subscriber profile. It is then responsible for querying the HSS database. The microservice 'Diameter handler (Diah)' is an interface that implements the diameter protocol towards the HSS. The main function of the microservice 'A' (anchor point controller) is to negotiate the media codec for assuring the exchange with the media processor unit. Telephony Server 'T' provides telephony related features to the Subscriber like ad-hoc conferences. Finally, the microservice 'M' provides point-to-point connectivity for performing calls in both directions.

The above use-case highlights key factors to take into account when virtualizing network services such as: *(i)* the resource allocation mechanism, notably when performing node-based systems due to the static assignation of computing and storage resources; *(ii)* scheduling algorithm when performing distributed cloud-based approaches. Performance degradation can be experimented when runnable processes wait for accessing resources. Performance results when comparing node-based and cloud-based approach evidence the relevance of resource sharing for saving resources.

1.4 VNFs placement

The placement problem or more precisely the resource allocation problem has been widely studied in the literature in the framework of cloud computing. These studies consider either 'static' or 'dynamic' allocation of memory, storage and computing capacities in the aim of hosting software applications in data centers while optimizing resource utilization and guaranteeing the proper operation of services.

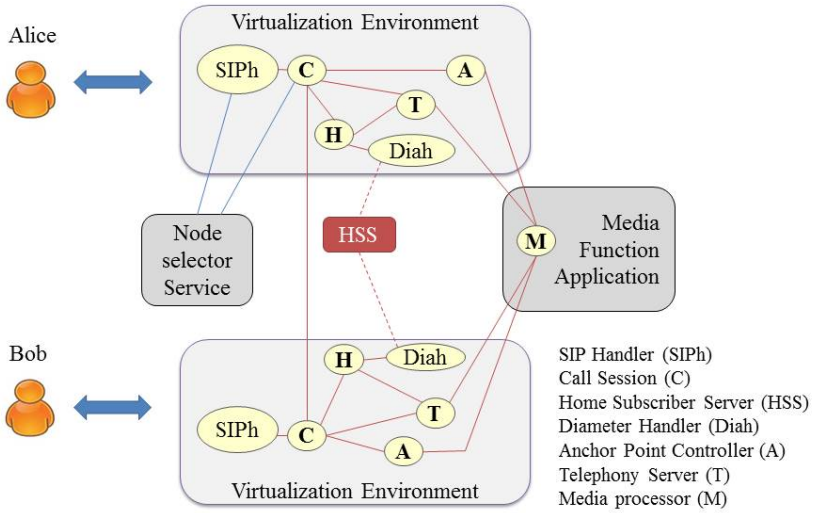


Figure 1.9: Microservices involved in a end-to-end IMS service [3].

The various VNFs composing an end-to-end service can eventually be placed at different geographic locations. Beyond computing and memory resources, the problem of VNFs placement requires to consider the capacity (bandwidth) of links connecting data centers with each other.

As shown in Figure 1.10, the key elements when assigning VNFs to data centers are:

- Network topology of data centers. For instance, a realistic topology may consider the Point of Presences (PoPs) of network operators which are generally distributed within the aggregation and core network segments. See for instance the three-level hierarchy proposed in [17];
- Availability of resources: storage, memory, computing, networking;
- Performance requirements of the end-to-end network service notably in terms of latency;
- Service chaining, i.e., when considering a virtual network service as a chain of components (e.g., VNFs).

When placing VNFs, the major challenge is to consider all the above-mentioned key elements. However, most contributions in this respect only consider certain elements together. We classify placement strategies as follows:

- By the nature of the placement in 'static' and 'dynamic'. While static solutions know in advance the set of requirements (VNFs), dynamic ones consider allocating resources on the fly at each request arrival.
- By the optimization goal when seeking load balancing among data centers, cost savings (e.g. when minimizing the number of used data-centers), guaranteeing service requirements (notably in terms of latency), or even end-to-end latency reduction (e.g., when allowing over-provisioning of resources).
- By the number of resources in mono-resource and multi-resource.
- By the number of sites in mono-site (a single geographic location with multiple servers) and multi-site (distributed data centers).
- By the type of request in mono-VNF and multi-VNF. This latter considers a request being composed of a chain of VNFs performing an end-to-end network service.

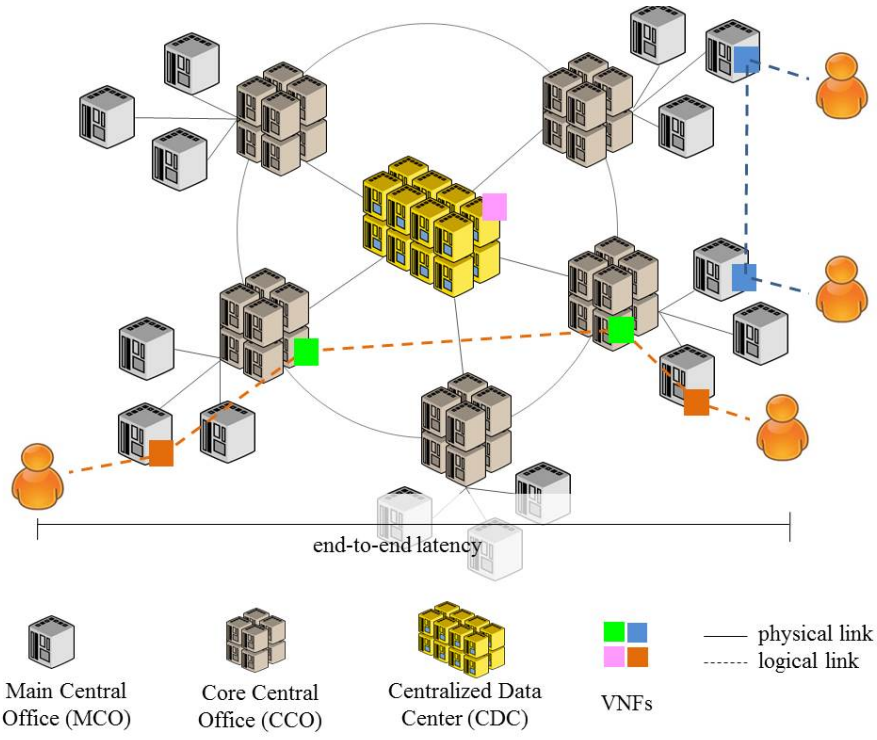


Figure 1.10: Key elements in VNFs placement.

1.4.1 VNFs placement as an optimization problem

Recent research efforts have addressed the VNFs placement as an optimization problem which looks for load balancing among data centers while minimizing the resource utilization (bandwidth, computing, memory, storage) and guaranteeing the performance requirements of network services (e.g., latency). Placement algorithms basically determine where and how the building blocks of a virtualized network service are instantiated.

The drawback of optimization approaches is that they require knowing the workload of requests 'a-priori' to provide the optimal solution, i.e., for mapping VNFs to hosting data centers. Nevertheless, when considering network services the workload varies according to traffic fluctuations, then placement and resource allocation need to be adapted on the fly.

To react quickly to changes in demands, optimization-based approaches search to minimize the convergence time of algorithms. For instance, in [18], the authors propose an optimization algorithm which gives a solution within 16 seconds for small service provider scenarios; however, the algorithm runtime linearly increases when augmenting the number of requests. The authors notably consider a NFV burst scenario where dedicated hardware becomes fully utilized and the spillover is handled by means of public cloud resources. The proposed model, called VNF-P for VNF placement, is implemented as an Integer Linear Programming (ILP).

1.4.2 Multi-site placement by heuristic approaches

In order to cope with large infrastructures, most placement models propose heuristic procedures for efficiently reaching near-optimal solutions. See for instance [19–22].

Unlike [18], the authors in [19, 21] consider the end-to-end delay of network services. In [19] resource allocation is performed by preventing over or under provisioning of resources and notably by guaranteeing end-to-end latency requirements. The authors model the VNFs placement problem as an ILP and propose a heuristic procedure for delivering results in a timely manner.

The problem is decomposed into three phases (i) determining the required number of VNF's instances (e.g., firewalls) to meet the service demand and placing them in the infrastructure (namely, PoP), (ii) assignment of service flows, i.e., VNF's instances are assigned to service flows and (iii) chaining VNFs, i.e., creating paths that interconnect previously placed network functions. This latter takes into account two crucial factors, namely end-to-end path latency and distinct processing delays added by different virtual network functions. The heuristic approach is evaluated while using realistic infrastructure topologies (namely, Internet Service Provider (ISP) environments of up to 50 PoPs), and small service chaining scenarios of up to 3 blocks. Results show that the ILP model leads to a reduction of up to 25% of the end-to-end latency notwithstanding 4% of resources over-provisioning.

A similar approach is presented in [23]. Here, the optimization criterion is based on CAPEX and OPEX cost. The model determines the optimal number of VNFs and where to place them. The algorithm knows a-priori the network topology, the set of middle-box specifications and the traffic requests.

1.4.3 Multi-provider scenarios

Dietrich et al. propose in [24], an approach for multi-provider network service embedding, where the placement of network functions takes into account the limited geographic footprint of providers. The main goal is to satisfy both customer and provider needs while improving the Quality of Experience (QoE) and maximizing revenues, respectively. The authors introduce a service model that simplifies the estimation of computing and bandwidth requests in service chains. They propose an orchestrator, called Nestor, which performs efficient VNFs placement on the basis of topology abstraction for obfuscating confidential information of network functions providers. In a first time, the orchestrator assigns network functions to data centers operated by multiple providers. Then, the assignment of network sub-functions to servers is carried out on the basis resource availability and network topology. The authors employ ILP for mapping network function to multi-provider data centers and heuristic algorithms for allocating servers to network sub-functions.

1.4.4 Multi-objective solutions

Addis et al. in [25] provide a multi-objective placement model which aims at minimizing the utilization of links and the total virtualization cost at the NFVI layer. The authors formulate the model as a Mixed Integer Linear Programming (MILP) which requires 15 minutes on average, to find an optimal solution. The algorithm is evaluated in underload conditions, on a realistic network topology which considers access, aggregation, and core networks. The capacity of links is limited and takes different values at each network level. The system takes into account two type of VNFs: tunneling and firewall. The end-to-end latency of the virtual network service and the runtime of VNFs are also considered. Results show that for strict end-to-end latency requirements the utilization of links is increased. On the contrary, when minimizing the link utilization, 'tunneling VNFs' (which increase the bitrate) are placed at the aggregation level and 'firewall VNFs' at the access and core levels.

1.4.5 Dynamic placement solutions

Mono-site resource allocation

Despite the VNFs placement problem is widely studied in the literature, few works consider the *dynamic* allocation of cloud resources (network, computing, storage), i.e., assigning request on the fly. A dynamic approach is presented in [26]. This work introduces a high-level orchestrator

for managing the resource allocation of Poisson-based request arrivals. Virtual nodes and virtual links are created and destroyed depending on traffic conditions for achieving adaptive resource utilization.

The placement decision takes into account the system load by means of a monitoring entity. Performance results are carried out in a mono-site test-bed of 4 hosts while evaluating three placement strategies (namely, optimization goals): *(i)* load balancing when choosing the least used host, *(ii)* energy savings while grouping requests in order to allocate them in a single host, and *(iii)* improving QoS while choosing the least busy host. Results notably demonstrate that *placement solutions strictly depend on optimization goals, also referred to as 'placement engine'*. This claim is consistent with results presented in [27], where the authors perform a trade-off between different optimization objectives, as well as with the multi-objective strategies introduced in [28] where the authors propose a specific algorithm for each optimization goal (namely, minimizing the number of Physical Machine (PM), load balancing) and even for each optimization scenario (namely, static or dynamic) by means of a three-dimensional vector approach ⟨CPU,RAM,I/O⟩. Unlike [26], [27] uses a Mixed Integer Quadratically Constrained Program (MIQCP) model for mapping VNFs to data-centers.

Multi-site adaptive placement

A dynamic adaptive placement of VNFs is presented in [17]. The placement strategy is notably based on both the state of data centers (which consider the load in terms of computing and memory) and the latency requirements of VNFs. The proposed strategy notably favors the placement of critical VNFs in terms of latency near to end-users. Offloaded VNFs are installed higher in the network. The offloading decision is taken on the basis of a 'target' threshold which is automatically adjusted according to the arrival rate of the various types of requests. An arriving request is accommodated in a data center when the average of occupied resources does not exceeds the 'target'. The target threshold is dynamically adapted by a classical hysteresis principle. When the average load of a given data center exceeds the maximal threshold, the target is reduced to favor deflections. Similarly, when the average load is under the minimal threshold, the target is increased to reduce deflections.

The model considers distributed data centers within a three-level hierarchy: Main Central Offices (MCOs), Core Central Offices (CCOs) and Centralized Data Centers (CDCs). The capacity of data-centers varies at each level. The authors evaluate the proposed strategy while considering two types of requests which arrive according to a Poisson process: *(i)* data plane functions which must be placed only in MCO due to strict latency constraints, *(ii)* control plane functions and Multi-access Edge Computing (MEC) applications which are delay tolerant and may be located anywhere. Despite service chaining is not considered, the authors evaluate the proposed solution against mechanism implemented in OpenStack and Open Networking Automation Platform (ONAP) while notably considering the blocking rate of VNFs (e.g., when data centers are not able to host more VNFs). Results show that the proposed algorithm improves the performance in terms of acceptance of VNFs. Another proposition of placement when using OpenStack is presented in [21].

Anticipated offloading of resource-specialized requests

Thompson et al. in [29] present a cooperative scheme in the framework of multi-resource cloud computing as an extension of their previous works which consider single resource systems (namely, Guillemin and Thompson [30] and Fricker et al. [31]). The authors specially consider resource-specialized request (e.g. a memory-specialized request demanding 1 cores and 64 GB RAM) where the worst scenario of asymmetry might generate waste of resources. To

mitigate this negative effect, the authors propose a placement strategy where jobs are forwarded to another data center for alleviating the local charge of the depleted resource. For this purpose, the algorithm uses local thresholds (one per resource) for deciding the offloading of requests (i.e., jobs demanding the scare resource) when the threshold is surpassed.

The system considers two data centers, each equipped with a limited capacity of GB Random Access Memory (RAM) and Central Processing Unit (CPU) (cores), and two kinds of requests (e.g., VNFs) demanding *(i)* a big chunk of GB RAM and only 1 CPU, (ii) 1 GB RAM and large number of cores. Requests of both types arrive at each data center at different rates. All required resources must be available upon VMs' instantiation. If required resources are not available, the arriving job (e.g., a VNF) is either forwarded to another data center or rejected. The resources are released after service completion. Results significantly improve the performance of both data centers. As in [17], chaining is not considered, however, the model may be extended to three types of resources (computing, memory, bandwidth).

1.5 Resource Sharing

Modern cloud computing business models are based on resource sharing and statistical multiplexing. Cloud computing infrastructures are able to perform overcommitment allocation of physical resources since service requirements are not all active at the same time. As a matter of fact, it is commonly observed that current cloud computing infrastructures are underused, i.e, storage and computing resources are reserved for specific applications even when they are not used [4].

1.5.1 Fairness as the basis of resource allocation

Fairness is an important performance criterion in all resource allocation strategies. The problem of fair resource allocation and scheduling strategies has been widely study in the literature. In networking, the so-called 'Max-min fairness' algorithms are the basis of 'best effort' services; for instance, when allocating the throughput to the various connections in Asynchronous transfer mode (ATM)-based networks [2] or in wireless communications by means of time-slot based schedulers [5]. Fairness concepts are also employed in certain congestion control strategies of TCP protocols for sharing the bottleneck capacity.

Resource allocation problems usually deal with various constraints, for instance priorities or deadlines. These constraints enforce the notion of fairness by enabling the allocation of resources according 'needs' [33]. See for instance the widely known priority management in ATM switching nodes [34].

At least the following two main allocation proprieties must be respected for achieving fairness:

- Sharing Principle: When considering a system with n users, each of them should not be able to get more resources than $\frac{1}{n}$ of all available resources [4].

- Pareto efficiency: It should not be possible to increase the satisfaction of a user without decreasing the satisfaction of at least another user [4].

A quantitative measure of 'fairness' called 'Index Fairness' was proposed by Jain in [35]. This metric, also known as Jain's index, has been widely used in the literature to evaluate any resource sharing or allocation problem. The index ranges from 0 to 1, where for instance, 0.15 means that a given algorithm is unfair to 85% of users.

[2] Gravey et al. introduces in [32] a resource allocation strategy for worst case traffic in ATM networks.

1.5.2 Mono-resource allocation

Max-min fairness

The max-min fairness algorithm was originally defined for the allocation of a single resource; nevertheless, it has been widely studied and extended to multi-resource allocation [36]. This algorithm relies on the principle that the smaller (minimum) demand should get the maximum possible amount of resources, i.e., the max-min algorithm prioritizes the small requests. A formal description of the max-min procedure is given in [37] as follows:

- Requests are sorted in ascending order, $x_1 \leq x_2 \leq x_3 ... \leq x_n$ where n is the number of requests (users).
- The server capacity C is divided by the number of requests n.
- C/n resources are allocated to each request.
- When a request receives a resource share larger than its demand, the unused amount of resources is equally shared among the remaining requests (users).
- The process ends when each request (user) gets no more than it asks for.

At the end, equal amount of resources are allocated to unsatisfied requests (big users), e.g., when sharing 16 cores among 4 users requiring 3, 4, 5 and 10 cores, the max-min algorithm allocates respectively 3, 4, 4.5 and 4.5 cores.

Weighted max-min fairness

The max-min share algorithm has been extended for considering heterogeneous users with different right to get resources. The procedure simply normalizes the request by a weight. Then, request with unsatisfied demands gets resources in proportion to their weights [37].

A weighted max-min fair allocation is commonly used in networks for dealing with different kinds of users or more specifically to provide Quality of Service (QoS)-based treatment. Marbach introduces in [38] a price-based priority service in which users are charged according to the priority of their traffic. It is shown that this priority-based service leads to a weighted max-min fair behavior when the network supports a continuum of priorities.

Round Robin

The Round Robin criterion [39] is widely employed in networking and computing systems. This algorithm which is coupled to the max-min criterion, allocates one unit of resource (namely, slot) to each user in a circular order. When a slot is allocated to a user that is not ready to use it, then, that same slot is offered to the next user. In each pass, a user can utilize only one slot. The main advantage of Round Robin is its simplicity, however, the drawback is a lack of flexibility. Round Robin is a preemptive (i.e., jobs are interrupted when the allocated slot is expired) and a starvation-free (any user is perpetually denied of getting service) algorithm. A window flow control is commonly used to prevent excessive queues (notably in network nodes, e.g., when a large enough window size is used throughout the network, the throughput rates are close to the ideal max-min fair rates) [40, 41].

Time-shared systems

As presented by Kleinrock in [42], the main goals of time shared systems are both enabling rapid service for small jobs and allowing users to be unaware of the presence of any other users. Each

user feels as if the entire capacity is allocated to him. To be more specific, *time shared systems, in the ideal case and at any time, the fraction of the total capacity offered to any user will be just the inverse of the number of users currently requesting service* [42]. These kinds of algorithms do not consider switching time [3] (also referred to as swap-time), hence, they provide results for 'ideal' systems [42].

A *Processor Shared System* (also known as 'egalitarian processor sharing') considers a Round Robin criterion in which the allocated time-slot is 'infinitesimal' (namely, quantum). Hence, users receive a quantum of service infinitely often, when the total required service time is received, users leave the system. This definition is identical to a model in which 'each user receives continuous processing at a rate C/k operations per second when there are a total of k users in the system and C is the processor's capacity in operations per second'. This claim is exposed by Kleinrock in [42]. In processor shared systems there is no queuing, all arriving users enter service immediately, but their service rate is proportional to the number of users in the system. Implementation guidelines are given in [43]. A packet-based version of 'processor sharing' is known as 'Fair Queuing', see for instance [44, 45].

Proportional fair

Presented in [46], the proportional fair algorithm can be considered as a specification of Weighted Fair Queuing (WFQ). It pays special attention to resource utilization efficiency, i.e., allocates resources to requests (users) in proportion to how efficiently they use them [36, 47]. The formal definition of proportionally fair allocation, which is given by 1.5.1, maximizes the sum of the logarithms of the utilities experienced by the users, i.e., more resources have to be allocated to those users whose utility increases quickly.

$$\max \sum_{i=1}^{c} \log U_i(A_i) \tag{1.5.1}$$

where U_i is the utility of the i-th user (request), A the proportionally fair allocation of the i-th user, and c the number of users.

Note that, the log-function is used for achieving a slow behavior, i.e., for avoiding neglecting users with low utilities [36]. An alternative definition of a proportionally fair allocation is given by Poullie et al. in [36] as follows: *a proportionally fair allocation A is that for any other allocation A' the sum of proportional changes to the consumers' satisfaction is not positive.*

1.5.3 Multi-resource allocation

Despite fairness-based allocation strategies have been widely studied in the literature, the focus has so far been specially on single-resource environments.

Slot-based policies

When considering multi-resource scenarios and heterogeneous service requests most solutions allocate the various resources, separately. See for instance [48] where the authors address the problem of scheduling heterogeneous data- and CPU- intensive jobs. In a first time, the allocation strategy is based on the CPU workload, and then, on the data requirement.

An interesting analysis of slot-based allocation solutions such as those presented in [48, 49] is performed in [4]. Due that slots are fixed fractions of resources, they do not represent the task demands, as a consequence, resource allocation is not efficient. The authors in [4] concretely perform the evaluation of a cluster of 2000 nodes serving 'Facebook' demands during a given

[3] The cost for removing a job and placing the next job on the processor.

period of time (namely, one month). Results are illustrated in Figure 1.11[4]. It is shown that at least 40% of CPU demands underutilize computing resources (assigned slots) while at least 50% overutilize them. In the case of memory, at least 90% of demands use only a half or allocated memory slots.

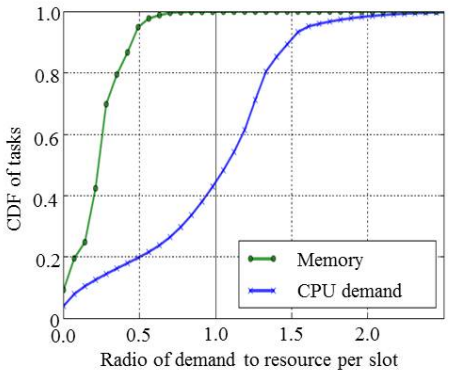


Figure 1.11: CDF of demand-to-slot ratio in slot-based allocations [4].

Dominant Resource Fairness

Ghodsi et al. propose in [4] the fair allocation of various resources types by means of a dominant resource principle. The presented algorithm, so-called Dominant Resource Fairness (DRF), is a generalization of max-min fairness to multiple resource types. The authors consider a computational model with n demand types (users) and m resources. Each user runs individual tasks where each of them is characterized by a demand vector, which specifies the number of required resources, e.g., ⟨3 CPUs, 4 GB RAM⟩. Demands are heterogeneous even those belonging to the same user.

In order to improve resource efficiency, the authors develop a fair allocation policy based on the dominant resource which corresponds to the user's 'dominant share'. The authors define the 'dominant share' as the maximum value among all 'shares' of a user. To be more specific, a 'share' is the ratio between the task resource demand and the total capacity of that resource, e.g., when considering a system with 12 CPUs and 21 GB RAM and a user's task demanding 1 CPU and 3 GB RAM, the user's dominant resource is memory since $\frac{1}{12} < \frac{3}{21}$.

The Dominant Resource Fairness (DRF) scheduling algorithm [4] tracks the total resources allocated to each user as well as the user's dominant share. At each step, the algorithm selects among the runnable tasks the user with the lowest dominant share. If there are enough resources for satisfying that user's task, it is launched. An example of DRF allocating resources is given in 1.1. The system considers 9 CPUs and 18 GB RAM to two users A and B running tasks that require ⟨1 CPU, 4 GB RAM⟩ and ⟨3 CPU, 1 GB RAM ⟩, respectively.

The total DRF allocation is given by the solution of an optimization problem, where the vectors ⟨x CPU, $4x$ GB RAM⟩ and ⟨$3y$ CPU, y GB RAM⟩ are the requirements of users A and B, respectively. Hence, the system intends maximizing allocations $max(x, y)$, while having $x + 3y \leq 9$ (CPU), $4x + y \leq 18$ (RAM) and equalizing dominant shares of users A and B, i.e., $4x/18 = 3y/9$. Then, users A and B respectively gets ⟨3 CPU, 12 GB RAM⟩ and B ⟨6 CPU, 2 GB RAM⟩ .

Performance results show that DRF leads to better throughput and fairness than the slot-

[4]A demand-to-slot ratio of 0.5 represents a job that gets a half of its requirement.

Table 1.1: Example of Dominant Resource Fairness (DRF) resource allocation.

Scheduled user	User A shares	User A dominant	User B shares	User B dominant	Next user	Total CPU	Total RAM
B	$\langle 0, 0 \rangle$	0	$\langle 3/9, 1/18 \rangle$	1/3	A	3/9	1/18
A	$\langle 1/9, 4/18 \rangle$	2/9	$\langle 3/9, 1/18 \rangle$	1/3	A	4/9	5/18
A	$\langle 2/9, 8/18 \rangle$	4/9	$\langle 3/9, 1/18 \rangle$	1/3	B	5/9	9/18
B	$\langle 2/9, 8/18 \rangle$	4/9	$\langle 6/9, 2/18 \rangle$	2/3	A	8/9	10/18
A	$\langle 3/9, 12/18 \rangle$	2/3	$\langle 6/9, 2/18 \rangle$	2/3	–	1	14/18

based policies. DRF has been adopted in 'Fair Scheduler' of Apache Hadoop[5] [50], which allows to big data applications to share resources in the cloud or in a single data-center. Hadoop is currently available in cloud solutions proposed by Amazon [51], Google [52], Oracle [53], among others.

Finally, Ghodsi et al. demonstrate that DRF satisfies the following fairness properties:

- *Sharing incentive,* while promoting users to share resources by ensuring equity among them.
- *Pareto efficiency,* as it is not possible to improve the allocation of a user without declining that of another.
- *Strategy-proofness,* as users cannot benefit from additional resources by misleading about their requirements.
- *Envy-freeness,* as users do not prefer resources of another user.

Fair allocation in heterogeneous servers

An extension of the DRF algorithm from a single server to multi-resource heterogeneous servers is presented in [54]. The algorithm, namely DRF in Heterogeneous environments (DRFH), assumes that user's tasks are divisible, nevertheless, due that this fact does not reflect reality, the authors propose using a First-Fit algorithm, i.e., selecting the first free server that fits the task requirements. In addition, for improving resource efficiency utilization, they propose a heuristic solution which chooses the 'best' (instead of 'first') server that most suitably matches task requirements. Both 'First Fit' and 'Best Fit' are evaluated by simulation while using Google cluster-usage traces of 900 users. The cluster contains 12000 servers. Each arriving job is divided into a number of tasks, each demanding CPU and RAM resources. Performance results show that the proposed solution improves resource utilization compared to slot-based schedulers while enabling the reduction of the execution time of jobs. It is demonstrated that DRFH fulfills almost all fairness properties of DRF [4] (namely, Pareto efficiency, strategy-proofness, and envy-freeness). A multi-Resource Fair Queuing system for packets processing is proposed in [55].

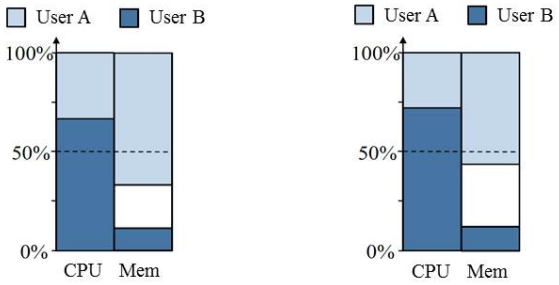
Asset Fairness

The principle of 'asset' fairness presented in [4] considers different resources (e.g., RAM, CPU) having the same value, i.e., neither resource is more important than another as each contributes for accomplishing a job (e.g., 2% of CPU is equal to 2% of RAM). Hence, the algorithm uses an

[5]Hadoop was originally designed for computer clusters built from commodity hardware.

aggregated value of resources for allocating them. A weight or worth can also be considered to each resource when combining them, e.g., the monetary cost \$2 per CPU and \$1 per GB RAM.

Ghodsi et al. [4] consider the asset fairness allocation as an optimization problem, where the vectors $\langle x$ CPU, $4x$ GB RAM $\rangle$ and $\langle 3y$ CPU, y GB RAM $\rangle$ are the requirements of users A and B, respectively. The system seeks to maximize allocations $max(x, y)$, while having $x + 3y \leq 9$ (CPU), $4x + y \leq 18$ (RAM) and to equalize total users' spends (CPU+RAM), i.e., $6x = 7y$. Then, users A and B respectively obtains (2.5 CPU, 10.1 GB RAM) and B (6.5 CPU, 2.2 GB RAM). This solution does not accomplish the 'sharing incentive' property. Figure 1.12 shows a comparison of 'asset fairness' and DRF algorithms.



(a) DRF (b) Asset Fairness

Figure 1.12: An example of 'DRF' and 'Asset fairness' allocations [4].

1.5.4 General Fairness Criterion

A unified formulation of fair resource assignment so-called *α-fairness* is introduced in [56] and extended in [5] for controlling the trade-off between efficiency and fairness. The authors use as starting point the Nash Bargaining Solutions (NBSs) [57] which are frequently used for fair allocation of achievable utilities of players in cooperative game theory.

The *General fairness criterion* (1.5.2) presented by Touati et al. [56], utilizes a parameter α for defining the degree of fairness. The α-fairness criterion concretely considers fair allocation as a function of utilities (rather than of the rate). It is given by,

$$\max_{x} \frac{1}{1-\alpha} \sum_{i=1}^{n} f_i(x_i)^{1-\alpha}, \alpha \geq 0, \alpha \neq 1 \tag{1.5.2}$$

where n is the number of users and f the utility function of the shared resource x.
According to [5, 56], this allocation corresponds to:

- the globally optimized allocation when $\alpha \to 0$,
- the proportional fair assignment when $\alpha \to 1$,
- the generalized harmonic mean fairness when $\alpha \to 2$,
- the generalized max-min allocation (most fair allocation) when $\alpha \to$ inf.

Generalized α-fair resource allocation

An extension of the above described *General fairness criterion* is presented by Altman et al. in [5]. In the framework of wireless networks and particularly downlink cellular networks, the

authors study the problem of 'fair power allocation' among the various users (say, n) requiring transmission in a base station. The throughput and the Signal Noise Ratio (SNR) are viewed as utilities of the power assignment. Thus, they define fairness as an utility function f of the resource x that is allocated, (i.e., the element x_i corresponds to the power level assigned to the i-th user) and introduce a weight π_i related to the resource of the i-th user.

Considering the generalized α-fairness utility function given by:

$$v(x) = \frac{1}{1-\alpha} \sum_{i=1}^{n} \pi_i (f_i(x_i))^{1-\alpha}, for\ \alpha \neq 1, \tag{1.5.3}$$

where $x_i \geq 0$ for $i \in [1, n]$, $\pi_i \geq 0$ for $i \in [1, n]$ and $\sum_{i=1}^{n} \pi_i x_i = \overline{x}$, where $\overline{x} > 0$; the authors concretely propose modifying (1.5.3) to deal with an objective function continuous in α as follows:

$$v(x) := \frac{1}{1-\alpha} \sum_{i=1}^{n} \pi_i ((f_i(x_i))^{1-\alpha} - 1), for\ \alpha \neq 1, \tag{1.5.4}$$

Hence, the resource allocator aims sharing 'fairly' some function of the resource x.

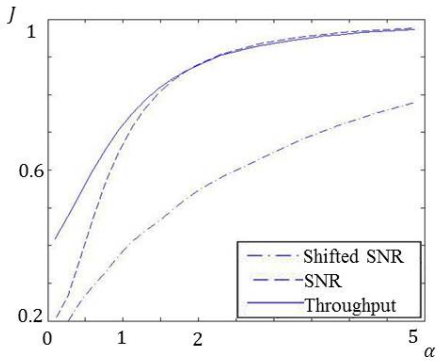


Figure 1.13: Jain's fairness index as a function of α [5].

For instance, in the case of power allocation problem, the authors consider sharing fairly the utility of a throughput instead of sharing fairly the available power (i.e., instead of sharing fairly x, sharing fairly $f(x)$). Despite f is linear, the sum $\sum_{n=1}^{N} f_n(x_n)$ is no more constant.

The authors precisely consider three utility functions $f_i(x)$: *(i)* $log(1+\frac{h_i x}{N_i^0})$, α-fair assignment of throughput; *(ii)* $\frac{h_i x}{N_i^0}$, α-fair assignment of SNR; *(iii)* $1 + \frac{h_i x}{N_i^0}$, assignment of shifted SNR for evaluating both SNR and throughput maximization and the max-min fairness; where h_i is the fading coefficient for the sub-carrier i and N_i^0 is the level of the background noise in the sub-carrier i.

Results show that in all three cases the fairness improves monotonously as α goes to infinity. The relation among different α-fair allocations for the three utility functions is given in Table 1.2 [5]. Similarly, for each α-fair allocation the authors calculate Jain's fairness ($J = \frac{(\sum_{i=1}^{n} SNR_i)^2}{n(\sum_{i=1}^{n} SNR_i^2)}$) index with respect to SNRs, see Figure 1.13 for an illustration.

Table 1.2: Generalized α-fair resource allocation in wireless networks: Relation among different α- fair allocation for three utility functions.

Utility function	$\alpha \to 0$	$\alpha \to 1$	$\alpha = 2$	$\alpha \to$ inf
Throughput	Shannon Capacity is maximized	Throughputs are assigned according to the proportional fair paradigm	Delay is minimized	Max-min fair assignment of SNR
SNR	SNR is maximized	SNRs are assigned proportionally fair	–	
Shifted SNR		Throughput is maximized	–	

Chapter 2

VNF modeling

Contents

2.1 General Assumptions 23
2.2 Use cases 24
2.3 Model description 26
2.4 Scheduling algorithms 29
2.5 Performance Analysis 30
2.6 A queuing model based on concurrent processing 34

In the framework of network function virtualization, we consider in this chapter the execution of Virtualized Network Functions (VNFs) in data centers whose computing capacities are limited. We assume that each VNF is composed of sub-functions to be executed on general purpose hardware, each sub-function requiring a random amount of processing time. Because of limited processing capacity, we investigate the relevance of resource pooling where available cores in a data center are shared by active VNFs. We study by simulation various algorithms for scheduling sub-functions composing active VNFs (namely Greedy, Round Robin and Dedicated Core algorithms). We additionally introduce an execution deadline criterion, which means that VNFs can renege if their sojourn time in the system exceeds by a certain factor their service time. This feature is especially relevant when considering the processing of real-time VNFs. Simulations show that sub-functions chaining is critical with regard to performance. When sub-functions have to be executed in series, the simple Dedicated Core algorithm is the most efficient. When sub-functions can be executed in parallel, Greedy or Round Robin algorithms offer similar performance and outperform the Dedicated Core algorithm. Enabling as much as possible parallelism and avoiding chaining when designing a VNF are fundamental principles to gain from the available computing resources. In order to achieve full parallelism while avoid waiting times, we propose in the last section a queuing model for executing virtualized sub-functions under a processor sharing discipline. Main contributions presented in this chapter are in [6, 58].

2.1 General Assumptions

An end-to-end network service (e.g., video streaming, mobile voice) can be represented as a forwarding graph of network functions, which is commonly referred to as a service chaining [2]

(see Section 1.3 for more details). Today's hardware-based approaches make the implementation of services extremely complex and time-consuming [59]. In a virtualized environment, a VNF Forwarding Graph (VNF FG) runs on the top of the virtualization layer and can be managed more efficiently [2]. For instance, to update a network service, it is enough to add a new VNF on a virtual machine; in the same way, to scale the network, it is only necessary to instantiate the underlying VNF on the computing facility.

Furthermore, each VNF consists of a number of software components, referred to in the following as sub-functions. The flexibility brought by virtualization should be exploited to realize modular and parallel sub-functions. We assume that cores of a computing facility are assigned for executing sub-functions according to a specific scheduling algorithm. This algorithm plays a key role in the system performance, especially when there are real-time constraints or deadlines for the execution of a particular VNF.

We pay special attention to the execution of VNFs in data centers with limited capacity in terms of computing. We more precisely investigate the feasibility of resource pooling. We consider a computing facility composed of a number of cores, which can be dynamically allocated for the execution of sub-functions of an active VNF.

2.2 Use cases

NFV approach enables the deployment of a large number of VNFs, which can be executed on cloud-computing as well as the fog computing environments. While large centralized data centers are commonly operated by the cloud framework, fog computing enables data center dissemination on the edge of the network infrastructure. The advantage of small data centers disseminated at the network edge is that they are close to end-users and ensure lower latency and better customer experience. The key difference between fog and cloud computing is in the limitation of computing capacity [60].

2.2.1 Virtualizing packet core network functions

Functions of mobile core network (e.g., those of the EPC of 4G mobile or future 5G networks) are good candidates for virtualization in the form of virtual Evolved Packet Core (vEPC), Cloud-EPC (C-EPC) or EPC as a Service (EPCaaS) [61, 62]. As shown in Figure 2.1 several functions can be handled by software, such as session setup, user authentication and access authorization, currently handled by the Mobility Management Entity (MME). This is also the case of packet filters, policy control and charging, supported until now by P-Gateway as well as mobility functions (e.g., in case of handover between eNodeBs or between LTE and other 3GPP access) held by S-Gateway.

Each VNF can be executed on Virtual Machines (VM) or containers. In this way, operators can instantiate each EPC component in order to cope with the incoming traffic demands. On the other hand, it is also possible to optimize resources during non-peak hours.

Moreover, a virtual EPC could be instantiated for a specific need, e.g., a company which is willing to operate its own private mobile network for its employees. The same applies for other traditional network functions, e.g., IP Multimedia Subsystem (IMS) for private Voice over IP (VoIP) networks.

2.2.2 Virtualizing wireless network functions

In spite of the promises of virtualization, several challenges should be dealt with in future implementations, particularly because some network functions have strict requirements in terms

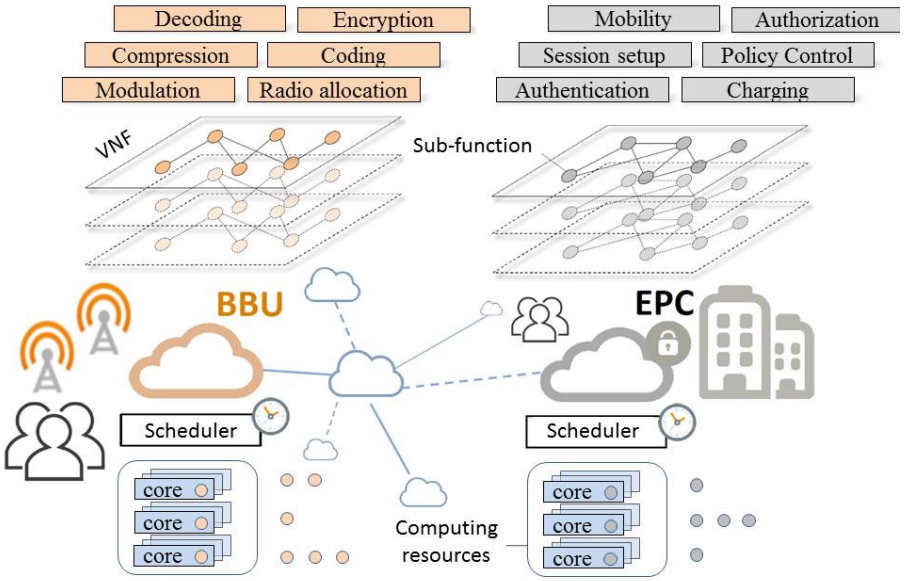


Figure 2.1: Network Function Virtualization, use-cases.

of real-time processing. This is notably the case of Radio Access Network virtualization for short vRAN. For instance, in 4G mobile networks, the Time Division Duplex (TDD) process requires a response ACK/NACK (i.e., ACK to proceed to the transmission of a new frame, or NACK to attempt a retransmission) to the User Equipment (UE) or eNodeB within 3ms after a frame is arrived, where 2ms are available for reception process and 1ms for transmission process [63]. This operation named Hybrid Automatic Repeat-Request (HARQ) is performed at the MAC level, after demodulation and decoding are done. If successful decoding is not possible, the packet retransmission is scheduled. Nevertheless, this fact degrades the customer experience, because it reduces the achievable UE peak rate [64].

Considering the realization of vRAN, traditional eNodeBs with integrated radio and base-band processing are replaced with shared processing and distributed radio elements, i.e., Radio Remote Head (RRH). So, virtual Base Band Unit (BBU) sub-functions are located at a centralized site, named below vBBU, where a pool of computing resources could be dynamically allocated based on traffic conditions. In addition, BBU virtualization can dramatically improve the global network performance; since several radio elements are handled by a single BBU pool, inter-cell coordination facilitates cooperative multi-point processing and massive MIMO implementation while avoiding interference [65].

In this way, the entire BBU functionality [66, 67] including RAN L1, L2, and L3 protocol layers could be represented as a suite of virtual sub-functions available for treating Physical Resource Blocks (PRB), as illustrated in Figure 2.1. Modulation and encoding process as well as data convergence process (e.g., packet compression and encryption) and radio bands allocation might be considered as virtual sub-functions of the base-band processing module. Thus, the processing load mainly depends of the Modulation and Coding Scheme (MCS), the execution time of these sub-functions is determined by the channel conditions between the eNodeB and the current UE. Depending on which value is reported by the UE, network transmits data with different MCS. If network gets high Channel Quality Indicator (CQI) from UE (i.e., good radio quality) it transmits the data with high-order modulation and a simple channel coding scheme. Likewise, for a low CQI, the network constructs the transmission block with a robust MCS. All sub-functions except modulation/demodulation and encoding/decoding might have small run time enabling the possibility of sharing computing resources [64].

Decoding function (e.g., LDPC, turbo codes) is the most complex in terms of processing load, since its number of iterations depends on the signal quality. Therefore, decoders are promising candidates for parallelization to meet real-time constraints; for instance, multiple code words may be decoded in parallel or even the decoder itself might be decomposed into multiple threads that run in parallel [64].

In view of the two above use cases (i.e., vBBU, vEPC), we see that virtualization requires an appropriate decomposition of virtual network functions in order to guarantee an efficient resource utilization in terms of computing by adequate scheduling algorithms. This fact enables resource pooling and statistical multiplexing in the execution of sub-functions on multi-core platforms, performing the same tasks with less hardware or capacity. It is set out in more detail in the following section.

2.3 Model description

2.3.1 Model settings

At present resource allocation in virtual platforms have near-static behavior; virtual machines or containers are reserved for a specific VNF (e.g., vEPC, vBBU) even though a VNF is sporadically invoked. As a consequence, efficiency in resource utilization is not achieved, since computing resources are frozen but no used. It would thus be better to perform statistical multiplexing on computing resources. More precisely, in the following, we assume that a set of cores is available to execute VNFs with dynamic resource orchestration. Cores are allocated for the execution of a VNF when that function is invoked.

As discussed in Section 2.2, in most cases, the runtime of a virtual network sub-function is deterministic. Nevertheless, other aspects discussed in [68] are intrinsic to the execution of a sub-function such as the computing architecture (e.g., GPU/CPU-based), memory access time, caching policies, disk access time, inter-processor communication, system buses and I/O buses behavior, virtualization technology[1], among others. In fact, the multiprocessing architecture plays an important role in the whole network virtualization performance. In the following, we include these issues in the holding time of cores, which becomes a random variable. This is a classical assumption in network performance modeling when various factors influence the execution of a job.

Taking into account a pool of cores which are statistically shared by several active VNFs, a significant gain in resource savings is expected. The counterpart is that the execution of a VNF might be delayed until some core is available in comparison with the case when computing resources are dedicated to a single VNF. In this work, we assume that a VNF is composed of sub-functions, each of them being executed on the multi-core platform. The goal of this work is to investigate how the underlying sub-functions should be scheduled or even conceived to improve the VNF performance.

In this context, the functional disaggregation in the virtualization process should take into account the correspondence between sub-functions, because it determines the behavior in the execution process. Then, a particular VNF could be viewed as a process flow with sub-functions either running in sequence or else being executed in simultaneous threads. The present work analyzes the behavior of scheduling algorithms for both configurations, i.e., when VNFs are executed as a chain of sub-functions in contrast with the case when sub-functions are allowed to be executed in parallel.

[1]The architecture of various virtualization technologies are given in Section 1.1.

2.3.2 Representation of a VNF

We consider a VNF as a suite of executable processes. Each of them executes a specific sub-function in such a way that the global VNF can be realized. Some processes can be started only when the output of the previous one is available. Two such processes are necessarily executed in series. On the contrary, some tasks can run in parallel even if the subsequent task can be executed only when the output of all parallel processes is available.

This leads to the representation of a VNF as depicted in Figure 2.2. The tasks t_{n-1} and t_n are executed in series but t_{n+1} is composed of m sub-tasks $t_{n+1,1}, \ldots, t_{n+1,m}$, which can be executed in parallel. Task t_{n+2} can be executed only when all the tasks $t_{n+1,j}$, $j = 1, \ldots, m$ are completed.

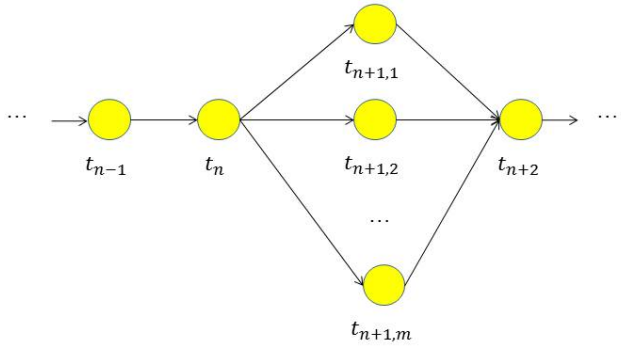


Figure 2.2: A VNF as a chain of sub-functions.

With regard to the execution of VNFs on a Graphics Processing Unit (GPU)/CPU-based server, we can assume that the computing capacity is shared among the various active VNFs. The server can be composed of a single or several cores. In the case of a multi-core platform, we assume that cores are shared among the various VNFs. The scheduler contained in the kernel of the operating system is in charge of allocating the capacities of the cores. See Figure 2.3 for an illustration.

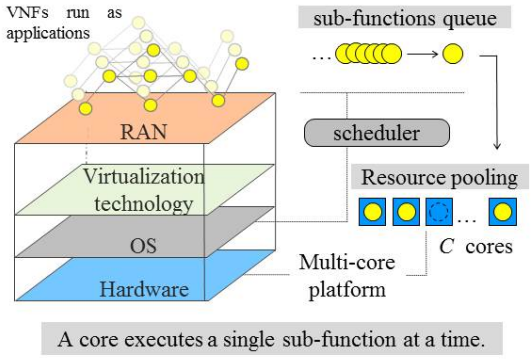


Figure 2.3: Architecture of the virtualization environment.

Each task of a VNF can be considered as a job requiring a certain amount of service but upon service completion, a job can simply leave the system or give rise to another job with another service requirement, or to several jobs each with a certain amount of service, which can give rise to another job only when all the parallel jobs are completed.

We can hence easily see that the presence of batches can have a significant impact on those

tasks which are executed in series, if the server capacity is shared among all active tasks. A VNF can momentarily receive more processing than other VNFs. Moreover, if the subsequent task of a VNF can start only when all the sub-tasks of a batch are all completed, then badly scheduling tasks can introduce significant latency in the execution of the global VNF.

2.3.3 Queuing system formulation

Let us consider a computing facility (i.e., a data center) equipped with a pool of c cores capable of executing elementary sub-functions. These sub-functions are part of a VNF, also referred to macro-function. Such a VNF is composed of k sub-functions; when a VNF is invoked the entire batch of k sub-functions has to be handled by the computing facility. The model proposed below takes k as a constant, however, it could be easily extended to the case when k is a random variable (e.g., with a geometric distribution).

VNF requests occur at the computing facility according to a Poisson process with rate λ. The execution time of a sub-function is random with an exponential distribution, with mean $1/\mu_j$ for the j-th sub-function of a VNF, where $j = 1, \ldots, k$. We assume that execution time of various sub-functions are independent, hence, the execution time of a macro-function is originally the sum of k exponential random variables. The mean execution time is

$$\mathbb{E}[s] = \sum_{j=1}^{K} \frac{1}{\mu_j}.$$

When all μ_j are equal to some $\mu > 0$, the execution time is simply an Erlang random variable with mean k/μ and variance k/μ^2, denoted, for short, by $\mathrm{Er}(k, \mu)$.

In general, this model involves a pool of cores which execute only one sub-function at a time. The holding time of a core by the jth sub-function of a VNF is exponential, with mean $1/\mu_j$. Hence, when a request occurs while all cores are busy, the function is queued, see Figure 2.4 for an illustration. The total amount of time r to treat a VNF by the computing facility is composed by the queuing delay q and the execution time of sub-functions s, so that, the system response time for the execution of a VNF is given by $r = s + q$. The queuing delay is the amount of time that the VNF spends in the system while none of its sub-functions is executed.

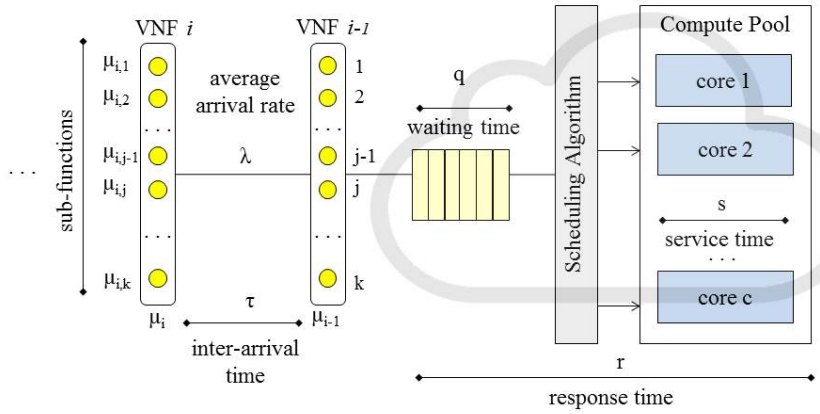


Figure 2.4: VNF modeling: Elements of the queuing system.

We assume that the waiting room of the computing facility is infinite. It is then clear that

the system is stable if and only if the load ρ defined by

$$\rho = \frac{\lambda \mathbb{E}[s]}{c}$$

is less than 1. In the following, we assume that this stability condition is always satisfied. That condition is not necessary in the case of reneging, since this phenomenon makes the system stable at the price of rejecting requests.

2.3.4 Performance Indicators

In view of the above queuing system formulation, we characterize its performance by the distribution of the sojourn time r of VNFs in the system. If r takes small values, then macro-functions (VNFs) are rapidly executed and consequently, the overall efficiency is achieved.

On the other hand, if the sojourn time is too large, then some VNFs are slowly executed and in some cases (e.g., vRAN) completing the execution of a VNF becomes useless. As presented in Section 2.2, a time budget of 3ms for BBU processing is available to allow continuous transmission per UE. This fact is equivalent to introduce the concept of reneging. This budget represents the time between the end of the uplink transmission and the start of the downlink subframe carrying the corresponding ACK/NACK [69]. More precisely, a customer reneges if its sojourn time in the system exceeds too much its service time. Reneging in multiple-servers queuing systems is discussed in [70]. The adopted reneging criterion is given below:

Definition 1 (Reneging macro-function)**.** *A VNF execution request reneges if its sojourn time $r > (1+\theta)s$ for some $\theta > 0$. This is equivalent to the condition $q > \theta s$, where q is queuing delay of the macro-function.*

In case of reneging, the VNF leaves the system and all resources used prior to its departure are wasted. Hence, the performance of the system is characterized beyond the sojourn time r by the reneging rate $\eta(\theta)$ as a function of the parameter θ.

Whether with the distribution of sojourn time or reneging, the system performance depends on the queuing delay of VNF requests, i.e., the time that a VNF spends in the system without receiving service. This queuing delay depends on how sub-functions are sorted for their execution. This is precisely the role of the scheduling algorithms described in the following section.

2.4 Scheduling algorithms

The scheduling algorithm determines which VNF and, more precisely, which sub-function gets access to a particular core of the computing facility. In other words, the scheduling algorithm selects which sub-function is executed. The scheduling strategy has a direct impact on the system response time as well as on the resources utilization efficiency. Thus, the scheduling analysis aims at identifying conditions and constraints to improve the system performance.

The scheduler selects the next sub-function to be processed among VNFs that are ready to be executed, and allocates to it a processing unit (core). Let us consider three scheduling algorithms: Dedicated Core (DC); Round Robin (RR) and Greedy (G) presented in the following subsections.

2.4.1 Allocating the entire macro-function to a Dedicated Core

This method processes all sub-functions forming a particular VNF upon the same core. The scheduler selects the VNF keeping the arrival order. In this case, the computing facility can

be described by an $M/G/c$ queue system, where the service time is the sum of independent exponential random variables. If all μ_j are the same, then an $M/\mathrm{Er}(k,\mu)/c$ queue system is obtained.

2.4.2 Allocating sub-functions by Round Robin criterion

This algorithm handles the VNFs under the RR criterion. Incoming VNFs integrate the circular cycle upon arrival, while the scheduler selects them in first-in-first-out order. Hence, sub-functions are assigned to the pool of cores in a circular order. As detailed in [71], Round Robin generally employs time-sharing, assigning to each task a time slot, and interrupting the task if it is not completed. In this work, the proposed algorithm does not force the sub-function going out of the core if it is not finished, since we assume that sub-functions are not divisible.

2.4.3 Greedy allocation of sub-functions

The Greedy (G) algorithm tries to complete the started VNFs as quickly as possible in their starting order while consuming available resources, i.e., when a VNF starts its service, the system seeks to finish it in the fastest way. Thus, VNFs which have started their service are prioritized over those which have not begun their execution. Hence, when a core becomes free, it is immediately taken by the oldest current VNF. Although there is no equity, the G algorithm remains work-conserving.

In other words, Greedy does not try to share the computing resources among the VNFs which are in the system as in the case of the RR algorithm, but it aims to complete the service of VNFs at the earliest possible way without fairness.

2.5 Performance Analysis

2.5.1 Simulation settings

The fundamental issue is to analyze the performance of the scheduling algorithms introduced in the previous section. We are interested in the sojourn time distribution of VNFs as well as in the reneging rate in case of deadline for the execution of a VNF. Keeping in mind that a VNF is formed by sub-functions, which can be executed in parallel or in series, the behavior of scheduling algorithms is analyzed for both cases.

Different scenarios have been considered in relation with the computing pool size and the number of sub-functions per VNF, taking values where $c < k$, $c > k$ and $c \gg k$. Furthermore, we have considered different load conditions, namely $\rho = 0.6$ (moderate load) and $\rho = 0.9$ (heavy load). Finally, we analyze the behavior when all sub-functions have the same mean execution time. This means that for all $j = 1, \ldots, k$, $\mu_j = \mu$ for some constant $\mu > 0$. The parameter $1/\lambda$ equal to the mean inter-arrival time of VNFs is taken as the time unit. The parameter μ is adjusted so that the load $\rho = k\lambda/(\mu c)$ is equal to a prescribed value.

The performance results analyzed below correspond to those scenarios with a heavy load and a number of cores greater than the number of VNF's sub-functions. The same configuration was applied when sub-functions belonging to a particular VNF have different mean execution time. We have also evaluated the performance when different classes of VNFs share the same multi-core platform. This latter scenario offers the possibility to support new services with the same infrastructure, having for instance the co-execution of RAN functionality with other network functions. The results obtained for these scenarios are qualitatively the same as those presented below and are not reported in this work.

2.5.2 Scheduling performance without reneging

Let us consider VNFs where their components represent a forwarding graph of sub-functions. For instance, the base-band procedure of the radio access network (namely, BBU) is a chained process whose functional blocks roughly are modulation/demodulation, encoding/decoding, radio scheduling, and RLC/MAC PDU generation/decomposition. Hence, we analyze the performance in terms of sojourn time of VNFs when those sub-functions are scheduled following the RR criterion as well as the DC algorithm [2]. Figure 2.5 illustrates the behavior of both algorithms considering a highly loaded system with parameter $\rho = 0.9$, which means that the arrival rate of VNFs is important for the computing capacity.

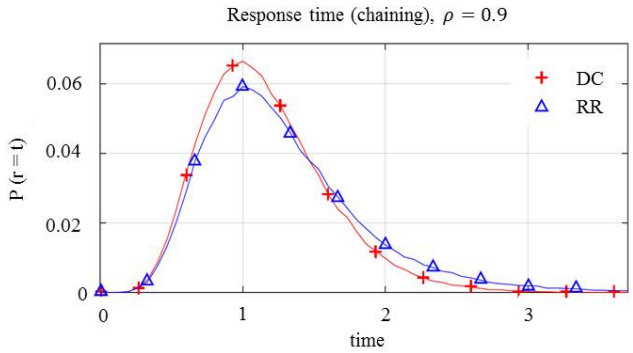


Figure 2.5: Scheduling performance considering chained sub-functions.

From Figure 2.5, it turns out that the simple DC algorithm offers slightly better performance than the RR algorithm, especially for the tail the of the sojourn time distribution. This will be confirmed by considering the case when VNFs can renege. The additional delay introduced by a higher sojourn time represents in most cases the degradation of the customer experience. For instance, when executing a vBBU, all information sent from the physical layer to the MAC layer and vice-verse has to deal with this extra time. In this case, a critical effect of a high sojourn time is the expiry of channel measurements sent from the UEs to the eNodeB, which are used for radio scheduling and other fundamental issues in the characterization of radio signals. As a consequence, the mobile network losses both energy and spectral efficiency.

Let us consider now, VNFs whose sub-functions are allowed to be executed in parallel. This means that the execution of a particular sub-function is independent of the results of the previous one. Figure 2.6 presents the behavior of three algorithms where the performance of both G and RR is notably better than the behavior obtained with the DC algorithm. Even more, G algorithm shows a slightly better performance than that of RR.

It is evident that the chaining constraint considered in the first simulation balks the advantages of RR algorithm in its attempt of fairness. In this way the simplicity embedded in DC criterion is the most appropriate for a computing pool environment, all the more as the complexity of RR does not improve the sojourn time of VNFs.

2.5.3 Scheduling performance when considering a deadline

In this subsection, we analyze the scheduling performance considering reneging (i.e., a deadline in the execution of VNFs) with parameter $\theta = 1$. As explained in Section 2.3, this factor represents the deadline present in some network functions which require real-time execution as in the case

[2]When network sub-functions are chained, the behavior of the Greedy algorithm corresponds to that of the DC scheduler.

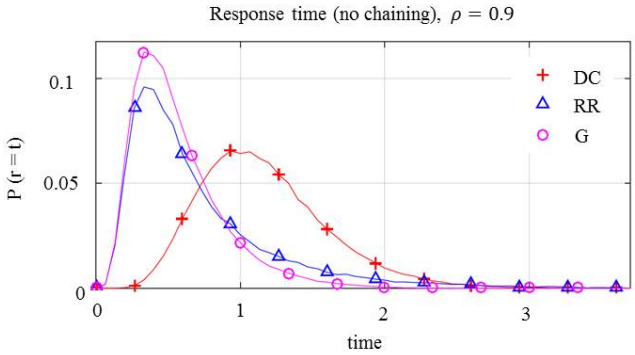


Figure 2.6: Scheduling performance considering no chained sub-functions.

of the base band processing of mobile networks. Considering the scenario where $c > k$ and applying the chaining constraint, Figure 2.7 illustrates the behavior of RR and DC algorithms. Again their performances in terms VNF's sojourn time are similar with an advantage for DC when considering the tail of sojourn time distributions.

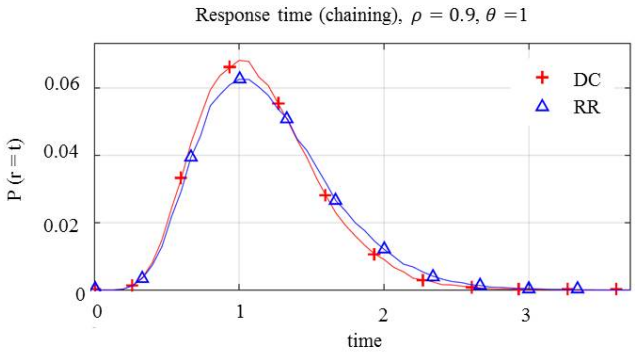


Figure 2.7: Scheduling performance considering chained sub-functions and reneging.

Table 2.1 shows reneging rates which represent the number of VNFs that have not finished their execution.

The utilization rate and waste rate show respectively the occupation of the computing platform by VNFs which have been completely processed and the occupation of cores by sub-functions belonging to VNFs which have reneged. DC offers a slightly better performance, in line with the observation made for the tail of the sojourn time distributions when there is no reneging.

Table 2.1: Scheduling performance with chained sub-functions.

	Reneging rate	Utilization rate	Waste rate
DC	1.6270	99.0821	0.6816
RR	4.5060	97.0478	2.4837

RR algorithm yields a higher reneging rate, and consequently worse utilization factor than DC algorithm. Hence, more computing resources are wasted.

Analyzing the case when sub-functions can be executed in parallel, the behavior of scheduling

algorithms turns out again more favorable for G and RR criteria. It is shown in Figure 2.8 where reneging rate was established with $\theta = 1$, it means that a VNF interrupts its service and leaves the system if its sojourn time is greater than the double of the required execution time. The same kind of behavior has been observed for smaller and greater values of θ.

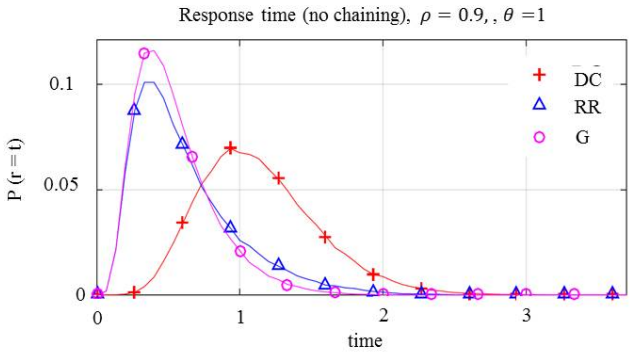


Figure 2.8: Scheduling performance considering no chained sub-functions and reneging.

As much as in terms of execution delay as in terms of reneging rate presented in Table 2.2, G has the best performance.

Table 2.2: Scheduling performance with no chained sub-functions.

	Reneging rate	Utilization rate	Waste rate
DC	1.6220	99.0478	0.6956
RR	2.6430	98.3832	1.2734
G	0.4560	99.7832	0.0625

It is evident that the behavior of DC is relegated by RR and G performance. Nevertheless, the utilization rate of DC remains interesting when compared with that resulting of RR execution. Results show that the worst algorithm in terms of resources saving is RR although it keeps a better sojourn time. Greedy becomes the most efficient and notably the most suitable when the execution of sub-functions is not limited by the chaining constraint.

2.5.4 Analysis of results

We have studied a system executing VNFs on a computing platform composed by a pool of cores; each VNF is composed of several sub-functions. We have analyzed by simulation the performance of three algorithms for scheduling the execution of sub-functions of active VNFs (namely, Round Robin, Dedicated Core and Greedy). It turns out that when sub-functions can be executed in parallel, the Greedy algorithm ensures the best performance in terms of execution delay. We have also considered the case when VNFs may renege because the sojourn time in the system exceeds some threshold related to the required amount of service. Still in this case, the Greedy algorithm offers the best performance.

In the case of chained sub-functions Greedy algorithm cannot be applied, and the performances observed with Dedicated Core and Round Robin are similar, so the complexity added by this latter is not justified.

This phenomenon has to be taken into account when designing VNFs executed on a pool of cores. In particular when decomposing a network service into components or microservices, the

best choice is to decompose a function into sub-functions which can be executed in parallel and independently of each other.

2.6 A queuing model based on concurrent processing

2.6.1 General principles of concurrent computing

Concurrent computing enables executing various jobs or tasks simultaneously for better performance. In concurrent environments, runnable jobs are executed all together by time-sharing processors. Concurrent computing is not the same that parallel computing, this latter enables the parallel execution of jobs in a strict sense, i.e., runnable jobs are executed at the *same* physical instant on separate processors. Conversely, in concurrent processing various jobs can share a single processing unit by interleaving execution steps of each process via time-sharing slices (also referred to as time-slots) [72, 73].

Then, concurrent processing relies on a processor sharing discipline (see Section 1.5 for details) where the available processing capacity C is shared between all runnable jobs. Hence, when there are n runnable jobs in the system, each of them receives service at the rate C/n. Note that jobs do not have to wait for service, their processing starts immediately upon arrival. Sharing the processing capacity at the same time-instant is an idealized assumption, however, is theoretically interesting because characterizing the sojourn time of jobs shall enable capturing important engineering rules for dimensioning the required computing capacity of infrastructures hosting VNFs.

2.6.2 Model settings

The present model specially considers the *'concurrent'* execution of jobs (sub-functions) belonging to VNFs. To be more specific, we consider VNFs (e.g., functions belonging to the mobile core network or even to the radio access network) composed of a set of sub-functions which are able to be executed in parallel. In other words, each VNF arriving to the computing center is split in parallel runnable jobs forming a batch of jobs. The functional splitting and data decomposition of VNFs are studied in Chapter 3.

As a consequence of the parallelization [3], the delivery time of the entire VNF is therefore determined by the completion time of each job. In other words, the performance of a massive parallelization essentially depends on the completion delay of jobs (microservices). In view of the random nature of the flow of requests (VNFs' jobs) in time, the probability distribution of the sojourn time of an arbitrary job (e.g., a network sub-function) quantifies its performance in stationary conditions. This distribution can then be used for dimensioning purposes in order to guarantee that, with a large probability, the service of a job is completed before some time lag.

2.6.3 Queuing formulation

To represent such a parallelized service mechanism through a simple model, we investigate the $M^{[X]}/M/1$ queuing system with Processor-Sharing discipline.

We assume that requests, hence all jobs therein, are simultaneously addressed to a single server whose computing capacity can be considered as the sum of individual capacities of processing units composing the Cloud (the issue of load balancing between distinct servers is therefore not considered here). This server can be represented by a single queue fed by the incoming flow of requests; in the present model, we assume that this flow is Poisson with constant rate λ. Any

[3] In the present section, we always refer to concurrent computing (processor sharing) even when using 'parallelization or parallel' as terminology.

incoming request simultaneously brings a 'batch' of elementary jobs for service, with the batch size (in terms of the number of jobs) denoted by B; the service time of any job pertaining to this batch is denoted by S. All random variables B (resp. S) associated with consecutive batches (resp. with jobs contained in a batch) are supposed to be mutually independent and identically distributed. In view of a fair treatment of requests by the server, we finally assume that all jobs in the queue are served according to the Processor-Sharing (PS) discipline (see Figure 2.9 for illustration).

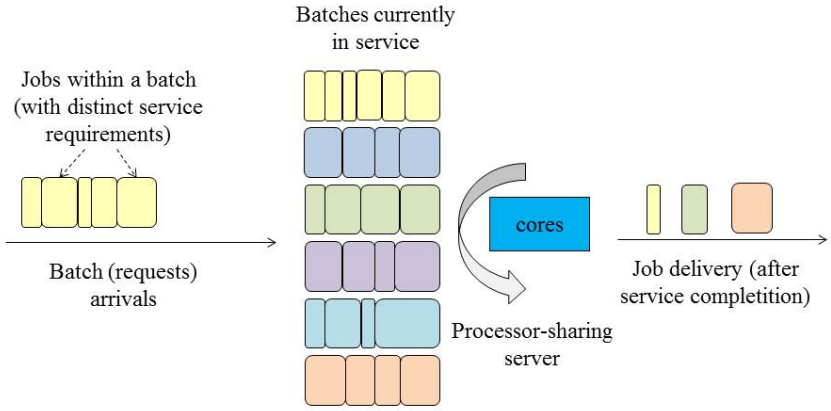


Figure 2.9: Processor-Sharing queue for the modeling of parallelized batch service [6].

Given that $\mathbb{E}(B) < +\infty$ and that the service S is exponentially distributed with parameter μ, the corresponding $M^{[X]}/M/1$ queue has a stationary regime provided that the stability condition

$$\lambda\,\mathbb{E}(B) < \mu \tag{2.6.1}$$

holds.

Let $\mathbb{P}(B = m) = q_m$, $m \geq 1$, define the distribution of the size B of any batch; it is known ([74], Vol.I, §4.5) that the number N_0 of jobs present in the queue has a stationary distribution whose generating function is given by

$$\mathbb{E}(z^{N_0}) = \frac{\mu(1-\varrho^*)(1-z)}{\mu(1-z) - \lambda z(1-\mathbb{E}(z^B))}, \quad |z| < 1, \tag{2.6.2}$$

where $\varrho^* = \lambda\mathbb{E}(B)/\mu$ is the system load; in particular, $\mathbb{P}(N_0 = 0) = 1 - \varrho^*$.

In order to evaluate the performance of virtualized network functions, we are concretely interested in the sojourn time distribution of both a single job and an entire batch which are presented by Guillemin et al. in [6], main contributions are summarized in Appendix A.

2.6.4 Job's sojourn time

When considering an $M^{[X]}/M/1$ queue with batch arrivals and PS discipline, we concretely determine the c.d.f. $\mathbf{G}_q : x \geq 0 \mapsto \mathbb{P}(W > x)$ of the sojourn time W of a job, together with its tail behavior at infinity.

While the mean value was studied in the technical literature by Kleinrock [75] [4]*, the exact distribution of the sojourn time of a job for exponential service times was so far not known.*

[4]The stationary distribution when considering $B = 1$ has been previously studied in the literature; the case of a general batch size $B \geq 1$ has been considered but for the derivation of the first moment $\mathbb{E}(W)$ only. To our knowledge, the distribution of sojourn time W for a batch size $B \geq 1$ is newly obtained in the present study.

We follow the approach of ([76], Sections 4 and 5) when applied to the queue with Random Order of Service, while accounting here for the specific properties of transform $\mathbf{G}_q^*$ established in [6], Section 4.

Distribution Function

To derive an integral formula for the distribution function of sojourn time W, first use the inverse Laplace formula to write

$$\forall\, x \geq 0, \quad \mathbb{P}(W > x) = \frac{1}{2i\pi} \int_{\Re(s)=c} \mathbf{G}_q^*(s)\, e^{xs} \mathrm{d}s \tag{2.6.3}$$

for any real $c > \sigma_q^+$.

Applying Cauchy's theorem on the closed contour of Figure 2.10 and letting the radius R tend to infinity, integral (2.6.3) on the vertical line $\Re(s) = c$ is shown to equal an integral on the finite segment $[\sigma_q^-, \sigma_q^+]$, specifically

$$\mathbb{P}(W > x) = \frac{-1}{2i\pi} \int_{\sigma_q^-}^{\sigma_q^+} \Delta\mathbf{G}_q^*(s)\, e^{xs} \mathrm{d}s \tag{2.6.4}$$

where $\Delta\mathbf{G}_q^*(s) = \mathbf{G}_q^*(s+i0) - \mathbf{G}_q^*(s-i0)$ denotes the difference of the upper limit $\mathbf{G}_q^*(s+i0) = \lim_{\varepsilon\downarrow 0} \mathbf{G}_q^*(s+i\varepsilon)$ and lower limit $\mathbf{G}_q^*(s-i0) = \lim_{\varepsilon\downarrow 0} \mathbf{G}_q^*(s-i\varepsilon)$.

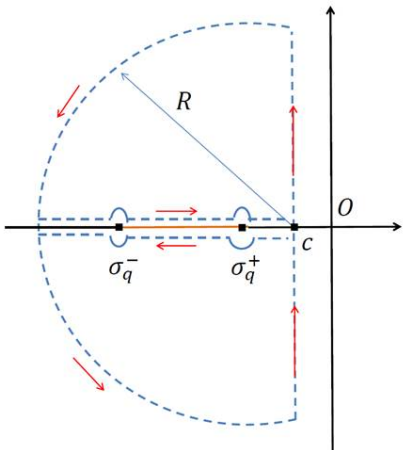


Figure 2.10: Closed integration contour avoiding the real axis [6].

Proposition 2.6.1. ***For*** $0 \leq \varrho < 1 - q$***, the complementary distribution function*** $\mathbf{G}_q : x \mapsto \mathbb{P}(W > x)$ ***of sojourn time*** W ***is given by***

$$\mathbb{P}(W > x) = (1 - \varrho - q)(1 - q) \times \int_0^\pi \frac{\sin\theta}{(1 + \varrho - q - 2\sqrt{\varrho(1-q)}\cos\theta)^2} \frac{e^{h_q(\theta,x)}}{\cosh\left(\frac{\pi}{2}\cot\theta\right)} \mathrm{d}\theta \tag{2.6.5}$$

for all $x \geq 0$***, with exponent***

$$h_q(\theta, x) = \cot\theta \left(2\Phi - \frac{\pi}{2} - \theta\right) - (1 + \varrho - q - 2\sqrt{\varrho(1-q)}\cos\theta)x$$

and $\Phi = \Phi(\theta)$ ***given by***

$$\tan\Phi = \frac{\sqrt{1-q}\sin\theta}{\sqrt{1-q}\cos\theta - \sqrt{\varrho}}, \quad \Phi \in [0,\pi], \tag{2.6.6}$$

As an application of the exact formula (2.6.5), the complementary distribution function of the sojourn time W is plotted in Figure 2.11 for several values of parameters q and $\varrho < 1-q$, the load $\varrho^* = \varrho/(1-q)$ being kept constant and set here to 0.8 for illustration. The influence of the batch size distribution (represented here by parameter q) is clearly illustrated by the fact that the larger q (or equivalently, the larger the mean batch size), the flatter the distribution of the sojourn time W.

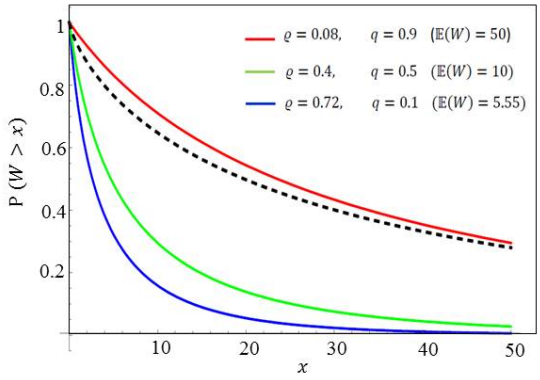


Figure 2.11: Function $x \mapsto \mathbb{P}(W > x)$ for different values of the pair (ϱ, q) with load $\varrho^* = \varrho/(1-q)$ fixed to 0.8 [6].

The characterization of the flatter tail as $\varrho + q$ takes values close to 1 is addressed in [6] Section 6.

Tail behavior at infinity

Using (2.6.5), we can now specify the tail behavior of the distribution of W.

Corollary 2.6.1. ***For large positive*** x ***and*** $0 < \varrho < 1-q$***, we have***

$$\mathbb{P}(W > x) \sim c_q(\varrho)\left(\frac{\pi}{x}\right)^{\frac{5}{6}} \exp\left[\sigma_q^+ x - 3\left(\frac{\pi}{2}\right)^{\frac{2}{3}}[\varrho(1-q)]^{\frac{1}{6}}\, x^{\frac{1}{3}}\right] \tag{2.6.7}$$

with coefficient

$$c_q(\varrho) = \frac{2^{\frac{2}{3}}}{\sqrt{3}\,[\varrho(1-q)]^{\frac{5}{12}}} \frac{(1-\varrho-q)(1-q)}{(\sigma_q^+)^2} \exp\left(\frac{\sqrt{1-q}+\sqrt{\varrho}}{\sqrt{1-q}-\sqrt{\varrho}}\right)$$

and $\sigma_q^+ < 0$ ***defined by***

$$\sigma_q^- = -(\sqrt{1-q}+\sqrt{\varrho})^2, \quad \sigma_q^+ = -(\sqrt{1-q}-\sqrt{\varrho})^2 \tag{2.6.8}$$

.

Heavy load In the case of heavy load when $\varrho \uparrow 1-q$, we note from

$$\mathbb{E}(W) = \frac{1}{1-\varrho-q}. \tag{2.6.9}$$

that the mean value of the product $(1 - q - \varrho)W$ is always equal to 1 for any value of parameter q. This motivates the following assertion.

Proposition 2.6.2. ***Let the scaled time*** $V = (1 - q - \varrho)W$.

When $\varrho \uparrow 1 - q$, ***we have*** $V \Longrightarrow V^{(0)}$ ***in distribution, the distribution function of the limit variable*** $V^{(0)}$ ***being given by***

$$\mathbb{P}(V^{(0)} > y) = 2\sqrt{y}\, K_1(2\sqrt{y}), \quad y \geq 0, \tag{2.6.10}$$

where K_1 ***denotes the second modified Bessel function of order 1.***

From the known behavior of Bessel function K_1 at infinity [77], we obtain

$$\mathbb{P}(V^{(0)} > y) \sim \sqrt{\pi} \cdot y^{\frac{1}{4}} \exp(-2\sqrt{y}) \tag{2.6.11}$$

for large y. The complementary distribution function of $V^{(0)}$ and its asymptotic at infinity are depicted in Figure 2.12.

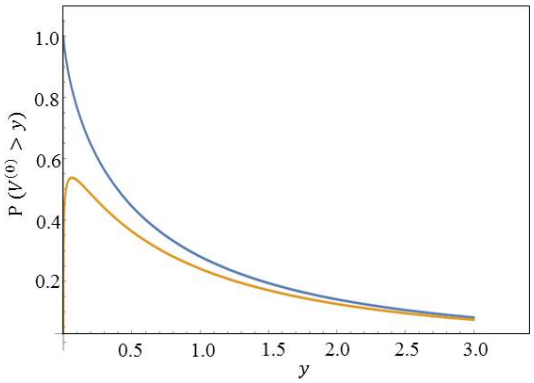


Figure 2.12: Function $y \mapsto \mathbb{P}(V^{(0)} > y)$ and its asymptotics for large y [6].

2.6.5 Batch's sojourn time

We address the evaluation of the distribution of sojourn time Ω of an entire batch in the considered $M^{[X]}/M/1$ queue with PS discipline. By definition, given a batch size $B = m$, $m \geq 1$, the duration Ω equals the maximum

$$\Omega = \max_{1 \leq k \leq m} W_k \tag{2.6.12}$$

of the sojourn times W_k, $1 \leq k \leq m$, of jobs which build up this batch. Let $\mathbf{G}_q(x) = \mathbb{P}(W > x)$ as above, together with $\mathbf{D}_q(x) = \mathbb{P}(\Omega > x)$ for $x \geq 0$.

We propose a simple approximation to function $\mathbf{D}_q$ in terms of function $\mathbf{G}_q$ whose validation is assessed by simulation.

Given a batch with size $B = m \geq 1$, consider that the m distinct sojourn times W_1, ..., W_m of the m jobs of this batch are mutually independent. From definition (2.6.12) and the latter notation, this independence scheme enables us to approximate the conditional distribution of Ω, given its batch size, as

$$\mathbb{P}(\Omega \leq x \mid B = m) \approx (1 - \mathbf{G}_q(x))^m, \qquad x \geq 0;$$

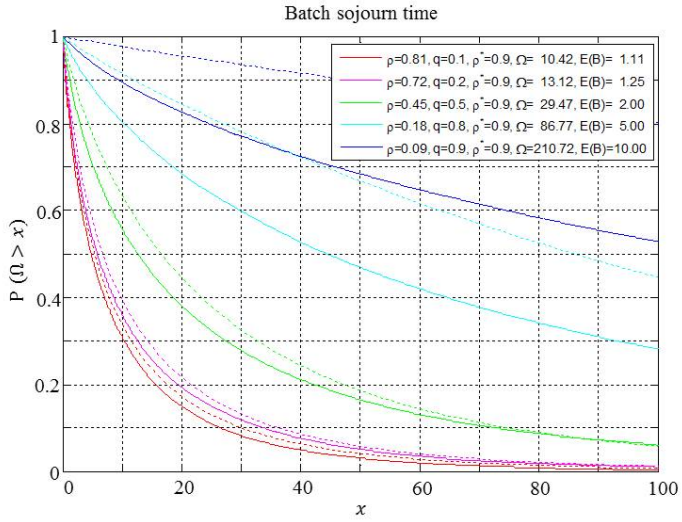


Figure 2.13: Distribution $D_q : x \mapsto \mathbb{P}(\Omega > x)$ of the batch sojourn time and its approximation [6].

on account of the geometric distribution of the batch size, the unconditional distribution of Ω is then approximated by

$$\mathbf{D}_q(x) \approx \mathfrak{A}_q(x), \quad x \geq 0, \tag{2.6.13}$$

where we set

$$\mathfrak{A}_q(x) = \frac{\mathbf{G}_q(x)}{1 - q + q\,\mathbf{G}_q(x)}.$$

Analysis in heavy load

We evaluate the distribution of the batch sojourn time Ω through a numerical simulation. This evaluation is performed for several values of the pairs (ϱ, q), keeping the system load $\varrho^* = \varrho/(1-q)$ equal to 0.9 (see Appendix A for light load analysis); in each simulation run, a number of 10^6 batch arrivals into the queue has been generated to guarantee a 99% confidence interval.Figure 2.13 shows the distribution $D_q : x \mapsto \mathbb{P}(\Omega > x)$ of the batch sojourn time (solid line) and its approximation (2.6.13) (dotted line).

We observe that (2.6.13) compares reasonably well to the exact distribution under the condition that batches have a small enough size; in fact, a statistical mixing of jobs takes place when considering small batches, thus ensuring the independent treatment of jobs within batches. On the other hand, the quality of approximation (2.6.13) decreases when increasing batch size, say, for a mean batch size $\mathbb{E}(B) \geq 5$; in fact, the independence assumption for jobs can be hardly envisaged in the presence of a small number of large batches.

Chapter 3

Cloud-RAN as an NFV use-case

Contents

3.1 System description **41**
3.2 Fronthaul analysis **44**
3.3 Runtime evaluation **51**
3.4 Worst-case study **59**

We study in this chapter the case of Cloud-RAN which aims to virtualize Radio Access Network (RAN) functions and to instantiate them in the cloud. Throughout this work, we use OAI [78], an open source solution that implements the RAN functionality in software. This chapter specially address the performance analysis of virtual RAN functions in terms of latency notably due to runtime of such functions in commodity servers. We further propose a method for improving the performance and validate it by simulation. These claims notably refer to [79–81].

3.1 System description

Current mobile networks have a distributed architecture where the base-band processing of radio signals (namely, Base Band Unit (BBU)) is located near to antennas. BBUs are implemented on proprietary hardware and are provided by a single vendor.

C-RAN (also referred to as Cloud-RAN, or Centralized-RAN) aims at centralizing the base-band processing coming from various cell-sites in a Central Office (CO) or more generally in the cloud. In other words, C-RAN dissociates antennas (RRHs) and signal processing units (BBUs). C-RAN can be seen as a BBU-pool, which handles tens or even hundreds of cells-sites (namely, Evolved NodeBs (eNBs) or next-Generation Node Bs (gNBs)). A site is typically composed of 3 sectors, each equipped with an RRH. The RRH has two RF paths for downlink (DL) and uplink (UL) radio signals, which are carried by fiber links to the BBU-pool.

C-RAN was introduced by China Mobile Research Institute in 2010 [82]. Since then, various studies and test-bed platforms have appeared in the literature. Certainly, the main challenge of this promising software-based approach is the required real-time behavior of virtual RAN functions. This problem has been largely studied and analyzed by industry [83, 84] and network operators [79, 81, 85], as well as academic researchers, notably via the development of several open-source or even proprietary solutions such as OAI [86, 87] and Amarisoft [88].

3.1.1 Architectural framework

Cloud-RAN exploits the NFV framework bringing applications closer to the radio access network. This proximity enables cost-effectiveness, scalability and flexibility due to the usability of shared COTS platforms and Coordinated Multi-point (CoMP) technologies, e.g., joint transmission, for reaching spectral efficiency and quality of user experience[1]. Cloud-RAN systems are based on open-platforms where the base-band functions can be instantiated on demand (RAN as a Service (RANaaS) [90]). In the same way, the computing resources can be dynamically allocated according to needs. Various Cloud-RAN applications are presented in Appendix B.

A Cloud-RAN system is composed of a forwarding graph of virtualized base band functions which are today deployed into a BBU. Hence, a virtualized BBU (vBBU) implements in software all network functions belonging to the three lower layers of the Evolved Universal Terrestrial Radio Access Network (EUTRAN) protocol-stack. These functions mainly concern IFFT/FFT (I/F), modulation and demodulation (M/D), encoding and decoding (CC), radio scheduling (RS), concatenation/segmentation of Radio Link Control (RLC) protocol, and encryption/decryption procedures of Packet Data Convergence Protocol (PDCP), for the downlink and uplink directions [91].

The vBBU is deployed on commodity hardware, i.e., multi-core GPU/CPU-based servers and employs a virtualization technology. This latter can be based on VMs and/or containers. In this work, we take advantage of the performance provided by containers which, unlike VMs run on a single common kernel. This gives them the benefit of being faster and more resource-efficient. This point has been studied in [92]. As shown in Figure 3.1, BBU functions can be represented as runnable-tasks (processes or jobs) which are placed in the highest layer of the Cloud-RAN system.

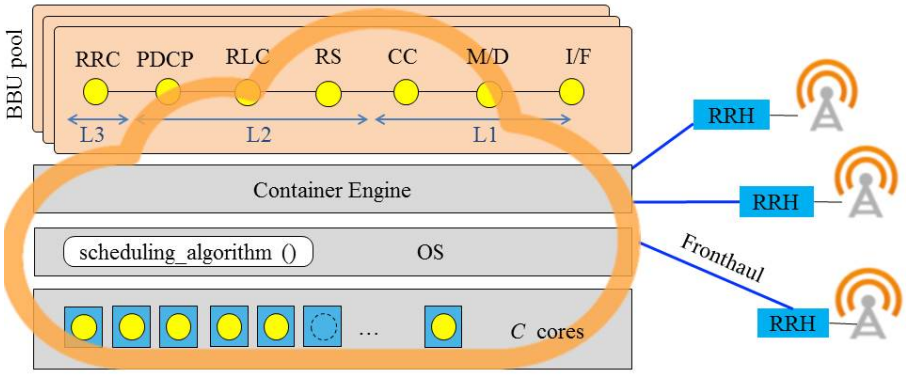


Figure 3.1: Cloud-RAN architecture.

3.1.2 Implementation guidelines

The implementation of software-based RAN functions in a data center, as depicted in Figure 3.2, calls for some software implementation principles. The main goal is to execute virtualized BBU functions sufficiently fast so as to increase the distance between RRHs and BBU functions (namely BBU-pool) and thus to improve the concentration level of BBUs in the CO for CAPEX and OPEX savings.

[1]Today's Coordinated Multi-point (CoMP) solutions typically incur large signaling between cells and are difficult to implement. In order to cope with this major disadvantage, research efforts have been devoted for dealing with non-coordinated approaches [89]

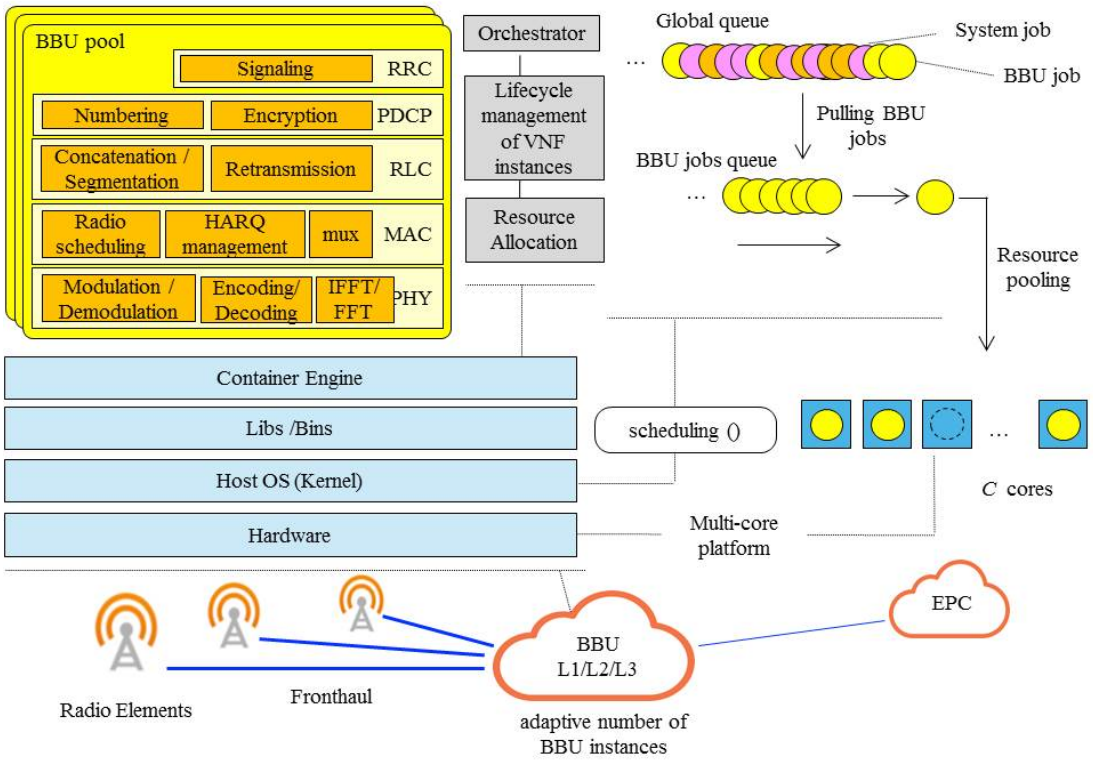


Figure 3.2: Cloud-RAN virtualization environment.

Let us consider a multi-core platform (pool of resources) which can host various eNBs. The main challenge is to guarantee the individual performance of each eNB while avoiding waste of resources. Thus, the computing platform processes the BBU functions of various radio network elements which are geographically distant.

The scheduling strategy plays a crucial role in the performance of eNBs, since it allocates the processing capacity and decides which runnable BBU-job will be executed, i.e., which processor executes a job and in what order the jobs are processed. We assume that all cores are controlled by a global scheduler. Partitioned scheduling is also possible in multi-core systems, however dedicating resources for a particular task or sub-function limits the performance of the VNF runtime [93]. Scheduling algorithms are implemented in the kernel of operating systems (OS). We assume that all BBU-jobs have the same and the highest priority in the system (OS). Thus, the scheduling policy allocates cores among the runnable BBU-threads. We use containers as virtualization technology. Virtualization solutions where the resource allocation of CPU and memory is handled by an external entity (e.g., the hypervisor in the case of VM) have lower performance than those where the resource provisioning is kept by the OS (e.g.,containers) [66].

The architecture of computing platforms is also important when evaluating the performance of VNFs, especially when considering the memory access time. Generally speaking, the time required to access instructions and data in memory is rarely negligible in general purpose computers. Commodity parallel computing platforms based on GPUs can significantly increase the performance, since they give direct access to the instruction set and parallel runnable elements. The performance analysis of computer architectures and memory access mechanisms is beyond the scope of this work. It is nevertheless important to note that recent studies have compared the LTE BBU execution on GPU and CPU based architectures [94]. Results show that GPU servers substantially increase the performance.

3.1.3 Functional splits

The Cloud-RAN architecture can support selective centralization of BBU functions. Several functional splits of the BBU have been widely considered in the literature [66,95,96] and notably by the 3GPP [97]. We can roughly classify Cloud-RAN architectures as fully and partially centralized; in this work, we focus our study in the case of full centralization which moves all base-band functions (BBUs) higher in the network. See Figure 3.3 for an illustration.

A fully centralized RAN architecture has the benefit of cost reduction due to fewer number of sites hosting BBUs. On the other hand, a partially centralized architecture distributes physical functions closer to antennas to enable advanced techniques such as massive beam-forming and at the same time centralizes the control plane to bring RAN functionality closer to applications.

Both configurations are illustrated in Figure 3.3. Note that having a fully centralized RAN enables the creation of new services, e.g., RANaaS which allows a radio networks to be deployed with a specific behavior (i.e., radio scheduling, data rate, retransmission procedures) tailored to a particular service or client (e.g., customized and private mobile networks).

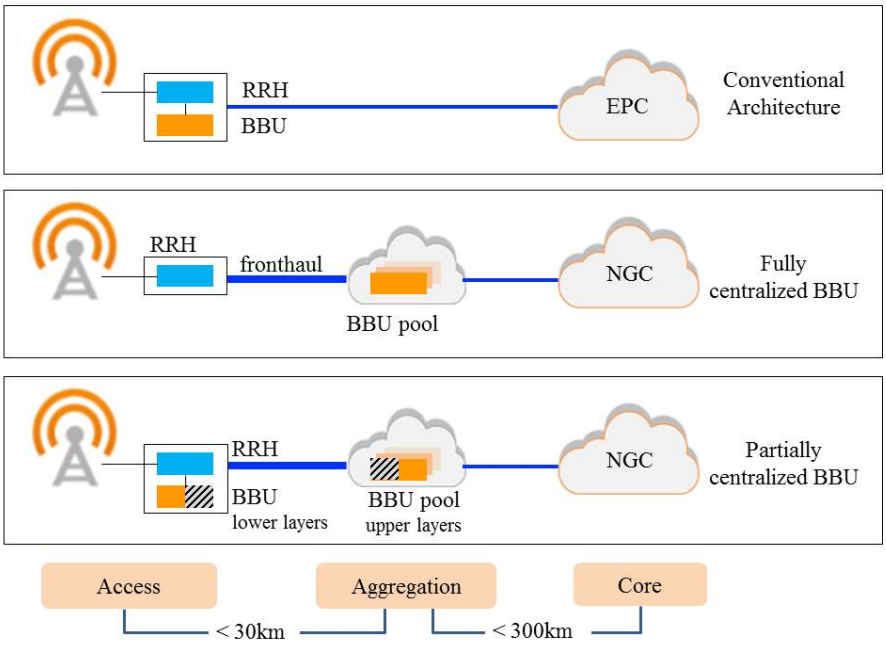


Figure 3.3: Cloud-RAN functional splits

3.2 Fronthaul analysis

3.2.1 Fronthaul size

The fronthaul size, i.e., the distance between the BBU-pool and antennas, is limited by the time-budget of the Round Trip Time (RTT) which includes the acknowledgment of each subframe. In Long Term Evolution (LTE), the acknowledgment messages and retransmission procedures in case of errors are handled by the HARQ process.

As shown in Figure 3.4, the BBU-pool has less than 3 ms for the whole base-band processing (namely, decoding, checking the Cyclic Redundancy Check (CRC), and encoding the ACK/NACK). The reception process (Rx) has a budget of 2 ms and the transmission process (Tx) 1 ms, denoted by T_{Rx} and T_{Tx}, respectively; the turnaround time is 8 ms.

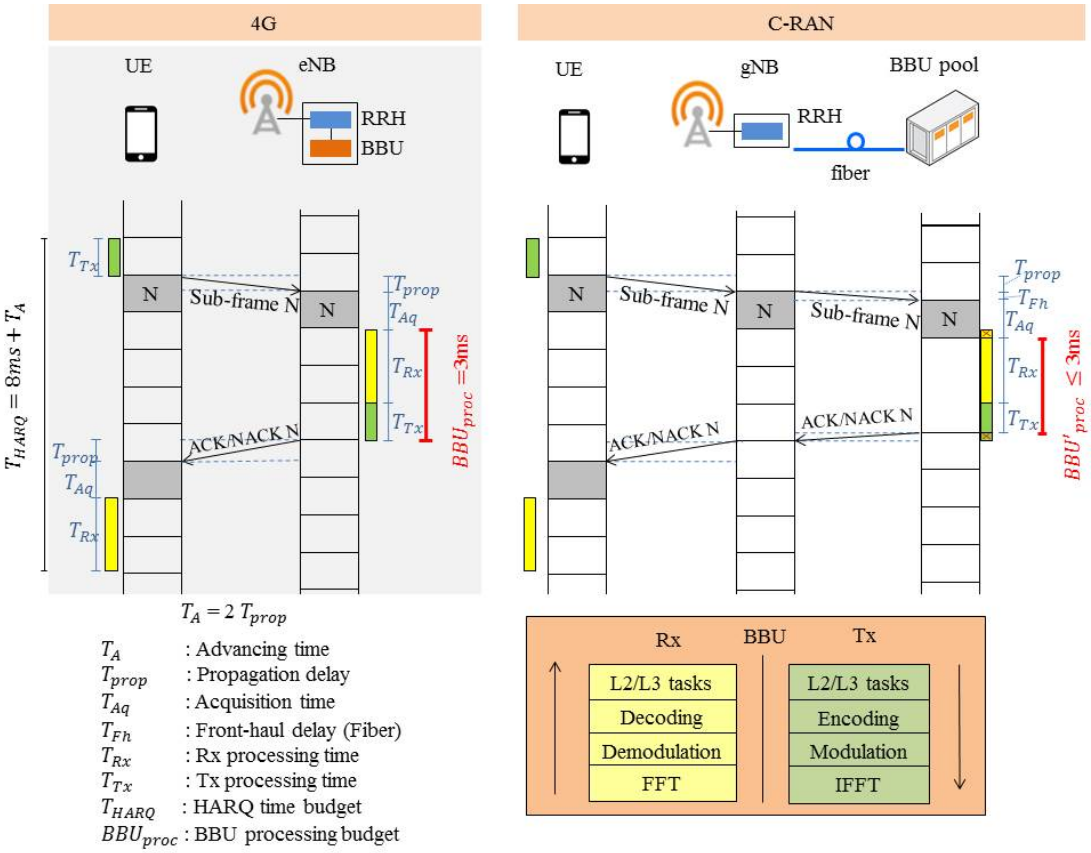


Figure 3.4: HARQ process in Cloud-RAN architectures.

In fact, to dimension the fronthaul size, it is first necessary to know the response time T_r of the BBU-pool, i.e., the required time to execute all BBU functions. Thus, the time-budget for the propagation of IQ signals, i.e., the fronthaul delay, is the remaining time after the base-band processing in the BBU-pool. Since the BBU-pool (CO) is linked with antennas by optic-fiber, the fronthaul size d can easily be obtained from the fronthaul time-budget, so-called fronthaul delay T_{Fh}, and the speed of light c, as $d = c * T_{Fh}$. Hence, the time budged for processing the whole baseband functions in a C-RAN system, BBU'_{proc}, is given by

$$BBU'_{proc} = BBU_{proc} - 2 * T_{Fh},$$

where BBU_{proc} is the 4G time budged (3 ms).

The HARQ mechanism considers an advancing time T_A in order to align signals in time due to propagation delay between the UE and the eNB. In LTE, there are 8 HARQ processes executed at the same time with an offset of 1 ms each which corresponds to the acquisition time of a subframe, T_{Aq}. Thus, $T_{HARQ} = RTT + T_A$, where

$$RTT = T_{Tx} + T_{Rx} + 2 * T_{Aq} + BBU'_{proc} + 2 * T_{Fh}.$$

Two significant points to take into account in a Cloud-RAN's fronthaul which will be predominantly fiber-based are:

- Attenuation due to greater fronthaul distances from few tens to cents of kilometers.

- Chromatic and polarization mode dispersion due to high fronthaul data rates (up to 10

Gbps) and long distances. The best way to prevent these phenomenons is employing coherent receivers and avoiding using fibers with high Polarization Mode Dispersion (PMD)

However, both aspects are out of the scope of this study.

3.2.2 Fronthaul capacity

One of the main issues of Cloud-RAN is the required fiber bandwidth to transmit base band signals between the BBU-pool (namely, Central Unit (CU)) and each antenna (namely, Distributed Unit (DU) or RRH).

The fronthaul capacity is defined by the number of gNBs hosted in the CO. The current widely used protocol for data transmission between antennas and BBUs is Common Public Radio Interface (CPRI) which transmits IQ signals. The transmission rate is constant since CPRI is a serial Constant Bit Rate (CBR) interface. It is then independent of the mobile network load [98].

The problem relies not only on the constant bit rate used by CPRI [98] but on the high redundancy present in the transmitted I/Q signals.

Many efforts are currently being devoted to reduce optic-fiber resource consumption such as I/Q compression [99], non-linear quantization, sampling rate reduction, and even, packetisation of CPRI. Several functional splits of the physical layer are also being an object of study in order to save fiber bandwidth [98, 100]. The required fronthaul capacity for the various functional splits is given in subsection 3.2.4.

3.2.3 Fronthaul transmission protocols

There is an open debate concerning the adoption of the appropriated fronthaul transmission protocol over fiber. The problem relies not only on the constant bit rate performed by the currently used CPRI protocol but on the high redundancy present in the transmitted I/Q signals. Incoming CPRI variants, notably those proposed by Ericsson et al. [101] perform CPRI packetization via IP or Ethernet. A similar approach for Radio over Ethernet (RoE) is being defined by the IEEE Next Generation Fronthaul Interface (1914) working group [102]. It specifies the encapsulation of digitized radio In-Phase Quadrature (IQ) payload for both control and user data. The xRAN Forum which is formed by industrials and network operators is also producing an open specification for the fronthaul interface [103]. It considers intra-PHY splitting as defined by 3GPP in TR 38.801 [97].

3.2.4 Required capacity per Functional Split

Forthcoming 5G standards consider the coexistence of several functional splits. The evolved CPRI (eCPRI) [101] proposes to split the EUTRAN protocol stack at each layer, which leads to five split options. 3GPP considers a more ambitious configuration where each layer is decomposed into high and low functions [97], giving rise to eight options. See Figure 3.5 for an illustration. Each functional split responds the requirement of specific services. The most ambitious one (namely, the 'PHY-RF split' which corresponds to the option 8 of 3GPP TR 38.801 Standard [97]) pretends high levels of centralization and coordination which enables a more efficient resource management (e.g., pooling of PHY resources) and radio performance. Nevertheless this configuration, here referred to as 'Functional Split (FS)-I', brings some deployment issues, notably, tight latency and high-bandwidth on the fronthaul requirements, as well as high-performance computing of the virtualized PHY functions.

We have illustrated in Figure 3.5 the various functions executed in a classical RAN. For the downlink direction, IP data packets are first segmented by the PDCP and RLC layers. Then, the

MAC layer determines the structure of the subframes (of 1 ms in LTE) forming frames of 10 ms to be transmitted to UEs. Once the MAC layer has fixed the Transport Blocks allocation for the UEs, information is coded in the form of Code Blocks (CBs). Then, remaining L1 functions are executed on the encoded data for their transmission (modulation, Fourier transform, giving rise to I/Q signals). In the uplink direction, the functions are executed in reverse order.

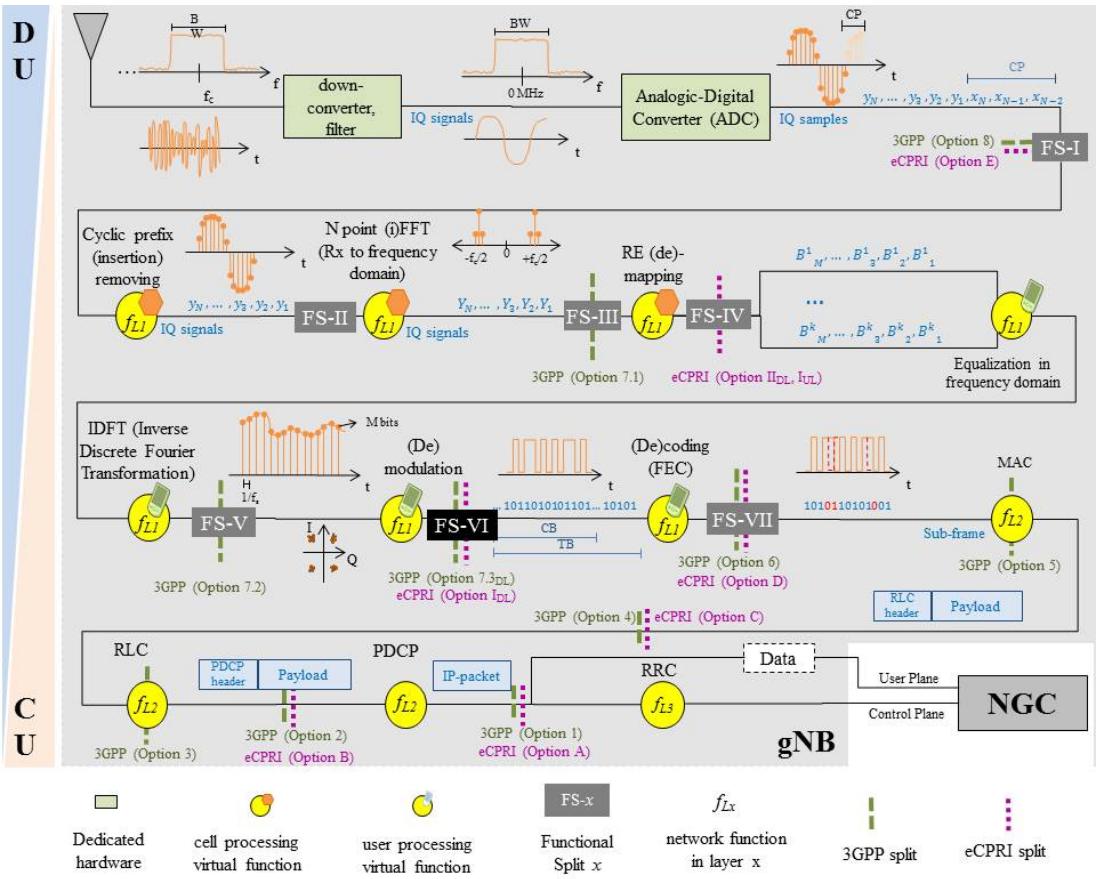


Figure 3.5: Fronthaul radio interfaces according to the various functional splits.

As illustrated in Figure 3.5, it is possible to separate the above various functions in order to implement a virtual RAN split into different pieces. In the following, we shall pay special attention to C-RAN supporting several BBU units in a same cloud infrastructure, thus offering high flexibility to implement radio resource management, coordination between radio sectors, etc. It is worth noting that new RAN implementations consider the coexistence of configurable functional splits where each of them is tailored to the requirements of a specific service or to a network slice. For instance, Ultra-Reliable Low-Latency Communications (URLLC) expects a one-millisecond round-trip latency between the UE and the gNB while enhanced Mobile Broad-Band (eMBB) demands only 4 milliseconds. The functional split actually defines the centralization level of RAN functions in the cloud-platform, i.e., it determines which functions are processed in dedicated hardware near to antennas (DU) and those which are moved higher in the network to be executed in centralized data centers (CU).

The required fronthaul capacity significantly decreases when the functional split is shifted after the PHY layer or even after the MAC layer [100]. In the following, we focus our analysis in the intra-PHY splitting options, keeping in mind that we are interested in a fully virtualized RAN architecture. The required fronthaul capacity (given in Mbps) for all intra-PHY functional splits, $FS-x$, is presented in Table 3.1 as a function of the cell bandwidth BW_{cell} (given in

MHz).

Functional Split I

The fully centralized architecture (Option 8 according to 3GPP), referred to as FS-I, only keeps in the DU the down-converter, filters and the Analogic-Digital Converter (ADC). IQ signals are transmitted from and to the CU by using the CPRI standard. The problem of FS-I is in the fact that the required fronthaul capacity does not depend on the traffic in a cell, but of the cell bandwidth i.e., the whole cell bandwidth is transmitted every millisecond even when there is less payload information. The required data rate per radio element is given by

$$\begin{aligned} R_1 &= 2 * M * f_s * F_{coding} * F_{control} * N_{ant} \\ &= 2 * M * N_{FFT} * BW_{sc} * F_{coding} * F_{control} * N_{ant}, \end{aligned}$$

where the various variables used in the above formula are given in Table 3.2.

Many efforts are currently being devoted for reducing the FS-I's required capacity, notably, the I/Q compression by redundancy removal or even by non-uniform quantization. These solutions are detailed in Appendix C.

In Table 3.1, we have computed the front capacity R_1 in function of the cell bandwidth. As the cell bandwidth increases, the required front haul capacity per sector and can reach 2.4 Gbps. Since each site is generally equipped with three sectors, we can observe that the required bandwidth reaches prohibitive values for this functional split.

Functional Split II

When implementing the Cyclic Prefix (CP) removal in the DU, the fronthaul capacity can be reduced. This solution may experiment correlation problems due to the Inter-symbol interference (ISI) apparition.

$$R_2 = 2 * M * N_{FFT} * (T_s + T_{CP})^{-1} * F_{coding} * F_{control} * N_{ant},$$

where T_{CP} is the average duration of a CP in a radio symbol. $T_{CP} = (500[us] - T_s[us] * 7)/7 =$ 4.76191 microseconds. $T_s = \frac{1}{BW_{sc}} = \frac{1}{15KHz}$. The useful data duration in a radio slot (500 microseconds) is given by $T_{UD-psl} = T_s * N_{sy-psl} = 66.67 * 7 = 466.67$ microseconds. The resulting fronthaul capacity is given in Table 3.1. The reduction in bandwidth requirement is rather small.

Table 3.1: Required fronthaul capacity in a Cloud-RAN system.

BW_{cell}	1.4	3	5	10	15	20
FS-I	153.6	307.2	614.4	1228.8	1843.2	2457.6
FS-II	143.4	286.7	573.4	1146.9	1720.3	2293.8
FS-III	86.4	172.8	360.0	720.0	1080.0	1140.0
FS-IV	60.5	121.0	252.0	504.0	756.0	1008.0
FS-V	30.2	60.5	126.0	252.0	378.0	504.0
FS-VI	6.0	12.1	25.2	50.4	75.6	100.8
FS-VII	5.5	11.1	23.1	46.2	69.3	92.4

Table 3.2: List of symbols.

Parameter	Description	Value
BW_{sc}	sub-carrier bandwidth	15 KHz
BW_{LTE}	LTE bandwidth	$1.4, 3, 5, 10, 15, 20$ MHz
BW_{uf}	useful bandwidth	18 MHz (20MHz)
f_c	nominal chip rate	3.84 MHz
f_s	sampling frequency	e.g., 30.72 MHz (20MHz)
F_{coding}	coding factor	10/8 or 66/64
$F_{control}$	control factor	16/15 (CPRI)
$F_{oversampling}$	oversampling factor	1.7
k	code rate	e.g., 11/12 [89]
M	number of bits per sample	15
N_{ant}	number of antennas for MIMO	e.g., $2x2$
N_{FFT}	number of FFT samples per OFDM symbol	e.g., 2048 (20MHz)
N_{RB}	total number of resource blocks per subframe	e.g., 100 (20MHz)
N_{sc}	total number of sub-carriers per subframe	e.g., 1200 (20MHz)
N_{sc-pRB}	number of sub-carriers per resource block	12
N_{sy-psl}	number of symbols per time slot	7 (normal CP)
N_{sy-psf}	number of symbols per subframe	14 (normal CP)
O_m	modulation order	2-QPSK,4-16QAM,6-64QAM,8-256QAM
ρ	RBs utilization (mean cell-load)	0.7
R_x	data rate when using the x-th functional split	
T_{CP}	average duration of a cyclic prefix	4.76μs (normal CP)
T_s	symbol duration	66.67μs (normal CP)
T_{UD-psl}	useful data duration per time slot	466.67μs (normal CP)

Functional Split III

By keeping the Fast Fourier Transform (FFT) function near to antennas the required fronthaul capacity can be considerably reduced. In this case, radio signals are transmitted in the frequency domain from radio elements to the BBU-pool for the uplink and vice versa for the downlink. This solution prevents from the overhead introduced when sampling the time domain signal. The oversampling factor is given by $F_{oversampling} = \frac{N_{FFT}}{N_{sc}} = 1.7$, e.g., $F_{oversampling} = 512/300 = 1.71$ for an LTE bandwidth of 5MHz. The corresponding fronthaul bit rate is then given by

$$\begin{aligned} R_3 &= 2 * M * N_{sc} * BW_{sc} * F_{coding} * F_{control} * N_{ant} \\ &= 2 * M * N_{sc} * (T_s)^{-1} * F_{coding} * F_{control} * N_{ant} \end{aligned}$$

As illustrated in Table 3.1, the fronthaul capacity is halved when compared with the initial CPRI solution. In the following, we show that the fronthaul capacity can still be reduced by a factor 10.

Functional Split IV

When including the de-mapping process in the DU, it is possible to adapt the bandwidth as a function of the traffic load in the cell, then the required fronthaul capacity is directly given by the fraction of utilized radio resources [100].

Here, only the Resource Blocks (RBs) which carry information are transmitted. For instance,

an eNodeB serving a high-density zone (e.g, a train station) presents a RB utilization of 11.96% and 20.69%, in the uplink and downlink directions, respectively. The highest utilization values observed in current deployed eNBs do not exceed 70% in the downlink direction (as observed in the Orange mobile network). Thus, we take $\rho = 0.7$ as the mean cell-load. This yields a front haul bit rate equal to

$$R_4 = 2 * M * N_{sc} * BW_{sc} * F_{coding} * F_{control} * N_{ant} * \rho$$

Numerical values given in Table 3.1 show that the gain with respect to the previous solution is rather small.

Functional Split V

This configuration presents a gain in the fronthaul load when MIMO schemes are performed. The equalization function combines signals coming from multiple antennas, as a consequence, the required fronthaul capacity is divided by N_{ant}. The required front haul capacity is then given by

$$R_5 = 2 * M * N_{sc} * BW_{sc} * F_{coding} * F_{control} * \rho$$

Table 3.1 shows a drop by a factor 2 with respect to the previous solution.

Functional Split VI

By keeping the demodulation/modulation function near to antennas, the required data rate is given by:

$$R_6 = N_{sc} * N_{sy-psf} * O_m,$$

where N_{sy-psf} is the number of symbols per subframe (i.e.,$N_{sy-psf} = 14$ when using normal cyclic prefix), O_m is the modulation order, i.e., the number of bits per symbol where 6 is the maximum order currently supported in LTE. The required fronthaul capacity is then up to 100 Mbps. This represents a significant when compared to the initial CPRI solution. It is also worth noting that this solution preserves the gain achievable by C-RAN.

Functional Split VII

Just for the sake of completeness, we consider now the case when keeping the channel coding function near to antennas, redundancy bits are not transmitted. Nevertheless this configuration reduces the advantages of C-RAN. DUs become more complex and expensive. The required fronthaul capacity is

$$R_7 = N_{sc} * N_{sy-psf} * O_m * k,$$

where k is the code rate, i.e., the ratio between the useful information and the transmitted information including redundancy. In LTE code rate k commonly ranges from 1/12 to 11/12 [89]. In Table 3.2, we use $k = 11/12$ as the worst-case.

Functional split selection

In view of the analysis carried out in this chapter, a fully centralized architecture performing the functional split VI seems to be the most appropriate to achieve both cloud and radio resource utilization effectiveness.

We aim to encapsulate the fronthaul payload within Ethernet, i.e., distributed units (RRHs) are connected to the BBU pool through an Ethernet network. RoE is considered by IEEE

Next Generation Fronthaul Interface (1914) Working Group as well as by the xRAN Fronthaul Working Group of the xRAN Forum.

The main issue of an Ethernet-based fronthaul is the latency fluctuation [104]. Transport jitter can be isolated by a buffer, however, the maximum transmission time is constrained by the processing time of centralized BBU functions. The sum of both transmission and processing time must meet Radio Access Network (RAN) requirements (i.e., 1 ms - DL and 2 ms - UL).

The transmission time can quickly rise due to the distance and the added latency at each hop (e.g., switches) in the network. The transmission time can be roughly obtained from the light-speed in the optic-fiber (e.g., 2.1x10^8 m/s), and latency of 50μs by hop [104]. For instance, the required transmission time for an eNB located 40 km from the BBU-pool rises $280\mu s + 50 * 8 = 680\mu s$. Hence, the remaining time-budget for BBU processing is barely 320μs in the downlink direction.

3.3 Runtime evaluation

3.3.1 Service chain

The RAN (BBU) service chain is mainly composed by the encryption/decryption procedures of the PDCP, concatenation tasks executed by the RLC protocol, the radio scheduling and the HARQ management handled by the MAC layer, and sub-functions performed by the PHY layer (channel coding, modulation/demodulation, and OFDM signal generation). See Figure 3.6 for an illustration.

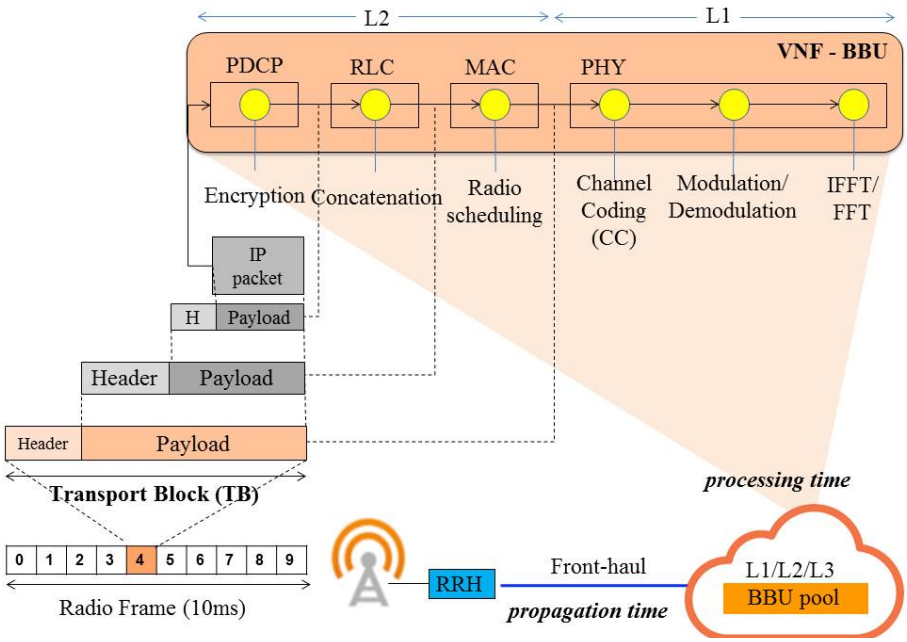


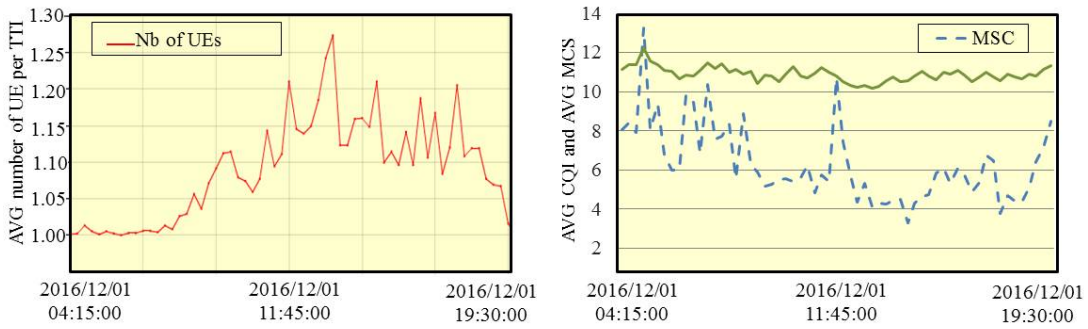
Figure 3.6: BBU functions.

Transmission service chain

RLC sub-functions are sequentially executed after the PDCP sub-function. The MAC layer and more specifically the radio scheduler allocates resources (time and frequency), namely PRB, to a UE in function of its data transmission needs and the radio channel quality. The number

of allocated PRB together with the MCS determine the Transport Block Size (TBS), i.e., the useful information. In other words, the MCS defines the modulation order O_m, and the coding scheme I_{MCS}.

To be more specific, the radio scheduler selects the UEs to be served in each Transmission Time Interval (TTI), i.e., one subframe of 1 millisecond [105]. The scheduler allocates resources according to some algorithm, for instance Round Robin or Proportional Fair among the various UEs which have data to receive or transmit. In general, the scheduler considers the radio conditions (namely CQI) to allocate radio resources while achieving some fairness criterion. Most common radio schedulers take into account the Channel Quality Indicator (CQI), the QoS Channel Indicator (QCI), the number of spatial layers, Inter-Cell Interference Coordination (ICIC), among others. In general, the CQI derives from channel models which are defined by vendors. The most important factor involved in the CQI measurement is the Signal-to Interference Noise Ratio (SINR). As an illustration, we captured the behavior of a commercial base station operating in a big city. A sample of one day is presented in Figure 3.7(a) for showing the average number of UEs scheduled per TTI. Similarly, the reported CQI and selected MCS is presented in 3.7(b).



(a) Average number of UEs per TTI

(b) Reported CQI and selected MCS

Figure 3.7: An example of commercial radio scheduler.

Reception service chain

The Rx base-band processing begins with the de-mapping process which selects the signal corresponding to a UE in order to demodulate and decode the received data. The MAC layer which handles the radio scheduling and HARQ mechanisms, performs de-multiplexing of the Uplink Shared Channel (UL-SCH) to restore the various logical channels containing the control messages and the IP packets.

3.3.2 VNF's runtime

When considering the execution of RAN functions (i.e., L1/L2/L3) in commodity hardware, the bottleneck notably resides in the runtime of PHY functions. Due to strict latency constraints (1 millisecond in downlink and 2 milliseconds in uplink), current mobile networks use high-throughput FPGA architectures to perform PHY functions, notably encoding and decoding processes.

In the following, we are concretely interested in quantifying the runtime of physical RAN functions by using a software-based-eNB (namely, an OAI-eNB). In order to emulate real radio conditions and an elastic number of connected UEs, we particularly build a traffic generator. These tools are presented below.

Traffic generation and eNB's settings

We evaluate the performance of a single cell with Single Input Single Output (SISO) configuration, 20 MHz of bandwidth and Frequency Division Duplex (FDD) transmission mode. We use an OAI-based eNB which runs in an x86-based computing platform of 2.6 GHz.

In order to emulate real radio conditions and multiple connected UEs, we generate the traffic load by simulation. We consider heterogeneous traffic where UEs with small needs in terms of radio resources coexist with UEs requiring high data rates. To be more specific, we build a resource grid such as those produced by commercial radio schedulers during the busy-hour [80]. An example of the emulated resource grid is shown in Figure 3.8.

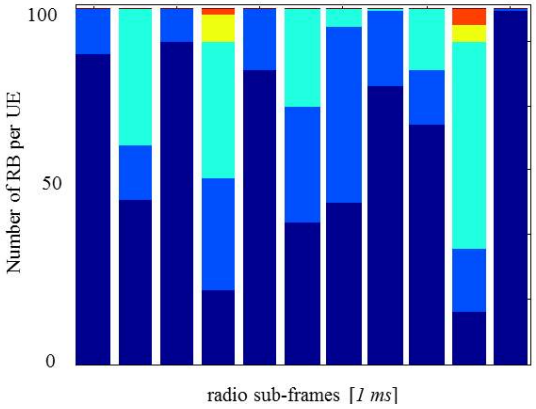


Figure 3.8: Emulated radio resource grid.

The simulation tool implements the Release 12 of LTE specifications for the PHY layer which is defined by the 3GPP TS 36.213, version 12.4.0 [106]. The scheduler selects both the MCS (redundancy bits) and the number of PRBs (radio resources) based on emulated radio conditions, the traffic in the cell, and the data load per UE. In order to study the worst case, we consider non-empty scheduling, i.e., all radio resources designated to each UE are used and need to be processed. The number of PRBs allocated to a UE ranges from 6 to 100 [106, 107]. The MCS varies between 0 and 27 and enables QPSK, 16-QAM, and 64-QAM modulation orders [106]. The detailed procedure is presented in Appendix D.

PHY UL runtime

While using the above described OAI-based eNB, we measure the runtime of the PHY layer during the reception process. We are concretely interested in the runtime performed per Transmission Time Interval (TTI) (1 millisecond). Figure 3.9 shows the execution time for the demodulation, channel decoding and FFT sub-functions. Besides the fact that the FFT runtime is constant and that the demodulation processing time is less fluctuating than the decoding process, it is worth noting that the runtime of the whole PHY layer is essentially determined by the decoding processing time.

Analysis: The FFT runtime only depends on the number of PRBs. The demodulation sub-function is directly influenced by the modulation order and the number of PRBs. The decoding process depends on the TBS, i.e., the CQI, the data load by UE and the traffic in the cell. Results show that the execution time of the PHY UL sub-functions can reach values of up to 1.8 ms. It does not leave enough margin to deploy a fully centralized RAN architecture which should additionally consider the fronthaul delay and the processing time of L2 and L3 functions within the reception budget, i.e., 2 ms.

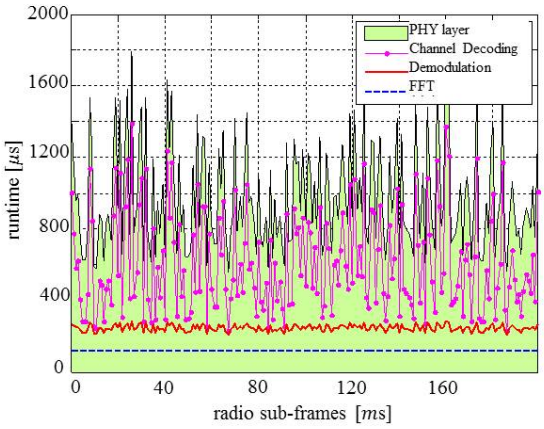


Figure 3.9: Runtime of virtual BBU functions (PHY layer, uplink).

PHY DL runtime

Similarly, the runtime of the PHY layer during transmission (downlink) for a single cell and multiple connected UEs is shown in Figure 3.10.

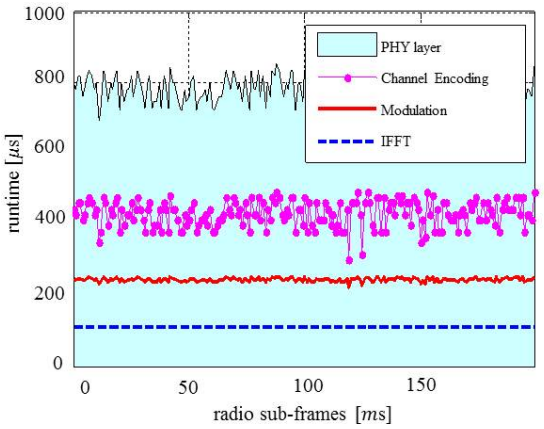


Figure 3.10: Runtime of virtual BBU functions (PHY layer, downlink).

Analysis: The execution time of the modulation sub-function depends on the modulation order and on the number of PRBs. The IFFT sub-function depends only on the number of PRBs. The Channel encoding depends on the TBS, i.e., the quality of radio conditions, the amount of data to be transmitted by each UE and the traffic load in the cell. Results show that the runtime of the whole PHY layer takes around 785 microseconds. It does not leave enough margin for processing L2 and L3 BBU functions. This fact may limit the implementation of centralized BBUs.

3.3.3 Runtime acceleration

As illustrated in the previous section, the execution of PHY-functions requires the largest part of computational capacity. The channel coding sub-function is the most expensive one in terms of latency in the uplink (decoding) as well as in the downlink (encoding) direction. In addition, unlike other PHY functions as modulation and IFFT/FFT, the channel coding processing has a non-deterministic behavior since it depends on the radio channel conditions [66, 79]. Thus, the

channel coding processing can be considered as the bottleneck when running RAN functions on commodity servers. These claims are in concordance with [66, 92].

In order to minimize the runtime of BBU functions and specially PHY functions, we investigate the relevance of parallel programing and resource pooling in multi-core systems. The main goal of the runtime reduction of RAN functions is twofold: *(i)* better resource utilization efficiency, while concentrating more gNBs within a given data center and *(ii)* moving higher in the backhaul network virtualized RAN functions.

In [108], namely CloudIQ, is shown that at least 22% of computing resources can be saved only when using statistical multiplexing and resource pooling in Cloud-RAN systems. It exploits the variations in the processing load of individuals BBUs to use fewer computing resources.

Decomposing monolithic RAN functions

For improving the efficient utilization of computing resources and notably for *reducing the runtime* of PHY RAN functions, we rely on the general philosophy of parallel computing which consists of splitting large tasks into smaller parallel runnable sub-tasks. The parallel execution of sub-tasks allows the runtime of the whole task to be shortened. Nevertheless, the parallel execution of RAN functions is not always possible since most tasks are chained, e.g., for a given data block (namely a subframe of 1 millisecond), the modulation process needs to be finished for launching the decoding process.

Current software-based RAN solutions (e.g., OAI, or even evolved models as CloudIQ [108]) execute an entire subframe in a single computing resource (e.g., core). We are concretely interested in splitting the subframe processing into parallel runnable tasks (data blocks) in order to decrease the execution time of the entire subframe. In other words, instead of functional parallelism, we propose using data parallelism.

In the following, we focus on the conception of parallel runnable data blocks in the aim of improving the performance in terms of latency. The main challenge when decomposing monolithic RAN functions into microservices or sub-functions is avoiding dependences, i.e, getting microservices or sub-functions that are able to run simultaneously in separated cores, ideally, without interactions between them.

Threading model

We propose the parallel processing of subframes as follows: the workload of a subframe is divided in slices where each of them corresponds to the data of a single UE, namely, a Transport Block (TB). These slices, so-called sub-tasks, are executed simultaneously on different cores. For further performance improvement, we propose splitting the workload of a single UE in parallel runnable Code Blocks (CBs).

Hence, the resulting threading model executes one thread per UE, and/or, one thread per CB for the channel decoding function. The threading model is illustrated in Figure 3.11. A global scheduler allocates a dedicated single core to each sub-task (either per UE or per CB) in order to avoid context switching overhead.

Concretely, we propose a dynamic multi-threading scheme (also referred to as thread-pool) which processes transport blocks by segments. A TB is the transmission radio unit which is composed of various RB, and these, in turn in Resource Element (RE). In this work, we consider RBs with normal CP (the most common in LTE networks) [109]. A RB consists of 12 sub-carriers in the frequency domain and 7 symbols in the time domain. Finally, a RE which consists of a single sub-carrier (15KHz) and one symbol, as shown in Figure 3.12.

In spite of REs are the smallest defined unit, they cannot be processed in parallel since the information contained therein is not meaningful. In fact, a CB is the smallest data unit that can

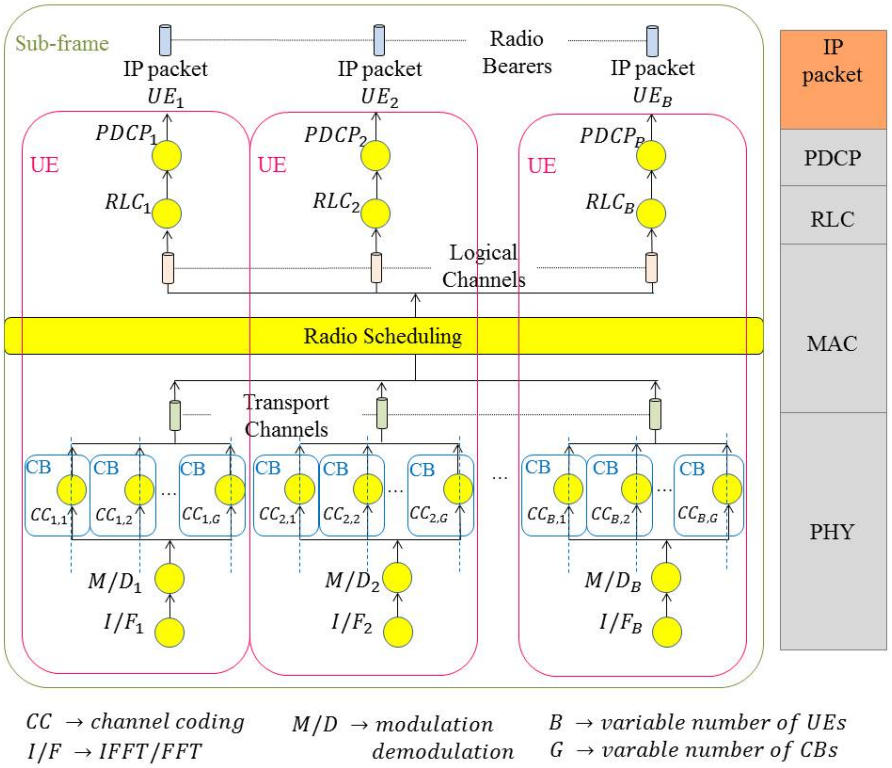


Figure 3.11: Functional and data decomposition of BBU functions: Multi-threading model.

be individually processed by the channel coding function [81]. See Figure 3.12 for an illustration. In current mobile networks (LTE) the code block size varies from 40 to 6144 bits where the last 24 bits of each CB corresponds to CRC.

When performing parallelism per UEs, the number of parallel coding jobs is determined by the number of UEs scheduled in a radio subframe. When applying parallelism per CBs, the number of parallel coding jobs is given by the product of the number of scheduled UEs and the number of CBs for each of them. To be more specific, the number of parallel channel coding threads per UE depends on the TBS. When the TBS is not big enough, parallelism is not performed since only one CB is produced. Conversely, when the TBS takes the maximum standardized value, the number of parallel threads is given by $\lceil \text{TBS/CBS} \rceil$. See Figure 3.13 for an illustration.

Modulation sub-function is sequentially executed after the code block concatenation. Finally, the OFDM signal is generated after the modulation sub-function is completed.

Figure 3.14 shows an illustration of additional procedures required for MIMO. Note that the transport block size is also different.

Scheduling Strategy

In the framework of parallel computing in multi-core platforms, we define a global scheduler which selects the next job to be processed among those which are ready to be executed. The scheduler allocates a single processing unit (core) to execute the next job in the queue in a First In Firs Out (FIFO) order. We use non-preemptive scheduling, in this way, a running job cannot be interrupted and it is executed until completion. This principle enables avoiding context-switching overhead. The scheduling procedure is detailed in Appendix E [79].

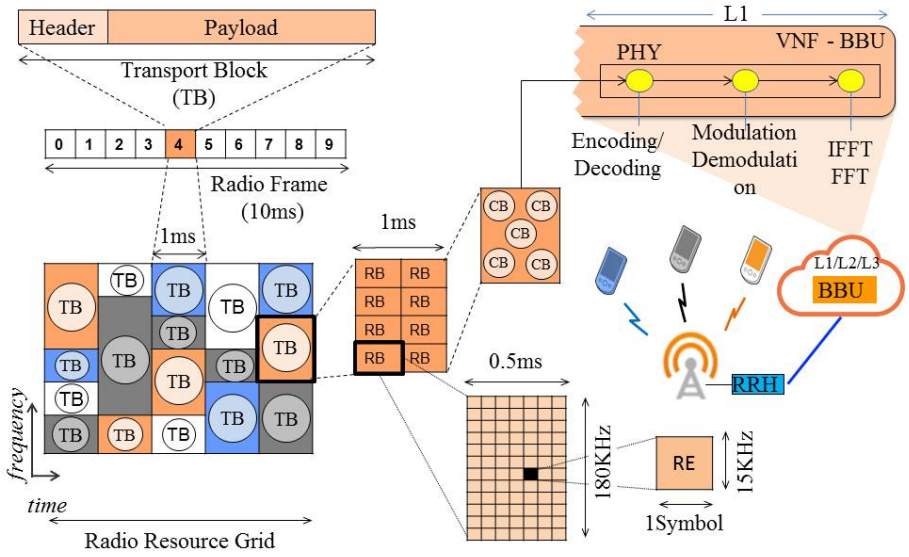


Figure 3.12: Radio data units.

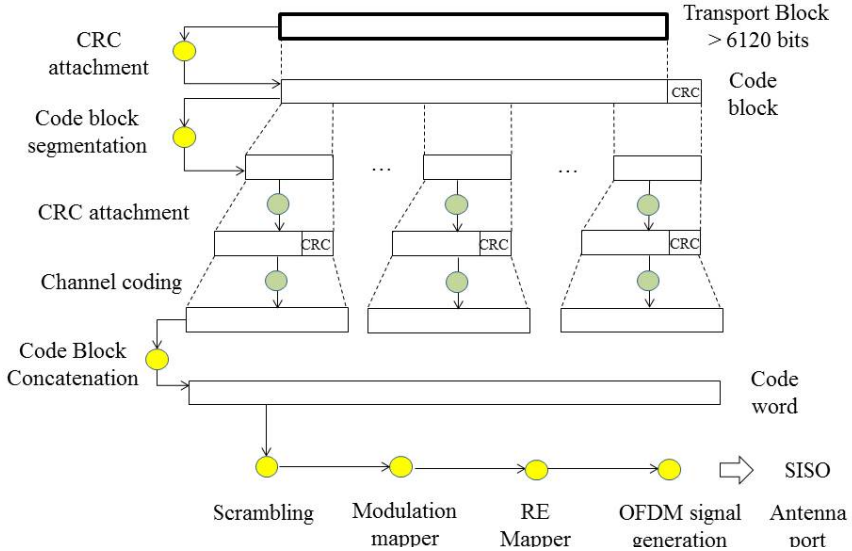


Figure 3.13: PHY threading model.

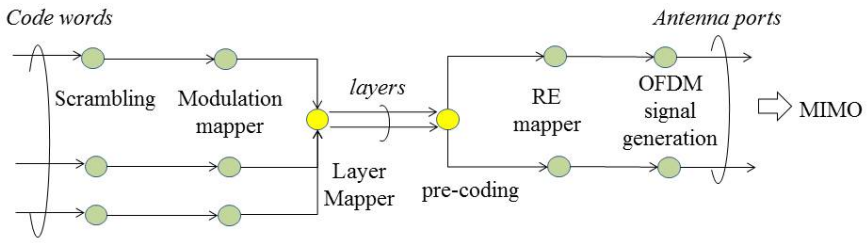


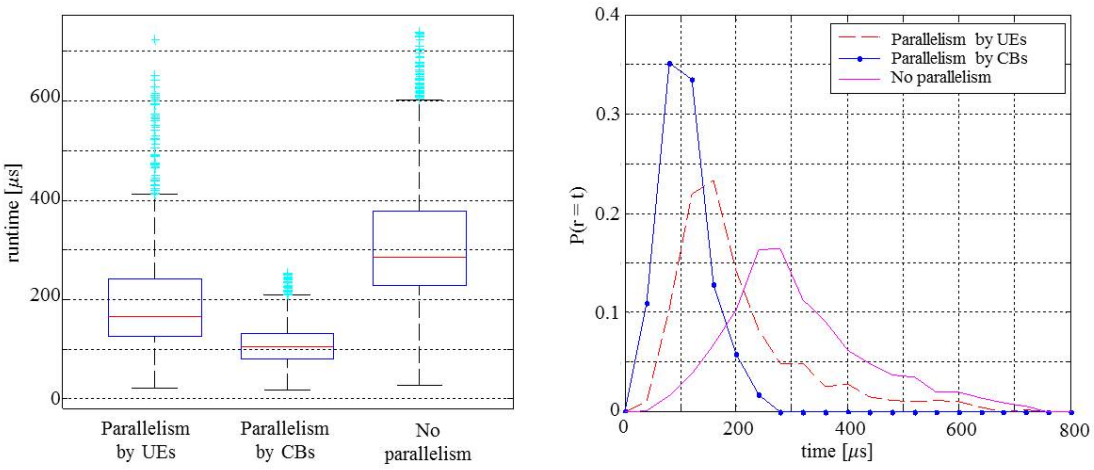
Figure 3.14: MIMO threading model.

3.3.4 Performance gain

We present below the gain that can be obtained when implementing the proposed threading model (Figure 3.11). We concretely evaluate by simulation the execution time of virtualized BBU functions of a single eNB running in a multi-core platform with C cores. We use the previously defined OAI-eNB and traffic conditions (see subsection 3.3.2), i.e., FDD transmission mode, SISO configuration and 20 MHz of bandwidth. A non-preemtive global scheduler allocates the capacity of cores to the various jobs. The scheduling strategy is described in 3.3.3.

We measure the runtime of the decoding sub-function when using a multi-core platform of 6 cores. We evaluate the gain when running both UEs and CBs in parallel, with respect to no-parallelism, i.e., the serial processing of jobs. The system executes up to eleven threads per UE when it has the maximum number of RBs and the maximum modulation order according to the LTE-Release 12.

Performance results are shown in Figure 3.15(a). The runtime of the decoding function is divided by 3 when executing CBs in parallel and halved when applying threads per UEs.



(a) runtime

(b) PDF

Figure 3.15: Channel decoding performance in a single eNB, $C = 6$.

The study of the probability density function of the channel decoding runtime shows that when executing one thread per CB, runtime values are more concentrated around the mean.

Even if the reduction experimented when running UEs in parallel is already very interesting in terms of latency, the parallelism by CBs offers better performance, especially when comparing the tail of both probability density functions. See Figure 3.15(b) for an illustration. Note that when applying parallelism by UEs, the probability density function exhibits a long tail that is similar to that obtained when there is no parallelism.

To be more specific, parallelism by CBs presents less statistical dispersion, i.e., the execution of the channel decoding function is near to a specific value (e.g., 300 μs for a multi-core platform with 6 cores). This fact is crucial for the cloudification of RAN functions. Parallelism by CBs requires notably more developing effort than parallelism by UEs. Then, this latter may be advantageous when considering implementation complexity.

Similar results are performed during the transmission processing (downlink direction). See [79].

3.4 Worst-case study

3.4.1 Worst-case definition

In Cloud-RAN, the worst-case in terms of execution time in the downlink direction is obtained when processing large-sized TBs, i.e., when a single UE is scheduled per subframe and has the best radio channel conditions. The radio scheduler selects then the highest modulation order (e.g. 64-QAM, $O_m = 6$) and the highest code-rate (i.e., no-redundancy bits are inserted, $k \to 1$). Conversely, when the radio scheduler allocates short-sized TBs (e.g., 6 RBs per UE and QPSK modulation), the minimum TB's runtime is performed. See Figure 3.16 for an illustration. In the uplink direction, in addition to the workload (TBS) the runtime is specially determined by the number of iterations performed during the channel decoding process. In fact, when a UE experiments poor radio conditions, the Bit Error Rate (BER) increases and consequently, the time of convergence of the channel error corrector (namely, turbo decoding).

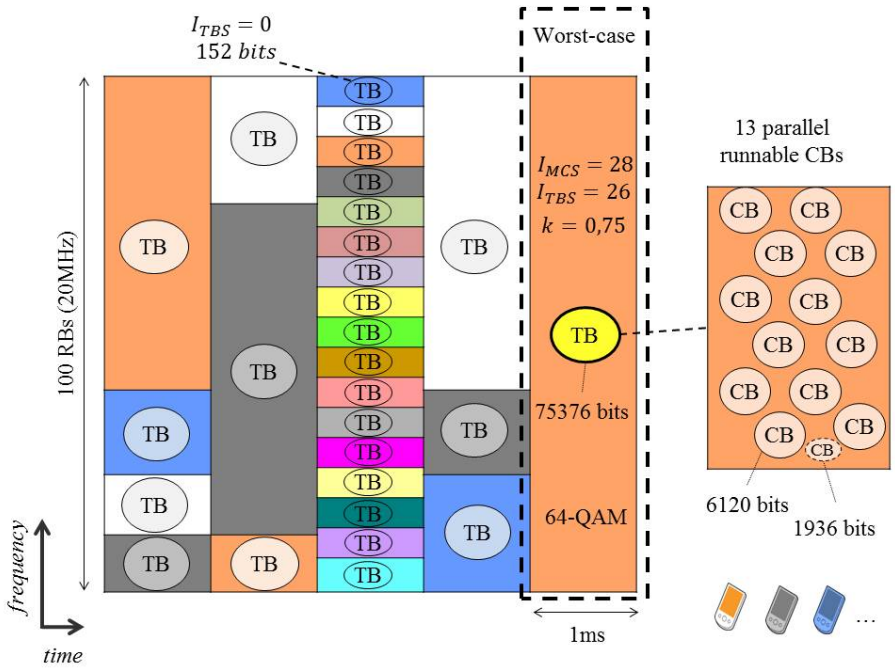


Figure 3.16: Cloud-RAN processing, worst-case analysis.

For an illustration, we determine the runtime of the channel coding function as a function of the MCS[2] while varying the traffic conditions of a single-UE. We evaluate the execution time for both uplink and downlink directions, i.e., decoding and encoding sub-functions respectively. The channel coding runtime as well as the worst case scenario are shown in Figure 3.17.

3.4.2 BBU-pool analysis

We analyze the performance of a BBU-pool in terms of latency, i.e., the response time r of a Cloud-RAN system which involves the sojourn time of BBU functions in the Cloud and the propagation delay between the BBU-pool and antennas.

We evaluate by simulation the behavior of a BBU-pool hosting 10 eNBs. Remote antennas (namely, RRH) are distributed within a 100 km radius around the BBU-pool,i.e., the workload of

[2]We use an OAI-based eNB whose settings are described in 3.3.2.

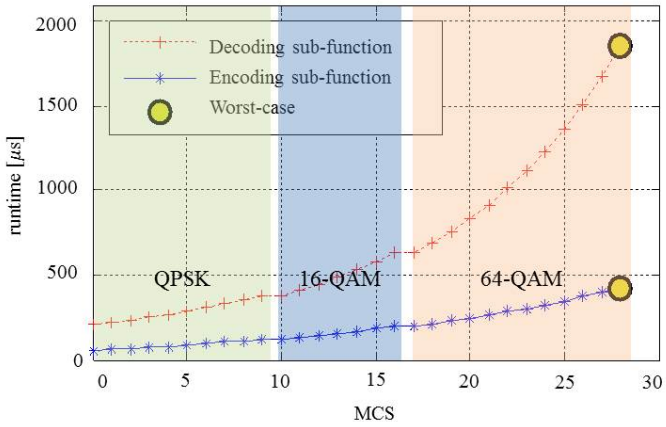


Figure 3.17: Channel coding runtime as a function of the MCS, 100 RBs.

all antennas arrive at the same time (worst-case). The propagation delay can be easily obtained from the light-speed ($2.5 * 10^8$m/s) in the optic fiber.

The base-band functions (encoding and decoding) are executed on a multi-core platform of C cores. All eNBs are set with identical characteristics as described in 3.3.2. They work on FDD transmission mode with SISO configuration and a bandwidth of 20 MHz.

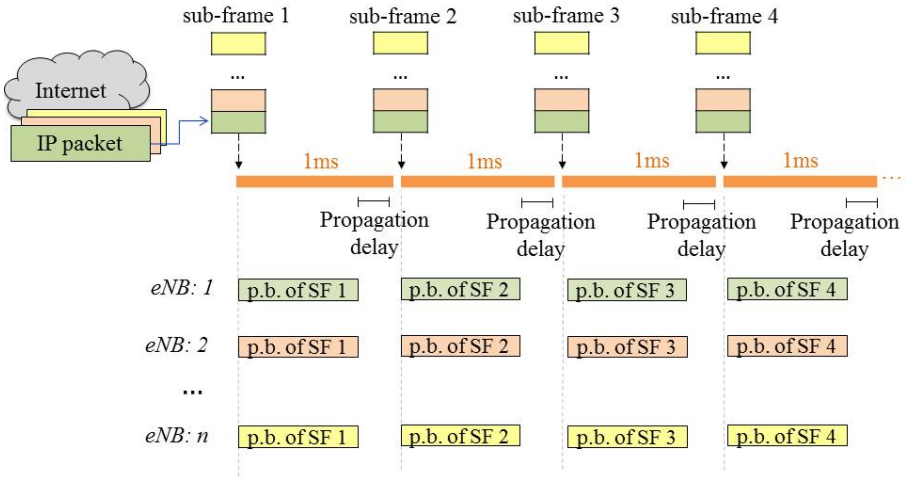


Figure 3.18: Downlink workload of a BBU-pool, worst-case scenario.

Thus, the multi-core system deals with two groups of subframes, one containing downlink subframes and the other the uplink workload; both arriving with a periodicity of 1 ms, The whole base-band processing of a subframe must be completed within 1 ms in the downlink, and 2 ms in the uplink direction. In Cloud-RAN, these time budgets must account not only the runtime of BBU functions, but also the propagation delay which is proportional to the fronthaul size. See Figures 3.18 and 3.19 for an illustration of downlink and uplink, respectively. When a subframe cannot be processed within the LTE time-budget, the subframe is lost. We are then interested in determining the minimum number of cores that is required for processing all subframes without exceeding their deadlines.

In a BBU-pool, thousands of subframes require service every millisecond. First-come, First-served (FCFS) becomes then highly interesting because the scheduling overhead due to context switching among subframes is minimal. In other words, a subframe is executed upon termination.

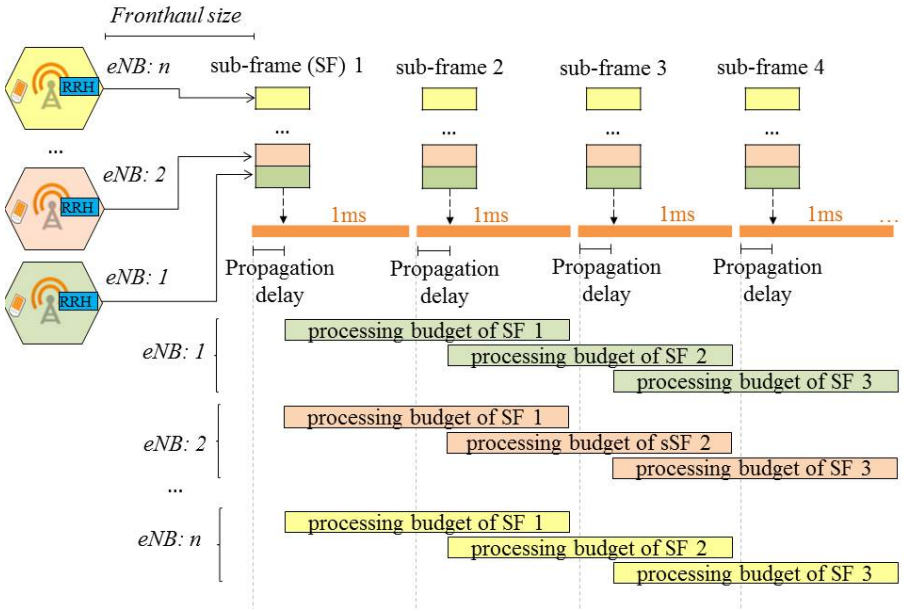


Figure 3.19: Uplink workload of a BBU-pool, worst-case scenario.

However, the impact of convoy effect (i.e., slow jobs hold the computing capacity keeping fast jobs on wait) need to be studied. When using data parallelism in the base-band processing of LTE subframes, jobs might have similar size (e.g., when encoding/decoding subframes by parallel runnable CBs). It will enable the mitigation of convoy effect.

3.4.3 Cloud-RAN sizing

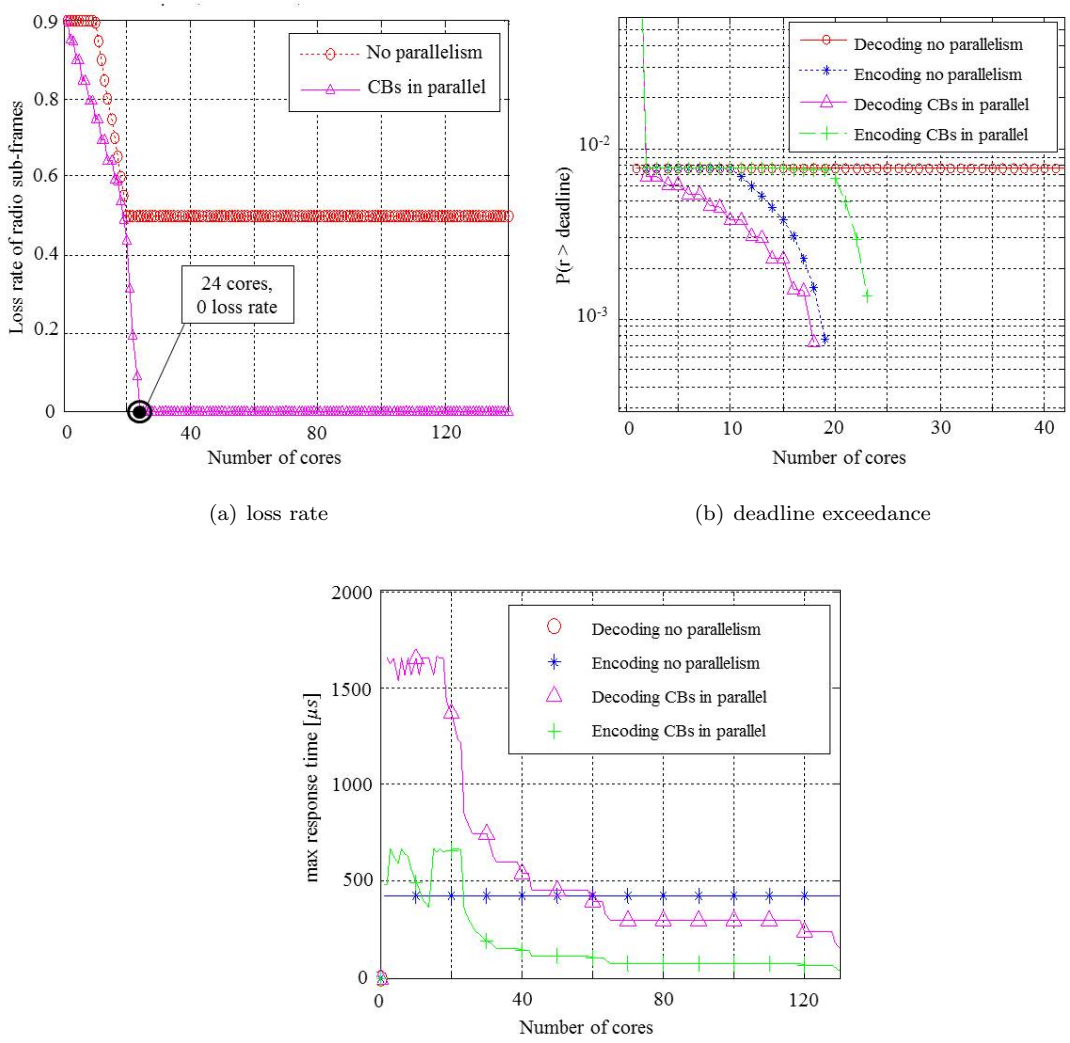
With the basis of the worst-case definition, we investigate the problem of determining the required computing capacity for hosting the base-band processing of various eNBs (say, 10) in a CO.

Figure 3.20(a) shows the loss rate of subframes as a function of the number of processing units. We can easily deduce that the required number of cores for processing a BBU-pool of 10 eNBs is $C = 24$. (This capacity achieves zero loss rate of LTE subframes when using parallelism by CBs.) Under this value, LTE deadlines are not respected as shown in Figure 3.20(b).

Figure 3.20(c) presents the speedup evolution of the execution of both encoding and decoding processes as a function of the number of cores. When parallelism is not performed, the speedup is limited by the biggest TBS allocated into a subframe, which depends on the LTE bandwidth. When executing CBs in parallel, it is bounded by the runtime of a CB of 6120 bits. This behavior is consistent with Amdahl's law, i.e., the performance improvement of parallel processing reaches at some time an upper boundary due to the serial part of the program [110].

3.4.4 Resource pooling performance

We are then interested in evaluating the behavior of the BBU-pool in real traffic conditions. The performance is also assessed when the computing capacity is both under and over sized with respect to the worst-case sizing. The loss rate of subframes according to the scheduling strategy and the number of cores is presented in Table 3.3. Results show that even when using a very high capacity (i.e, $C \gg 24$), both no parallelism and parallelism by UEs present non-zero loss rate.



(a) loss rate

(b) deadline exceedance

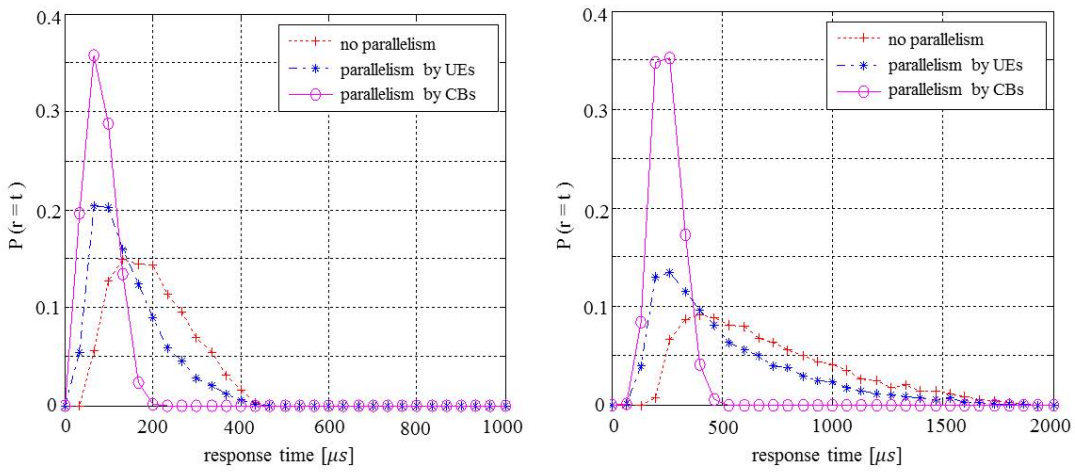
(c) speedup evolution

Figure 3.20: BBU-pool performance (10 eNBs), worst-case scenario.

Table 3.3: Loss rate of subframes in a BBU-pool of 10 eNBs.

No parallelism	Parallelism by UEs	Parallelism by CBs	C
0.00370	0.00260	0.00000	$C < 24$
0.00200	0.00060	0.00000	$C = 24$
0.00190	0.00055	0.00000	$C \gg 24$

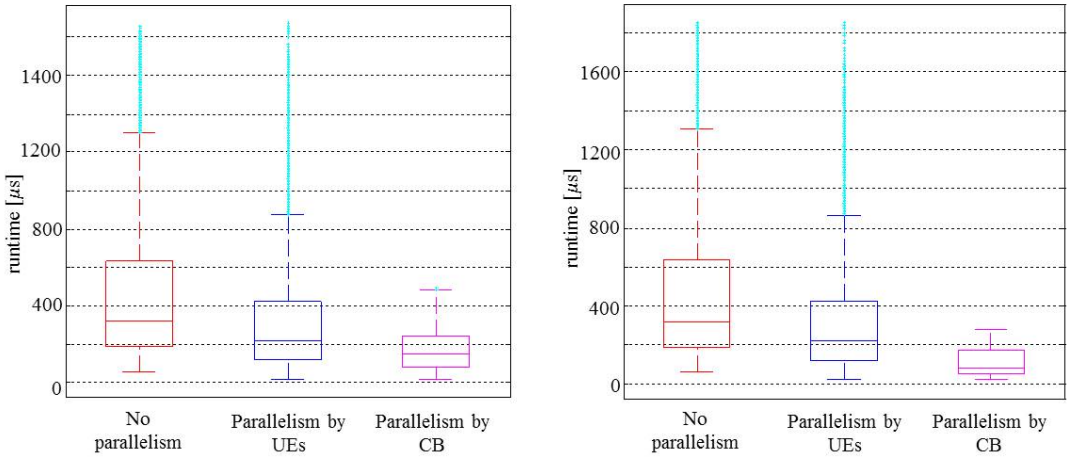
The response time of a Cloud-RAN system can experiment high variability. In fact, the processing time is a function of the traffic load as well as of the radio scheduler behavior. This latter determines the number of UEs allocated per subframe and the TBS of each of them. In addition, each eNB introduces a different delay in the arrival of its subframes at the CO. This delay depends on the eNB location. The probability density function of the channel coding runtime in a Cloud-RAN system hosting 10 eNBs is shown in Figure 3.21.



(a) encoding (DL) (b) decoding (UL)

Figure 3.21: Probability density function of channel coding runtime, 10 eNBs, $C = 24$.

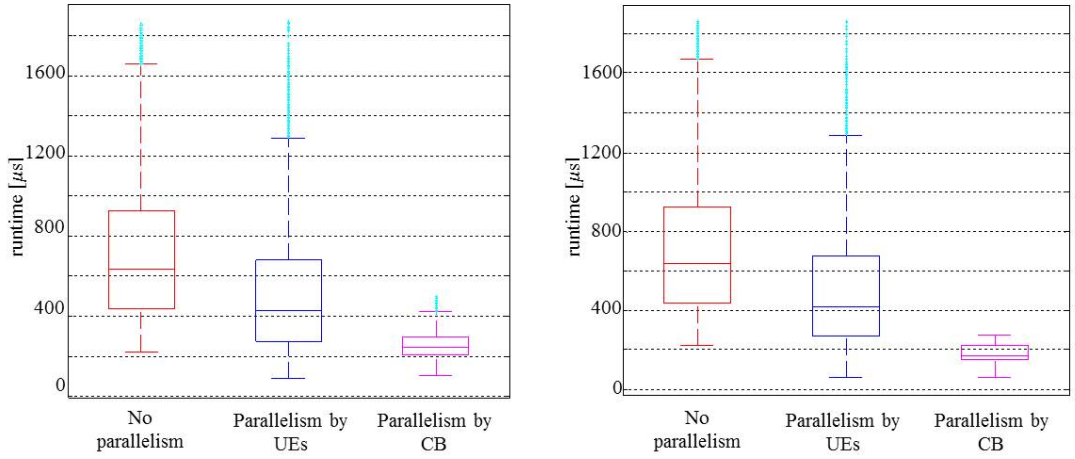
Results show that parallelism by CBs enables less dispersion, i.e., response time values are more concentrated around the mean. We are also interested in the minimum response time that can be achieved when over-sizing the computing capacity. Results including both uplink and downlink processing are shown in Figure 3.22. We observe that when increasing the computing capacity, the performance can be improved only when using the finest granularity of data-parallelism.



(a) $C = 24$ (b) $C = 130$

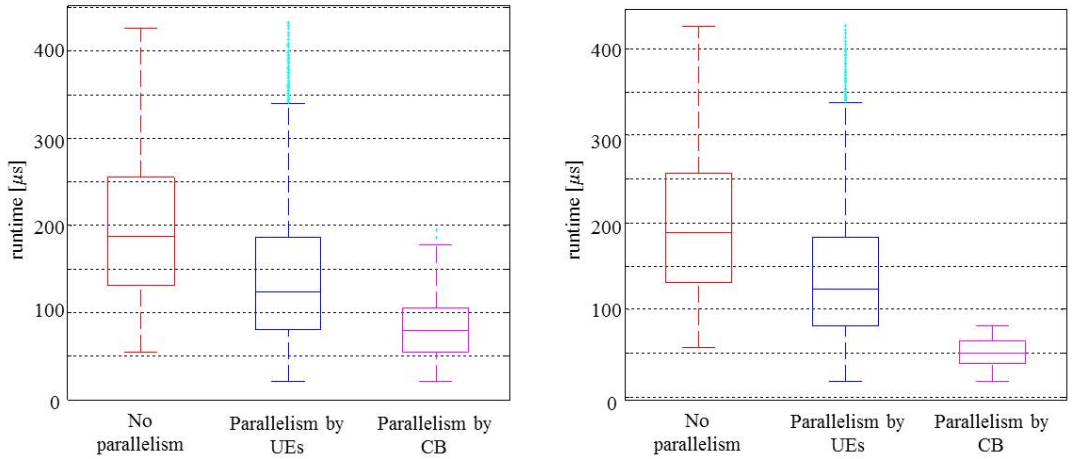
Figure 3.22: Cloud-RAN performance (UL+DL) for real traffic conditions, 10 eNBs.

In fact, parallelism per CBs offers the best performance and can significantly reduce the processing time of channel coding function in the BBU-pool. This function can then be executed within one third of the time budget and the encoding function can be performed within one fifth of the LTE deadline. See Figures 3.23 and 3.24 for an illustration.



(a) $C = 24$ (b) $C = 130$

Figure 3.23: Decoding performance (UL), 10 eNBs.



(a) $C = 24$ (b) $C = 130$

Figure 3.24: Encoding performance (DL), 10 eNBs.

Chapter 4

C-RAN modeling for dimensioning purposes

Contents

4.1 Modeling Principles 65
4.2 Batch model 69
4.3 Numerical experiments 75

We study in this chapter a batch queuing model, namely the $M^{[X]}/M/C$ multi-service system, to assess the needed processing capacity in a data center while meeting RAN latency requirements. The proposed model is validated by simulation when processing a hundred base stations in a multi-core system. These matters are presented in [111, 112].

4.1 Modeling Principles

4.1.1 Modeling data processing

From a modeling point of view, each antenna (RRH) represents a source of jobs in the uplink direction, while for the downlink direction, jobs arrive from the core network, which provides connection to external networks (e.g., Internet or other service platforms). There are then two queues of jobs for each cellular sector, one in each direction. Since the time-budget for processing downlink subframes is half of that for uplink ones, they might be executed separately on dedicated processing units. However, dedicating processors to each queue is not an efficient way of using limited resources.

Nelson et al. in [7] evaluate the performance of different parallel processing models when considering "centralized" (namely, single-queue access on multi-core systems) and "distributed" architectures (namely, multi-queue access on multi-core systems). Parallelism (so-called "splitting") and no-parallelism (so-called, "no splitting") behaviors are also considered. Results show that for any system load (namely, ρ) the lowest (highest) mean job response time is achieved by the "centralized/splitting" ("distributed/no splitting") system, i.e., the best performance in terms of latency (response time) is achieved when processing parallel runnable tasks in a single shared pool of resources. See Figures 4.1 and 4.2 for an illustration.

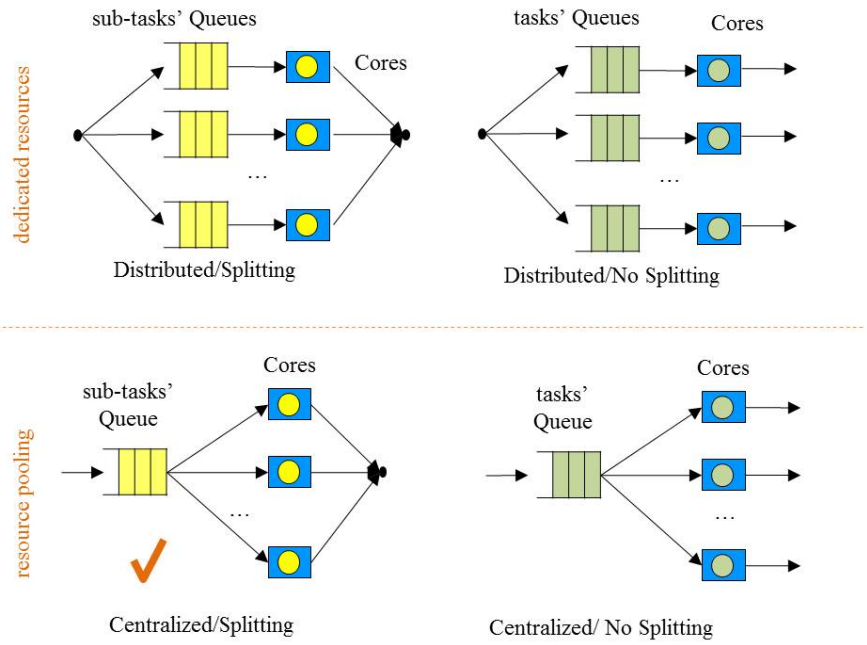


Figure 4.1: Parallel processing models [7].

In view of the above observations, we propose to use a single-queuing system with a shared pool of processors, namely a multi-core system with C cores. A global scheduler allocates computing resources to each runnable encoding (downlink) or decoding (uplink) job.

We assume that vBBUs (notably, virtual encoding/decoding functions) are invoked according to a Poisson process, i.e., inter-arrival times of runnable BBU functions are exponentially distributed. This reasonably captures the fact that there is a sufficiently great number of antennas, which are not synchronized. The occurrence of jobs then results from the superposition of independent point processes. This justifies the Poisson assumption. In practice, frames occur with fixed relative phases. The Poisson assumption is then in some sense a worst case assumption. Job arrivals are not synchronized because RRHs are at different distances of the BBU-pool. Furthermore, when considering no dedicated links, the fronthaul delay (inter-arrival time) can strongly vary because of network traffic.

The parallel execution of encoding and decoding tasks on a multi-core system with C cores can be modeled by bulk arrival systems, namely, an $M^{[X]}/G/C$ queuing system[1]. We further consider each task-arrival to be in reality the arrival of B parallel runnable sub-tasks or jobs, B being a random variable. Each sub-task requires a single stage of service with a general time distribution. The runtime of each sub-task depends on the workload as well as on the network sub-function that it implements. The number of parallel runnable sub-tasks belonging to a network sub-function is variable. Thus, we consider a non-fixed size bulk to arrive at each request arrival instant. The inter-arrival time is exponential with rate λ. The batch size B is independent of the state of the system.

In the case of Cloud-RAN, full functional parallelism is not possible since some base-band procedures (i.e., IFFT, modulation, etc.) require to be executed in series. However, data

[1]By analogy, concurrent computing of VNFs can be formalized by a single server queuing system with a processor sharing discipline and batch arrivals, referred to as $M^{[X]}/G/1-PS$, where the single service capacity is done by the addition of individual capacities of all cores in the system. Because switching tasks produces undesirable overhead, this approach is not further considered in the present study. Note that, the $M^{[X]}/M/1-PS$ queuing system is presented in Section 2.6.

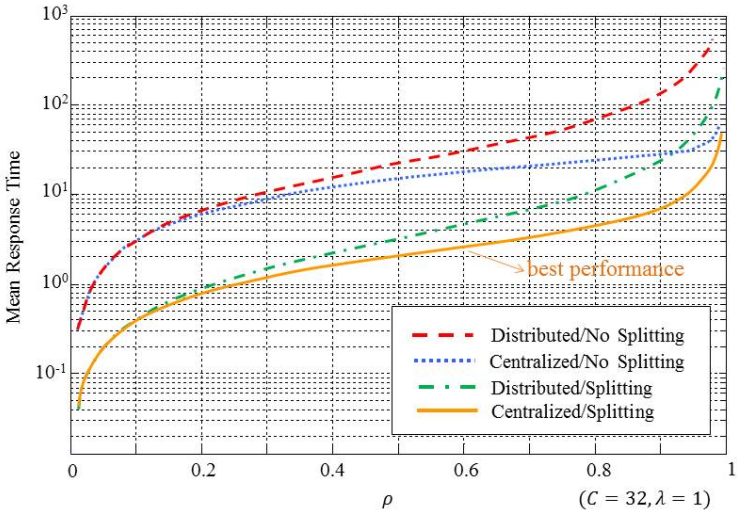


Figure 4.2: Performance of parallel processing systems [7].

parallelism of BBU functions (notably decoding and encoding) promises significant performance improvements. These claims are thoroughly studied in [79,80]. Results show that the runtime of BBU functions can be significantly reduced when performing parallel processing in a subframe, i.e., through the parallel execution either of UEs or even of smaller data units, so-called CBs. We present below a stochastic service model for each of the parallelization schemes in order to evaluate the performance of a Cloud-RAN system.

4.1.2 Parallelism by UEs

In LTE, several UEs can be served in a subframe of 1 millisecond. The maximum and minimum number of UEs scheduled per subframe are determined by the eNB bandwidth. LTE supports scalable bandwidth of 1.25, 2.5, 5, 10 and 20 MHz. In a subframe, each scheduled UE receives a TB (namely, a group of radio resources in the form of RB) either for transmission or reception. For example, when considering an eNB of 20 MHz, 100 RBs are available. According to LTE [106], the minimum number of RBs allocated per UE is 6. Hence, the maximum number of connected UEs per subframe is given by $b_{max} = \lfloor 100/6 \rfloor$. The TBS is determined by the radio scheduler in function of the individual radio channel conditions of UEs as well as the amount of traffic in the cell.

From the previous section, the parallel base-band processing (notably channel coding) of LTE subframes can be modeled as an $M^{[X]}/G/C$ queuing system. When considering parallelization per UE, the number of jobs within a batch corresponds to the number of UEs scheduled in a radio subframe, e.g., the number of decoding jobs per millisecond in an eNB of 20 MHz range from 1 to 16. A subframe then comprises a variable number of UEs, which is represented by the random variable B.

We further assume that the processing time of a job (namely that of a TB) is exponential. This assumption is intended to capture the randomness in the processing time of UEs due to non-deterministic behavior of the channel coding function. For instance, the decoding runtime of a single UE can range from a few tens of microseconds to almost the entire time-budget[2], i.e., 2000 microseconds [86]. In practice, this service time encompasses the response time of each component of the cloud computing system, i.e., processing units, RAM memory, internal buses,

[2]Runtime values are for reference and correspond to the execution of OAI-based channel coding functions on x-86-based General Purpose Processors (GPPs) of 2.6 GHz.

virtualization engine, data links, etc. In the following, we precisely assume that the service time of a TB (i.e., a job) is exponentially distributed with mean $1/\mu$. If we further suppose that the number B of UEs per subframe is geometrically distributed with mean $1/(1-q)$ (that is $\mathbb{P}(B=k)=(1-q)q^{k-1}$ for $k \geq 1$), the complete service time of a frame is then exponentially distributed with mean $1/((1-q)\mu)$.

The geometric distribution as the discrete analog of the exponential distribution capture the variability of scheduled UEs in a subframe. The size B depends on both the number of UEs requiring service in the cell and the radio channel conditions of each of them. In addition, B is strongly related to the radio scheduling strategy (e.g, round robin, proportional fair, etc.). The number of UEs always varies from 1 to b_{max}, where this latter quantity is a function of the eNB's bandwidth. In LTE, b_{max} is reached, when users experiment bad radio conditions, i.e., when using a robust modulation as QPSK and a high degree of redundancy. For average radio conditions and non-saturated eNBs, it is more probable to have small-sized batches of UEs. The geometric distribution is intended to reflect the mix between radio conditions of UEs and their transmission needs.

With regard to the global Cloud-RAN architecture, the total amount of time t which is required to process BBU functions is given by $t = s + w$, where, s is the job's runtime and w is the waiting time of a job while there are no free processing units. The fronthaul delay between RRHs and the BBU-pool is then captured by the arrival distribution. See Figure 4.3 for an illustration.

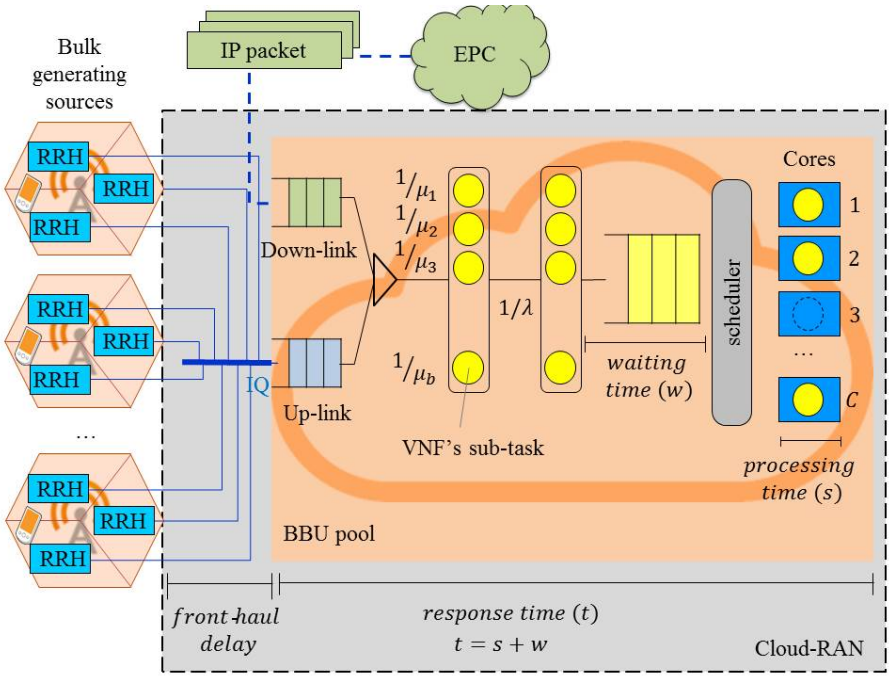


Figure 4.3: Stochastic service system for Cloud-RAN.

When assuming that the computing platform has a non-limited buffer, the stability of the system requires:

$$\rho = \frac{\lambda \mathbb{E}[B]}{\mu C} < 1. \tag{4.1.1}$$

In the following, we are interested in the sojourn time of subframes (batches) in the system, having in mind that if the sojourn time exceeds some threshold (i.e., ≈ 1 millisecond for encoding and ≈ 2 milliseconds for decoding) the subframe is lost. If we dimension the system so that

the probability for the sojourn time to exceed the threshold is small, we can then approximate the subframe loss rate by this probability. It is worth noting that in LTE, retransmissions and reception acknowledgments are handled per subframe by the HARQ process. When a TB is lost the whole subframe is resent.

4.1.3 Parallelism by CBs

In LTE, when a TB is too big, it is split into smaller data units, referred to as CBs. If we assume that the processing time of a CB is exponential with mean $1/\mu'$, we again obtain an $M^{[X]}/M/C$ model, where the batch size is the number of CBs in a TB. If this number is geometrically distributed, the service time of a TB is exponential, as supposed above. The key difference now is that individual CBs are processed in parallel by the C cores. The scheduler is able to allocate a core to each CB owing to the more atomic decomposition of subframes and TBs.

4.1.4 No parallelism

If the processing of TBs or CBs is not parallel, scheduling is based on subframes as presented in [108]. Still assuming a multi-core system, where subframes arrive according to a Poisson process, we are led to consider an $M/G/C$ queuing system. By making exponential assumptions for service times of CBs and TBs as well as supposing a geometric number of CBs per TB, we obtain an $M/M/C$ queue, which is well known in the queuing literature [93].

4.2 Batch model

From the analysis carried out in the previous section, the $M^{[X]}/M/C$ model can reasonably be used to evaluate the processing time of a subframe in a Cloud-RAN architecture based on a multi-core platform. While the sojourn time of an arbitrary job of a batch has been analyzed in [113], the sojourn time of a whole batch seems to have received less attention in the technical literature. In this section, we derive the Laplace transform of this last quantity; this eventually allows us to derive an asymptotic estimate of the probability of exceeding a large threshold.

Let us consider an $M^{[X]}/M/C$ queue with batches of size B arriving according to a Poisson process with rate λ. The service time of a job within a batch is exponential with mean $1/\mu$. We assume that the stability condition (4.1.1) holds so that a stationary regime exists. The number N of jobs in the system in the stationary regime is such that [113].

$$\phi(z) \stackrel{def}{=} \mathbb{E}\left(z^N\right) = \frac{\sum_{k=0}^{C-1}(C-k)p_k z^k}{C - \frac{\lambda}{\mu} z \left(\frac{1-B(z)}{1-z}\right)},$$

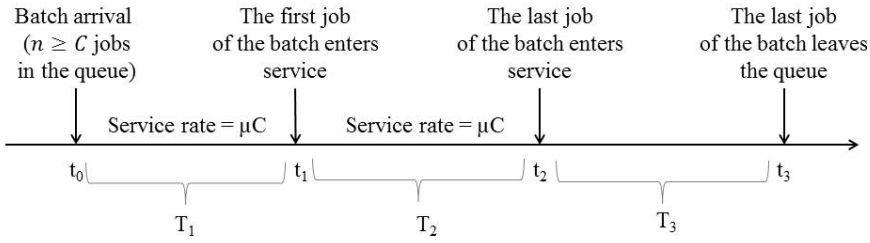
where $p_k = \mathbb{P}(N = k)$ and $B(z)$ is the generating function of the batch size B, i.e., $B(z) = \sum_{k=0}^{\infty} \mathbb{P}(B = k) z^k$. As explained in [113], the probabilities p_k for $k \geq 1$ satisfy the balance equations:

$$\begin{aligned}
p_1 &= \frac{\lambda}{\mu} p_0, \\
p_k &= \left(1 + \frac{\lambda - \mu}{k\mu}\right) p_{k-1} - \frac{\lambda}{\mu k} \sum_{\ell=0}^{k-2} p_\ell b_{k-1-\ell} && 2 \leq k \leq C, \\
p_k &= \left(1 + \frac{\lambda}{\mu C}\right) p_{k-1} - \frac{\lambda}{\mu C} \sum_{\ell=0}^{k-2} p_\ell b_{k-1-\ell} && k \geq C,
\end{aligned}$$

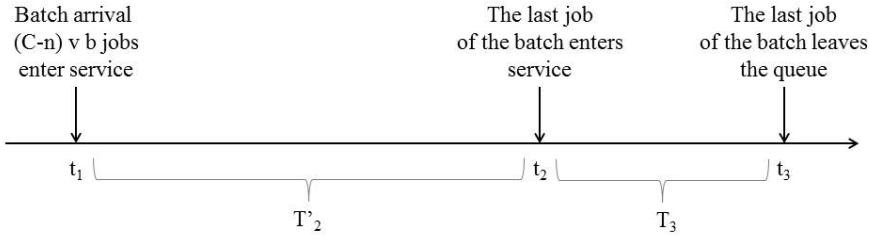
where b_ℓ is the probability that the batch size is equal to ℓ. We see in particular that the probabilities p_k for $k = 2, \ldots, C$ linearly depend on p_0, which can eventually be computed by using the normalizing condition $\sum_{k=0}^{C-1}(C-k)p_k = C(1-\rho)$.

We consider a batch of size b arriving at time t_0 and finding n jobs in the queue. We consider two cases (see Figure 4.4):

- **Case $n \geq C$:** In that case, the first job of the tagged batch has to wait before entering service.
- **Case $n < C$:** In that case, $b \wedge (C-n) \stackrel{def}{=} \min(b, C-n)$ jobs of the tagged batch immediately enter service; the $0 \vee (b+n-C) \stackrel{def}{=} \max(0, b+n-C)$ jobs have to wait before entering service.



(a) Case 1.



(b) Case 2.

Figure 4.4: Two cases upon the arrival of a batch.

4.2.1 Analysis of the first case

In the case $n \geq C$, the tagged batch will have to wait for a certain time before the first job enters service. Let t_1 denote the time at which the first job of the tagged batch begins its service. We obviously have that $T_1 = t_1 - t_0$ is equal to the sum of $n - C + 1$ independent random variables exponentially distributed with mean $1/(\mu C)$. The Laplace transform of T_1 is defined for $\Re(s) \geq 0$ by

$$\mathbb{E}_b\left(e^{-sT_1}\right) = \left(\frac{\mu C}{s+\mu C}\right)^{n-C+1},$$

where $\mathbb{E}_b$ is the expectation conditionally on the batch size b.

Let t_2 denote the time at which the last job of the batch enters its service. The difference $T_2 = t_2 - t_1$ is clearly the sum of $b-1$ independent exponential random variables with mean $1/(\mu C)$ (the quantity μC being the service rate of the system); the Laplace transform of this difference is

$$\mathbb{E}_b\left(e^{-sT_2}\right) = \left(\frac{\mu C}{s+\mu C}\right)^{b-1}.$$

To completely determine the sojourn time of the tagged batch, it is necessary to know the number y_b of jobs, which belong to this batch and which are in the queue when the last job of the batch begins its service. Let $t_1 = \tau_1 < \tau_2 < ... < \tau_b = t_2$ denote the service completion times of jobs (not necessarily belonging to the tagged batch) in the interval $[t_1, t_2]$. (Note that the point t_1 corresponding to the time at which the first job of the tagged batch enters service is itself a service completion time of one customer present in the queue upon the arrival of the tagged batch.) By definition τ_n is the time at which the n-th job of the tagged batch enters service.

Let us denote by y_n the number of jobs belonging to the tagged batch at time τ_n^+. Then, the sequence (y_n) is a Markov chain studied in Appendix F, where the conditional transition probabilities are expressed in terms of Stirling numbers of the second kind $S(n,k)$ [114] defined for $0 \leq k \leq n$ by

$$S(n,k) = \sum_{j=0}^{k} \frac{(-1)^{k-j}}{(k-j)!j!} j^n. \tag{4.2.1}$$

Stirling numbers are such that $S(n,n) = 1$ for $n \geq 0$, $S(n,1) = 1$ and $S(n,0) = 0$ for $n \geq 1$, and satisfy the recursion for $n \geq 0$ and $k \geq 1$

$$S(n+1,k) = kS(n,k) + S(n,k-1).$$

To formulate the results we alternatively use the polynomials $\mathcal{A}_{n,k}(x)$ defined by means of Stirling numbers as follows:

$$\mathcal{A}_{n,p}(x) = p! \sum_{j=0}^{n} \binom{n}{j} S(j,p) x^{n-j}. \tag{4.2.2}$$

The polynomials $\mathcal{A}_{n,p}(x)$ satisfy the recursion for $n, p \geq 0$

$$\mathcal{A}_{n,p}(x) = (x+p)\mathcal{A}_{n-1,p}(x) + p\mathcal{A}_{n-1,p-1}(x)$$

and $\mathcal{A}_{n,p}(0) = p!S(n,p)$.

With the above notation, when the b-th job of the tagged batch enters service, there are y_b jobs of this batch in the queue. The time T_3 to serve these jobs is

$$T_3 = \mathcal{E}(y_b\mu) + \mathcal{E}((y_b-1)\mu) + \ldots + \mathcal{E}(\mu),$$

where $\mathcal{E}(k\mu)$ for $k = 1, \ldots, y_b$ are independent random variables with mean $1/(k\mu)$. The Laplace transform of T_3 knowing y_b is

$$\mathbb{E}_b\left(e^{-sT_3} \mid y_b = k\right) = \frac{k!}{\prod_{\ell=1}^{k}\left(\frac{s}{\mu}+\ell\right)} = \frac{k!}{\left(\frac{s}{\mu}+1\right)_k}, \tag{4.2.3}$$

where $(x)_k$ is the Pochammer symbol (a.k.a. rising factorial) defined by $(x)_k = x(x+1)\ldots(x+k-1)$. By using Lemma F.0.1, it follows that the Laplace transform of the sojourn time T of a batch of size b in the system when there are $n \geq C$ customers in the queue upon arrival is

$$\mathbb{E}_b\left(e^{-sT} \mid N = n \geq C\right) = \frac{C!}{C^b}\left(\frac{\mu C}{s+\mu C}\right)^{n+b-C} \sum_{k=0}^{C} \frac{S(b,k)}{(C-k)!} \frac{k!}{\left(\frac{s}{\mu}+1\right)_k}, \tag{4.2.4}$$

which can be rewritten by using the polynomials $\mathcal{A}_{n,p}(x)$ defined by Equation (4.2.2) as

$$\mathbb{E}_b\left(e^{-sT} \mid N = n \geq C\right) = \frac{1}{C^{b-1}}\left(\frac{\mu C}{s+\mu C}\right)^{n+b-C} \sum_{k=0}^{C} \binom{C-1}{k-1} \mathcal{A}_{b,k-1}(1) \frac{1}{\left(\frac{s}{\mu}+1\right)_k}, \quad (4.2.5)$$

4.2.2 Analysis of the second case

When the number n of jobs in the queue is less than C upon the arrival of the tagged batch of size b, then $b \wedge (C-n)$ customers immediately begin their service. Let us first assume that $b+n > C$. Taking the tagged batch arrival as time origin, the last job of the tagged batch enters service at random time T_2' with Laplace transform

$$\mathbb{E}(e^{-sT_2'}) = \left(\frac{\mu C}{s+\mu C}\right)^{n+b-C}.$$

The number of jobs of the tagged batch present in the system when the last job enters service is Y_n such that

$$\mathbb{P}(Y_n = k) = \mathbb{P}(y_{b+n-C} = k \mid y_1 = C-n) = \frac{1}{C^{n+b-C-1}} \binom{n}{k+n-C} \mathcal{A}_{n+b-C-1,k+n-C}(C-n).$$

by using Equation (F.0.2). For a given value $Y_n = k$, the time T_3 needed to serve all jobs of the tagged batch has Laplace transform given by Equation (4.2.3). By using Lemma F.0.1, we conclude that under the assumption $n < C$ and $b+n > C$, the sojourn time T of the tagged batch has Laplace transform

$$\mathbb{E}_b\left(e^{-sT} \mid N = n, b+N > C, N < C\right) = \left(\frac{\mu C}{z+\mu C}\right)^{n+b-C} \sum_{k=C-n}^{C} \mathbb{P}(Y_n = k) \frac{k!}{\left(\frac{s}{\mu}+1\right)_k} \quad (4.2.6)$$

and hence

$$\mathbb{E}_b\left(e^{-sT} \mid N = n < C, b+N > C\right) = \left(\frac{\mu C}{z+\mu C}\right)^{n+b-C} \tau(n,b;s), \quad (4.2.7)$$

where

$$\tau(n,b,;s) = \frac{1}{C^{n+b-C-1}} \sum_{k=C-n}^{C} \binom{n}{k+n-C} \mathcal{A}_{n+b-C-1,k+n-C}(C-n) \frac{k!}{\left(\frac{s}{\mu}+1\right)_k}. \quad (4.2.8)$$

When $b+n \leq C$, all jobs of the tagged batch enter service just after arrival and the Laplace transform of the sojourn time is

$$\mathbb{E}_b\left(e^{-sT} \mid N = n, b+n \leq C\right) = \frac{b!}{\left(\frac{s}{\mu}+1\right)_b}. \quad (4.2.9)$$

4.2.3 Main result

By using the results of the previous sections, we determine the Laplace transform $\Phi(s) = \mathbb{E}(e^{-sT})$ of the sojourn time of a batch in the $M^{[X]}/M/C$ queue.

Theorem 4.2.1. *The Laplace transform $\Phi(s)$ is given by*

$$\Phi(s) = \beta(s)\left(\phi\left(\frac{\mu C}{s+\mu C}\right) - \phi_C\left(\frac{\mu C}{s+\mu C}\right)\right) + \mathbb{E}\left(\frac{B!}{\left(\frac{s}{\mu}+1\right)_B}\mathbb{P}(N \leq C-B)\right) + \sum_{n=0}^{C-1} p_n \mathbb{E}\left(\tau(n,B;s)\left(\frac{\mu C}{s+\mu C}\right)^{n+B-C}\right), \quad (4.2.10)$$

where

$$\begin{aligned} \beta(s) &= \mathbb{E}\left(\frac{1}{C^{B-1}}\left(\frac{\mu C}{s+\mu C}\right)^{B-C}\sum_{k=0}^{C}\binom{C-1}{k-1}\frac{\mathcal{A}_{B,k-1}(1)}{\left(\frac{s}{\mu}+1\right)_k}\right), \quad (4.2.11)\\ \phi_C(z) &= \sum_{n=0}^{C-1} p_n z^n, \end{aligned}$$

and $\tau(n,b;s)$ defined by Equation (4.2.8).

Proof. By conditioning on the batch size b, we have from the two previous sections

$$\mathbb{E}_b(e^{-sT}) = \beta_b(s)\sum_{n=C}^{\infty} p_n\left(\frac{\mu C}{s+\mu C}\right)^n + \sum_{n=0}^{C-1} p_n \tau(n,b;s)\left(\frac{\mu C}{s+\mu C}\right)^{n+b-C} + \frac{b!}{\left(\frac{s}{\mu}+1\right)_b}\mathbb{P}(N \leq C-b)$$

with

$$\beta_b(s) = \frac{C!}{C^b}\left(\frac{\mu C}{s+\mu C}\right)^{b-C}\sum_{k=0}^{C}\frac{S(b,k)}{(C-k)!}\frac{k!}{\left(\frac{s}{\mu}+1\right)_k}$$

and $\tau(n,b;s)$ is defined by Equation (4.2.8). Note that we use the fact that $\tau(n,b;s) = 0$ if $b < C-n$ in the above equation. By deconditioning on the batch size, Equation (4.2.10) follows. □

Following [113], let us define z_1 the root with the smallest module to the equation

$$V(z) \stackrel{def}{=} C - \frac{\lambda}{\mu} z\left(\frac{1-B(z)}{1-z}\right) = 0;$$

the root z_1 is real and greater than 1. The negative real number

$$s_1 = -\mu C\left(1 - \frac{1}{z_1}\right)$$

is the singularity with the smallest module of the Laplace transform $\Phi(s)$ if $s_1 > -\mu$ (namely, $z_1 < \frac{C}{C-1}$ or $V\left(\frac{C}{C-1}\right) > 0$).

Corollary 4.2.1. *If $s_1 > -\mu$, then when t tends to infinity*

$$\mathbb{P}(T > t) \sim \frac{\mu C U(z_1)\beta(s_1)}{s_1 z_1^2 V'(z_1)} e^{s_1 t}, \quad (4.2.12)$$

where $U(z) = \sum_{k=0}^{C-1}(C-k)p_k z^k$. If $s_1 < -\mu$, then the tail of the distribution of T is such that

when t tends to infinity

$$\mathbb{P}(T > t) \sim \kappa e^{-\mu t}, \tag{4.2.13}$$

where

$$\kappa = \mathbb{E}(B\mathbb{P}(N + B \leq C)) + C\mathbb{E}\left(\left(\frac{C}{C-1}\right)^B - 1\right)\sum_{n=0}^{\infty} p_{C+n}\left(\frac{C}{C-1}\right)^n + \sum_{n=0}^{C-1} p_n C\mathbb{E}\left(\mathbb{1}_{\{B>C-n\}}\left(\left(\frac{C}{C-1}\right)^{n+B-C} - \frac{n}{C-1}\right)\right).$$

Proof. When $s_1 > -\mu$, the root with the smallest module of the Laplace transform $\Phi(s)$ is s_1 and the estimate (4.2.12) immediately follows by using standard results for Laplace transforms [115].

When $s_1 < -\mu$, the root with the smallest module is $-\mu$. We have for s is the neighborhood of $-\mu$,

$$\beta(s) \sim \frac{\mu}{\mu+s}\mathbb{E}\left(\frac{1}{C^{B-1}}\left(\frac{\mu C}{C-1}\right)^{B-C}\sum_{k=0}^{C}\binom{C-1}{k-1}k\mathcal{A}_{B,k-1}(1)\right) = \frac{\mu}{\mu+s}C\mathbb{E}\left(\left(\frac{C}{C-1}\right)^B - 1\right)\sum_{n=0}^{\infty} p_{C+n}\left(\frac{C}{C-1}\right)^n,$$

where we have used Equation (F.0.5) for $\ell = 1$. In addition, under the same conditions,

$$\mathbb{E}\left(\frac{B!}{\left(\frac{s}{\mu}+1\right)_B}\mathbb{P}(N \leq C - B)\right) \sim \frac{\mu}{\mu+s}\mathbb{E}(B\mathbb{P}(N + B \leq C))$$

and

$$\sum_{n=0}^{C-1} p_n\mathbb{E}\left(\tau(n,B;s)\left(\frac{\mu C}{s+\mu C}\right)^{n+B-C}\right) \sim \frac{\mu}{\mu+s}\sum_{n=0}^{C-1} p_n\mathbb{E}\left(\left(\frac{C}{C-1}\right)^{n+B-C}\frac{1}{C^{n+B-C-1}}\sum_{k=C-n}^{C}\binom{n}{k+n-C}k\mathcal{A}_{n+B-C-1,k+n-C}(C-n)\right) = \frac{\mu}{\mu+s}\sum_{n=0}^{C-1} p_n C\mathbb{E}\left(\mathbb{1}_{\{B>C-n\}}\left(\left(\frac{C}{C-1}\right)^{n+B-C} - \frac{n}{C-1}\right)\right),$$

where we have used Equation (F.0.5) for $\ell = C - n$. Gathering the above residue calculations yields Equation (4.2.13). □

Corollary 4.2.1 states that when the service capacity of the system is sufficiently large, the tail of the sojourn time of a batch is dominated by the service time of a single job. It is also worth noting that contrary to what is stated in [113], the same result holds for the decay rate of the sojourn time of a job in the system. Finally, when C is large and for moderate values of the load and the mean batch size, $\kappa \sim \mathbb{E}(B\mathbb{P}(N + B \leq C)) \sim \mathbb{E}(B)$. This means that there is roughly a multiplicative factor $\mathbb{E}(B)$ between the tail of the sojourn time of a batch and that of a job.

When the batch size is geometrically distributed with mean $1/(1-q)$, (i.e., $\mathbb{P}(B = k) =$

$(1-q)q^{k-1}$), we have $s_1 = -(1-q)\mu C(1-\rho)$ and

$$z_1 = \frac{C}{qC + \frac{\lambda}{\mu}} > 1 \text{ for } C > \frac{\lambda}{(1-q)\mu}. \tag{4.2.14}$$

We have $z_1 < \frac{C}{C-1}$ if and only if $\rho > 1 - \frac{1}{C(1-q)}$.

4.3 Numerical experiments

In this section, we evaluate by simulation the behavior of a Cloud-RAN system hosting the base band processing of a hundred base-stations. The goal is to test the relevance of the $M^{[X]}/M/C$ model for sizing purposes and to derive dimensioning rules. C-RAN sizing refers to determining the minimum number of servers (cores), which are required to ensure the processing of LTE subframes within deadlines for a given number of base stations (eNBs), as well as the maximum fronthaul distance between antennas and the BBU-pool.

In LTE, deadlines are applied to the whole subframe. For instance, when the runtime of the base-band processing of a subframe in the uplink direction exceeds 2 milliseconds, the whole subframe is lost and therefore retransmitted. In order to bring new perspectives for the radio channel efficiency, we also evaluate the loss of single users, so that RAN systems might hold less redundant data. The loss of subframes and UEs are captured in the $M^{[X]}/M/C$ model by the impatience of batches and customers, respectively.

4.3.1 Simulation settings

We evaluate a C-RAN system hosting 100 eNBs where each of them has a bandwidth of 20 MHz. All eNBs have a single antenna (i.e., work under SISO configuration) and use FDD transmission mode. Antennas (eNBs) are distributed around the computing center within a 100 km radius.

In the following, we focus our analysis on the decoding and encoding functions carried out during uplink and downlink processing, respectively, due to their non-deterministic behavior, as well as, because they are the greatest computing resource consumer of all BBU functions [80,86]. To assess the runtime of decoding and encoding functions, we use OAI's code, which implements RAN functions in open-source software [86].

4.3.2 Model analysis

In order to represent the behavior of a Cloud-RAN system by using the $M^{[X]}/M/C$ model, we feed the queuing system with statistical parameters captured from the C-RAN emulation during the busy-hour; see Figure 4.5. We capture the behavior of the decoding function in a multi-core system performing parallelism by UEs. The obtained parameters are as follows:

- The mean service time of decoding jobs, $\mathbb{E}[S]$, is equal to 281 microseconds. Each decoding job corresponds to the data of a single UE.
- The mean number of decoding jobs requiring service at the same time, i.e, the mean batch size, is given by $\mathbb{E}[B] = 5$. The number of UEs scheduled per subframe can vary between 1 and 16 for an eNB of 20 MHz. This can be approximated by a geometric distribution with parameter $q = 0.8$ ($q = 1 - \frac{1}{\mathbb{E}[B]}$). Batch-sizes are in the interval $[1, 16]$ with probability equal to 0.97. Figure 4.5 gives the percentage of each of the subframe types in the system.
- The mean inter-arrival time of batches is 10 microseconds. Each eNB generates a bulk of decoding jobs (subframe) every millisecond. The mean inter-arrival time is computed by dividing the periodicity of subframes by the number of eNBs.

- The time-budget (deadline) for the uplink processing is given by $\delta = 2000$ microseconds.

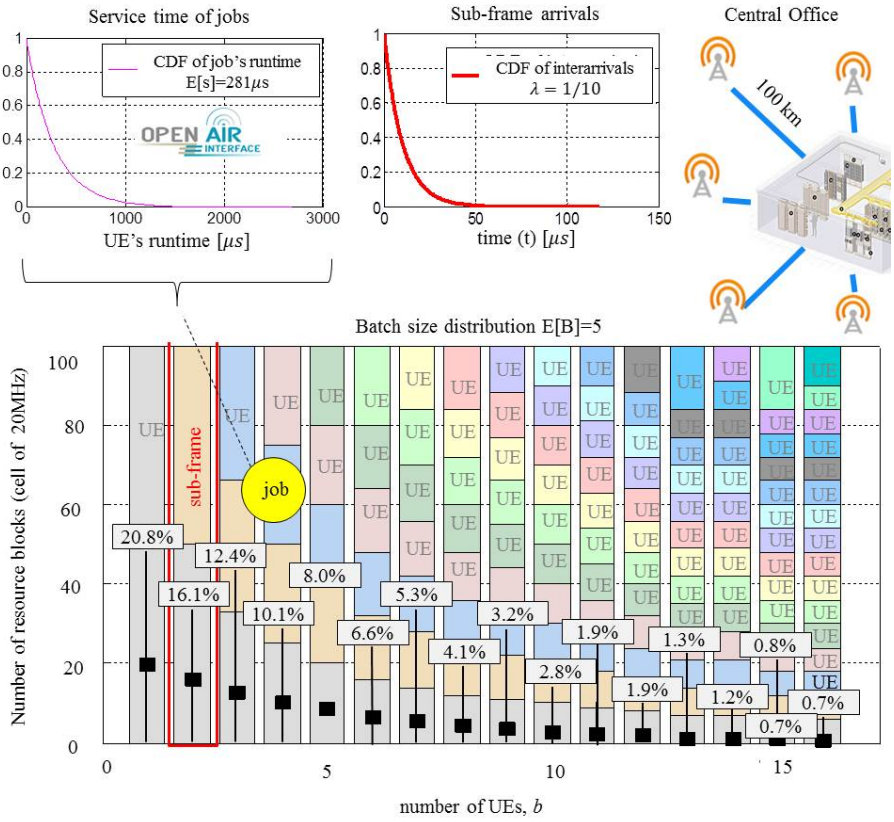


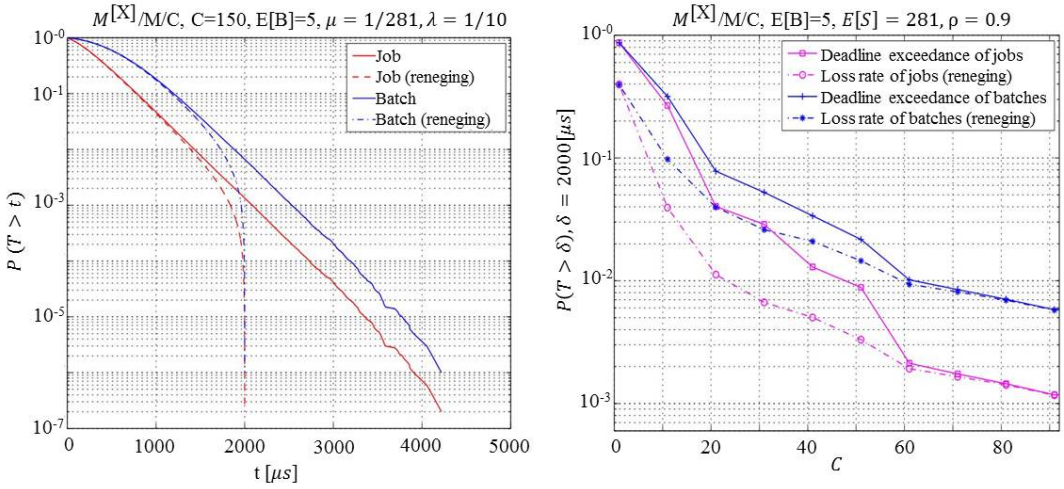
Figure 4.5: Statistical parameters of Cloud-RAN.

We can then evaluate the $M^{[X]}/M/C$ model with the following parameters: $\mu = 1/281$ and $\lambda = 1/10$. By Equation (4.1.1) for $C = 150$, the load is $\rho = 0.9367$. The CDFs of the sojourn time of jobs and batches are shown in Figure 4.6(a). By using Corollary 4.2.1, we verify that if D is the sojourn time of a job in the $M^{[X]}/M/C$ queue, then $\mathbb{P}(T > t)/\mathbb{P}(D > t)$ tends to a constant when $t \to \infty$. It can also be checked that the slopes of the curves $-\log(P(D > t))/t$ and $-\log(P(T > t))/t$ for large t are both equal to μ.

In practice, aborting the execution of subframes, which overtake deadlines, is highly desirable to save computing resources. We are then interested in the behavior of the $M^{[X]}/M/C$ with reneging of both customers and batches. A job (customer) leaves the system (even during service) when its sojourn time reaches a given deadline δ. In the case of reneging of batches, the sojourn time of a batch is calculated from the arrival until the instant at which the last job composing the batch is served. Results with impatient customers and batches are depicted in Figure 4.6(a).

With impatience, the loss rate of jobs and batches, is respectively 0.0013 and 0.0065. We observe that the gap between the two rates (i.e., 0.0065/0.0013) is close to the mean batch size $\mathbb{E}[B]$. This is true when loss rates are at least of order 10^{-3}.

Due to the complexity of the theoretical analysis of impatience-based models, we choose to use the performance of an $M^{[X]}/M/C$ system without reneging for sizing a Cloud-RAN infrastructure. Since this model stochastically dominates the system with reneging, we obtain conservative bounds. As illustrated in Figure 4.6(b), we verify for both jobs and batches that the probability of deadline exceedance is always greater in a system without reneging, and moreover,



(a) sojourn time of jobs and batches

(b) deadline exceedance of jobs and batches

Figure 4.6: $M^{[x]}/M/C$ behavior.

these two probabilities are close to each other when C increases.

4.3.3 Cloud-RAN dimensioning

The final goal of Cloud-RAN sizing is to determine the amount of computing resources needed in the cloud (or a data center) to guarantee the base-band processing of a given number of eNBs within deadlines. For this purpose, we evaluate the $M^{[X]}/M/C$ model (without reneging) while increasing C, until an acceptable probability of deadline exceedance (say, ε). The required number of cores is then the first value that achieves $P(T > \delta) < \varepsilon$.

We validate by simulation the effectiveness of the $M^{[X]}/M/C$ model with the behavior of the real C-RAN system during the reception process (uplink) of LTE subframes. See Figure 4.7 for an illustration. Results show that for a given $\varepsilon = 0.00615$, the required number of cores is $C_r = 151$, which is in accordance with the real C-RAN performance, where the probability of deadline exceedance is barely 0.00018.

When C takes values lower than a certain threshold C_s, the C-RAN system is overloaded, i.e., the number of cores is not sufficient to process the vBBUs' workload; the system is then unstable. The threshold C_s can be easily obtained from Equation (4.1.1); for $\rho = 1$, $C_s = \lceil \lambda * E[B]/\mu \rceil = 141$ cores.

4.3.4 Performance evaluation

We are now interested in the performance of the whole Cloud-RAN system running in a data center equipped with 151 cores. The system processes both uplink and downlink subframes belonging to 100 eNBs. Results show an important gain when performing parallelism per CBs during both reception (see Figure 4.8(a)) and transmission (see Figure 4.8(b)) processing.

It is observed that more than 99% of subframes are processed within 472 microseconds and 1490 microseconds when performing parallelism by CBs and UEs, respectively. It represents a gain of 1130 microseconds (CB) and 100 microseconds (UE) with respect to the original system (non-parallelism). These gains in the sojourn time enable the operator to increase the maximum distance between antennas and the central office. Hence, when considering the light-speed in the

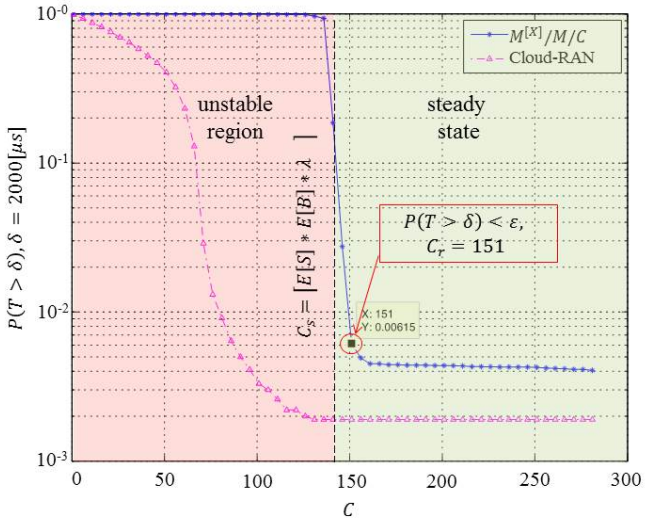


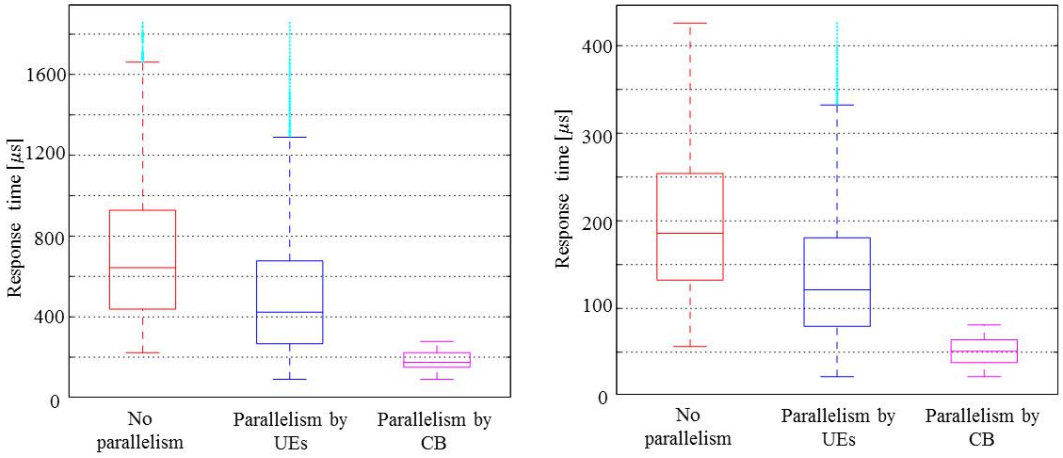
Figure 4.7: C-RAN sizing when using the $M^{[X]}/M/C$ model.

optic-fiber, i.e., $2.25 * 10^8$m/s, the distance can be increased up to ≈ 250 km when running CBs in parallel. Figure 4.9 shows the CDF of the sojourn time of LTE subframes when performing parallel programing.

4.3.5 Analysis of results

We have studied in this chapter the performance of virtualized base-band functions when using parallel processing. We have concretely evaluated the processing time of LTE subframes in a C-RAN system. In order to reduce the latency, we have investigated the functional and data decomposition of BBU functions, which leads to batch arrivals of parallel runnable jobs with non-deterministic runtime. For assessing the required processing capacity to support a C-RAN system, we have introduced a bulk arrival queuing model, namely the $M^{[X]}/M/C$ queuing system, where the batch size follows a geometric distribution. The variability of the fronthaul delay and jobs' runtime are captured by the arrival and service distributions, respectively. Since the runtime of a radio subframe becomes the batch sojourn-time, we have derived the Laplace transform of this latter quantity as well as the probability of exceeding certain threshold to respect LTE deadlines.

We have validated the model by simulation, when performing a C-RAN system with one hundred eNBs of 20 MHz during the busy-hour. We have additionally illustrated that the impatience criterion reflecting LTE time-budgets is not incident when the probability of deadline exceedance is low enough. Finally, once the C-RAN system is dimensioned, we have evaluated its performance when processing both uplink and downlink subframes. Results show an important gain in terms of latency when performing parallel processing of LTE subframes and the approach based on the batch model to sizing C-RAN systems.



(a) uplink (Rx) (b) downlink (Tx)

Figure 4.8: Cloud-RAN performance, 100 eNBs, $C = 151$.

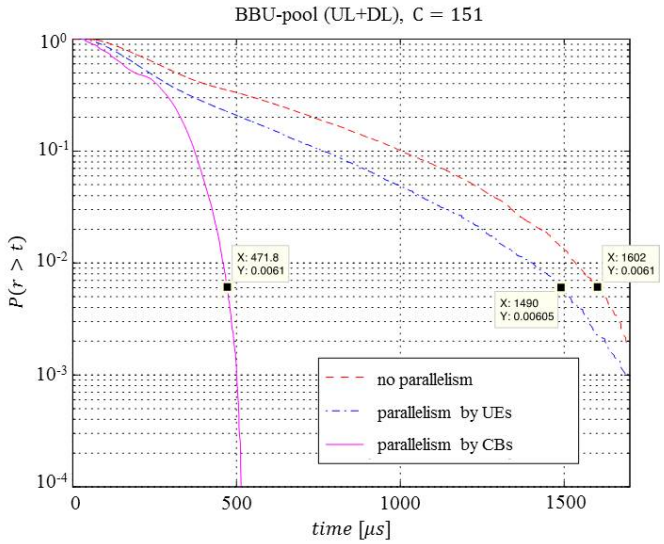


Figure 4.9: CDF of the sojourn time of radio subframes.

Chapter 5

Proof of Concept: C-RAN acceleration

Contents

5.1 Test-bed description 81
5.2 Implementation outline 83
5.3 Performance evaluation 87

We present in this chapter, the implementation of the proposed theoretical models in a multi-core platform in order to validate them. We describe the implementation methodologies used for performing a thread-pool in the aim of dealing with the various parallel runnable jobs. We describe both the queuing principles and the scheduling strategy when the number of parallel runnable jobs exceeds the number of available free cores.

A thoughtful performance analysis is carried out for determining the gain that can be obtained when scheduling channel coding jobs per both UEs and CBs.

5.1 Test-bed description

On the basis of various open-source solutions we implemented an end-to-end virtualized mobile network which notably includes a virtualized RAN. In this test-bed, both the core and the access network are based on OAI code. KVM and Openstack are used as virtualization environments. Physical servers are connected via Intranet, notably one hosting the software-based *eNB* (access network) and two named *UGW-C* and *UGW-U* implementing the control and user plane of the core network, respectively. The Radio element is performed by an USRP B210 card by means of which commercial smartphones can be connected.

As shown in Figure 5.1 the virtualized core network is based on a complete separation of the user and control planes as recommended by 3GPP for 5G networks and referred to as CUPS (Control User Plane Separation), implemented in b<>com's solution [116]. When a UE attaches to the network, the AAA (Authentication, Authorization, and Accounting) procedure is triggered by the MME. User profiles are validated by the HSS data base which stores various parameters such as sim-card information (OP, key, MNC, MCC), apn, imei, among others. When access is granted to the UE, the DHCP component provides it the IP-address, which is taken out of the address-pool established in the AAA component. The end-to-end connection is assured after the creation of the GTP-U and GTP-C tunnels. The NAT component provides address translation and is deployed between the SGi interface and the Internet network.

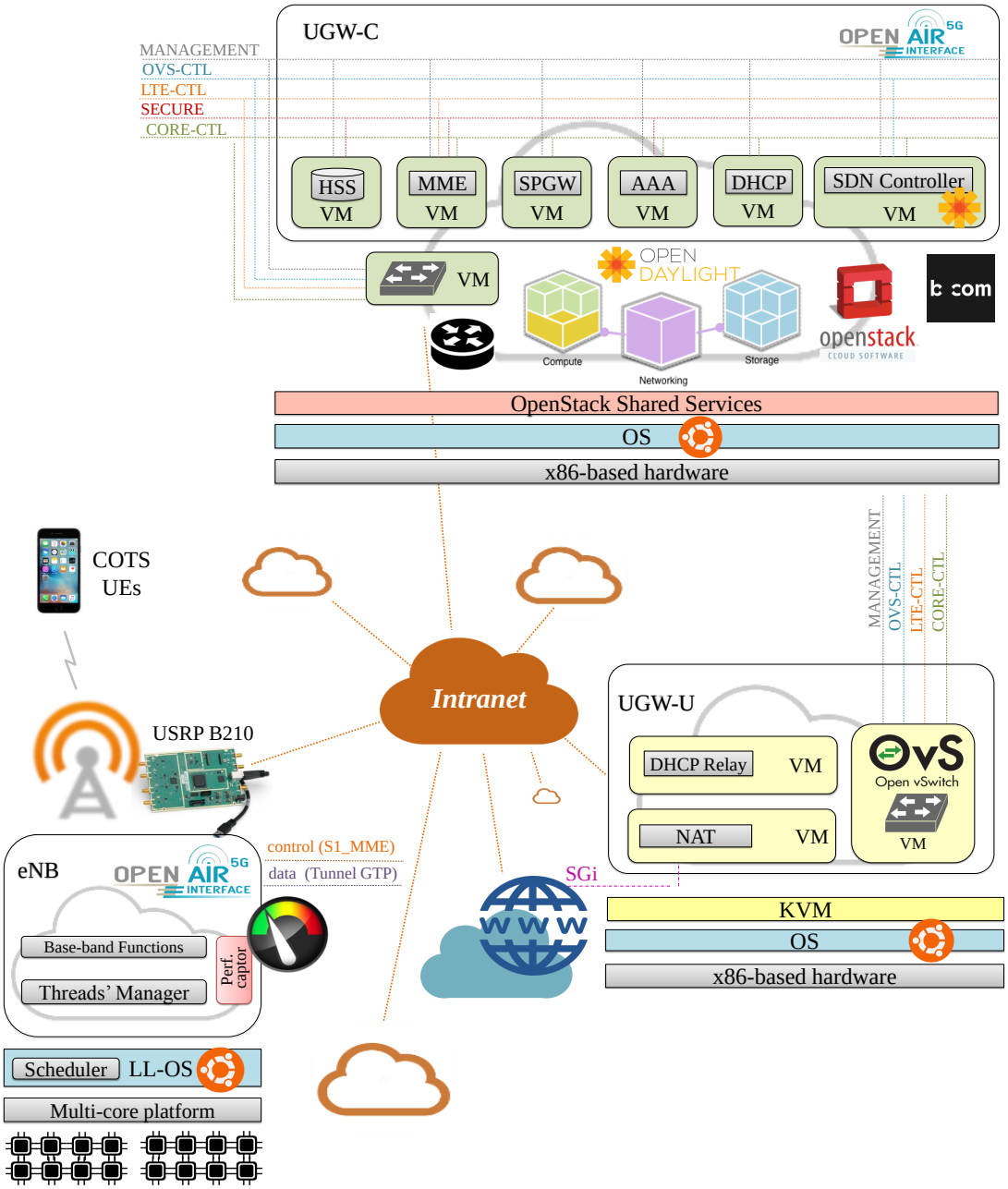


Figure 5.1: Test-bed architecture.

The platform implements the proposed models and scheduling strategies which perform the parallel processing of the most expensive virtual RAN function in terms of latency, namely, the channel coding function. The parallel processing of both encoding (downlink) and decoding (uplink) functions is carried out by using multi-threading in a multi-core server. The workload of threads is managed by a global non-preemptive scheduler (Thread's manager), i.e., a thread is assigned to a dedicated single core with real-time OS priority and is executed until completion without interruption. The isolation of threads is provided by a specific configuration performed in the OS which prevents the use of channel coding computing resources for any other job.

5.2 Implementation outline

We specifically perform massive parallelization of channel encoding and decoding processes. These functions are detailed here below, before presenting the multi-threading mechanism and the scheduling algorithm.

5.2.1 Encoding function

The encoder (See Figure for an illustration) consists of 2 Recursive Systematic Convolutional (RSC) codes separated by an inter-leaver. Before encoding, data (i.e., a subframe) is conditioned and segmented in code blocks of size T which can be encoded in parallel. When the multi-threading model is not implemented CBs are executed in series under a FIFO discipline. Thus, an incoming data block b_i is twice encoded, where the second encoder is preceded of the permutation procedure (inter-leaver). The encoded block (b_i, b_i', b_i'') of size $3T$ constitutes the information to be transmitted in the downlink direction. Hence, for each information bit two parity bits are added, i.e., the resulting code rate is given by $r = 1/3$. With the aim of reducing the channel coding overhead, a puncturing procedure may be activated for periodically deleting bits. A multiplexer is finally employed to form the encoded block x_i to be transmitted. The multiplexer is nothing but a parallel to serial converter which concatenates the systematic output b_i, and both recursive convolutional encoded output sequences, b_i', and b_i''.

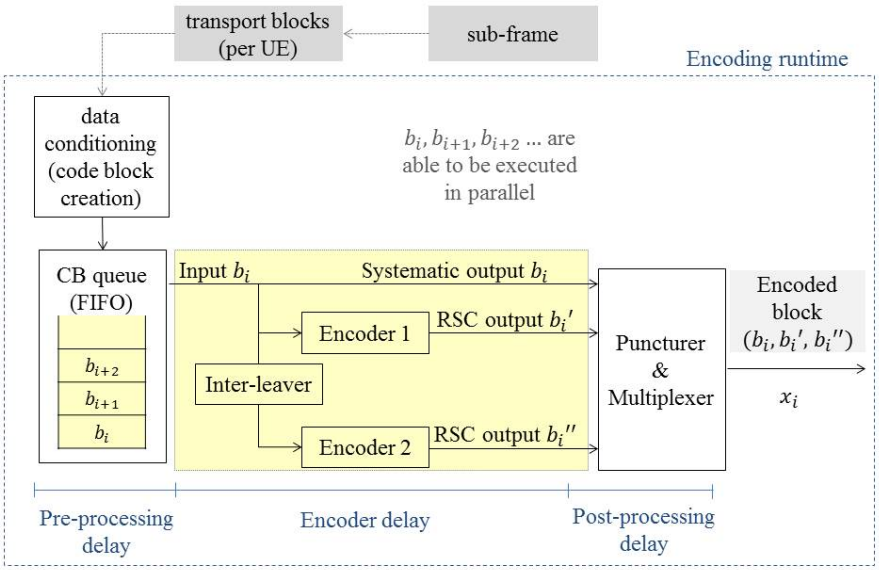


Figure 5.2: Block diagram of encoding function.

5.2.2 Decoding function

Unlike encoding, the decoding function is iterative and works with soft bits (real and not binary values). Real values represent the Log-Likelihood Ratio (LLR), i.e., the radio of the probability that a particular bit was 1 and the probability that the same bit was 0. (Log is used for better precision).

The decoding function occurs as follows: received data $R(x_i)$ is firstly de-multiplexed in $R(b_i)$, $R(b_i)'$, and $R(b_i)''$ which respectively correspond to the systematic information bits of i-th code block, b_i, and to the received parity bits, b_i' and b_i''.

$R(b_i)$ and $R(b_i)'$ feed the first decoder which calculates the LLR (namely, extrinsic information) and passes it to the second decoder. The second decoder uses that value to calculate LLR and feeds back it to the first decoder after a de-interleaved process. Hence, the second decoder has three inputs, the extrinsic information (reliability value) from the first decoder, the interleaved received systematic information $R(b_i)$, and the received values parity bits $R(b_i)''$. See Figure for an illustration.

The decoding procedure iterates until either the final solution is obtained or the allowed maximum number of iterations is reached. At termination, the hard decision (i.e., 0 or 1 decision) is taken to obtain the decoded data block, $\widehat{x_i}$. The data block is either successfully decoded or is not. The stopping criterion corresponds to the average mutual information of LLR, if it converges the decoding process may terminate earlier. Note that, there is a trade-off between the runtime (i.e., number of iterations) and the successful decoding of a data block.

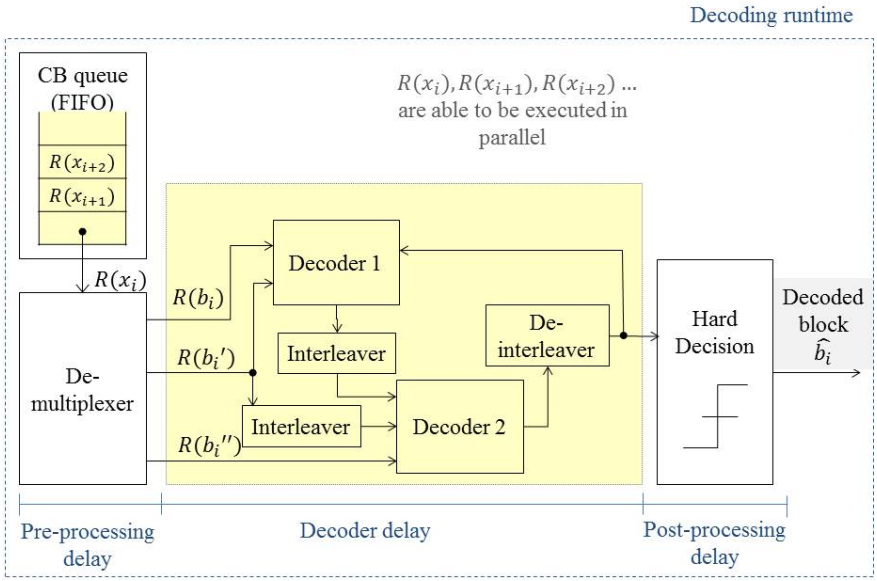


Figure 5.3: Block diagram of decoding function.

5.2.3 Thread-pool

On the basis of massive parallel programming we propose splitting the channel encoding and decoding function in multiple parallel runnable jobs. The main goal is improving their performance in terms of latency.

In order to deal with the various parallel runnable jobs, we implement a thread-pool, i.e., a multi-threading environment. A dedicated core is affected to each thread during the channel

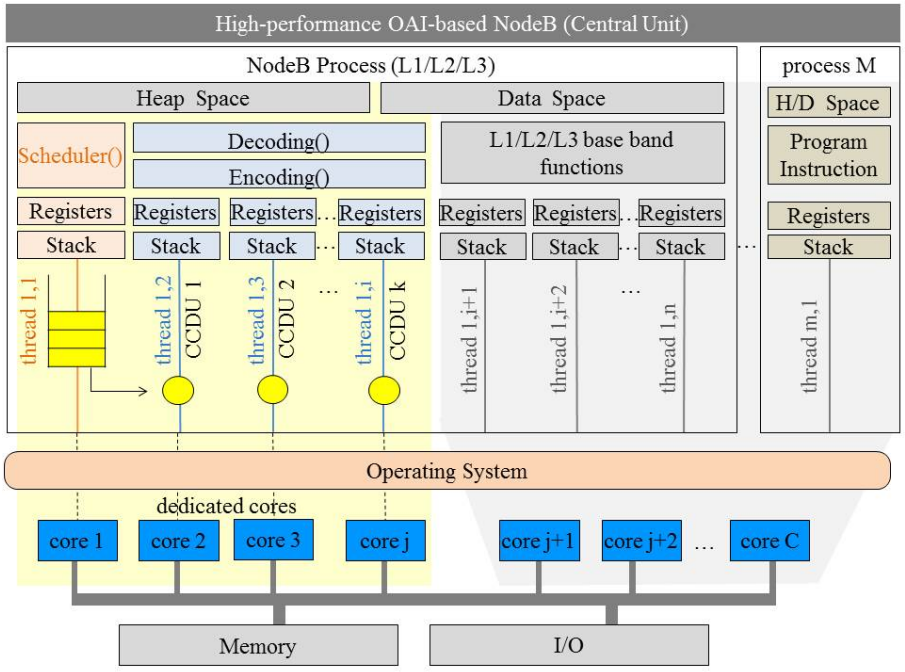


Figure 5.4: Multi-threading implementation.

coding processing. When the number of runnable jobs exceeds the number of free threads, jobs are queued.

In the aim of reaching low latency, we implement multi-threading within a single process instead of multitasking across different processes (namely, multi-programming). In a real-time system creating a new process on the fly becomes extremely expensive because all data structures must be allocated and initialized. In addition, in a multi-programing Inter process communications (IPCs) go through the OS which produces system calls and context switching overhead.

When using a multi-threading (namely, POSIX [117]) process for running encoding and decoding functions, other processes cannot access resources (namely, data space, heap space, program instructions) which are reserved for channel coding processing.

The memory space is shared among all threads belonging to the channel coding process which enables latency reduction. Each thread performs the whole encoding or decoding flow of a single Channel Coding Data Unit (CCDU). We define a CCDU as the suite of bits which corresponds to a radio subframe (no-parallelism), a TB or even a CB. When performing parallelism, CCDUs arrives in batches every millisecond. These data units are appended to a single queue (see Algorithm 1) which is managed by a global scheduler. We use non-preemptive scheduling, i.e., a thread (CCDU) is assigned to a dedicated single core with real-time OS priority and is executed until completion without interruption.

Threads' isolation is not provided by the POSIX API, thus, an specific configuration have been performed in the OS to prevent the use of channel coding computing resources for any other jobs. The global scheduler (i.e., the thread's manager) runs itself within a dedicated thread and performs a FIFO discipline for allocating cores to Channel Coding (CC) jobs which are waiting in the queue to be processed. Figure 5.4 illustrates j cores dedicated to channel coding processing, remaining $C - j$ cores are shared among all processes running in the system, including those belonging to the upper-layers of the eNB.

Algorithm 1 Queuing channel coding jobs

```
1: CB_MAX_SIZE ← 6120
2: procedure QUEUING
3:     while subframe_buffer ≠ ∅ do
4:         SF ← get subframe
5:         nUE ← get UE's number
6:         while nUE > 0 do
7:             CCDU_UE ← get_TB(nUE-th,SF)
8:             if CB_parallelism_flag = true then
9:                 while CCDU_UE ≥ CB_MAX_SIZE do
10:                    CCDU_CB ← get_CB(nCB-th,CCDU_UE)
11:                    queue ← append(CCDU_CB)
12:                end while
13:            else
14:                queue ← append(CCDU_UE)
15:            end if
16:            nUE ← nUE-1
17:        end while
18:    end while
19: end procedure
```

5.2.4 Queuing principles

The CCDU's queue is a chained list containing the pointers to the first and last element, the current number of CCDUs in the queue and the mutex (namely, mutual exclusion) signals for managing shared memory. The mutex mechanism is used to synchronize access to memory space in case of more than one thread requires writing at the same time. In order to reduce waiting times, we perform data context isolation per channel coding operation, i.e., dedicated CC threads do not access any global variable of the gNB (referred to as 'soft-modem' in OAI).

Scheduler

The scheduler takes from the queue the next CCDU to be processed, and update the counter of jobs (i.e., decrements the counter of remaining CCDUs to be processed). The next free core executes the first job in the queue.

In case of decoding failure, the scheduler purges all CCDUs belonging to the same UE (TB). In fact a TB can be successfully decoded only when all CBs have been individually decoded. (See Algorithm 2)

Channel coding variables are embedded in a permanent data structure to create an isolated context per channel coding operation, in this way, CC threads do not access any memory variable in the main soft-modem (eNB). The data context is passed between threads by pointers.

5.2.5 Performance captor

In order to evaluate the multi-threading performance, we have implemented a 'performance captor' which gets key timestamps during the channel coding processing for uplink and downlink directions. With the aim of minimizing measurements overhead, data is collected in a separate process, so-called 'measurements collector' which works out of the real-time domain.

The data transfer between both separate processes, i.e., the 'performance captor' and the 'measurements collector' is performed via an OS-based pipe (also referred to as 'named pipe' or 'FIFO pipe' because the order of bytes going in is the same coming out [118]).

Timestamps are got at several instants in the aim of obtaining the following KPIs:

Algorithm 2 Thread pool manager

```
1: procedure SCHEDULING
2:     while true do
3:         if queue = ∅ then
4:             wait next event
5:         else
6:             CCDU ← pick queue
7:             process(CCDU)
8:             if decoding_failure = true then
9:                 purge waiting CCDU of the same TB
10:            end if
11:            acknowledge(CCDU) done
12:        end if
13:    end while
14: end procedure
```

- Pre-processing delay, which includes data conditioning, i.e., code block creation, before triggering the channel coding itself.

- Channel coding delay, which measures the runtime of the encoder (decoder) process in the downlink (uplink) direction.

- Post-processing delay, which includes the combination of CBs.

Collected traces contain various performance indicators such as the number of iterations carried out by the decoder per CB as well as the identification of cores affected for both encoding and decoding processes. Decoding failures are detected when a value greater than the maximum number of allowed iterations is registered. As a consequence, the loss rate of channel coding processes as well as the individual workload of cores can be easily obtained.

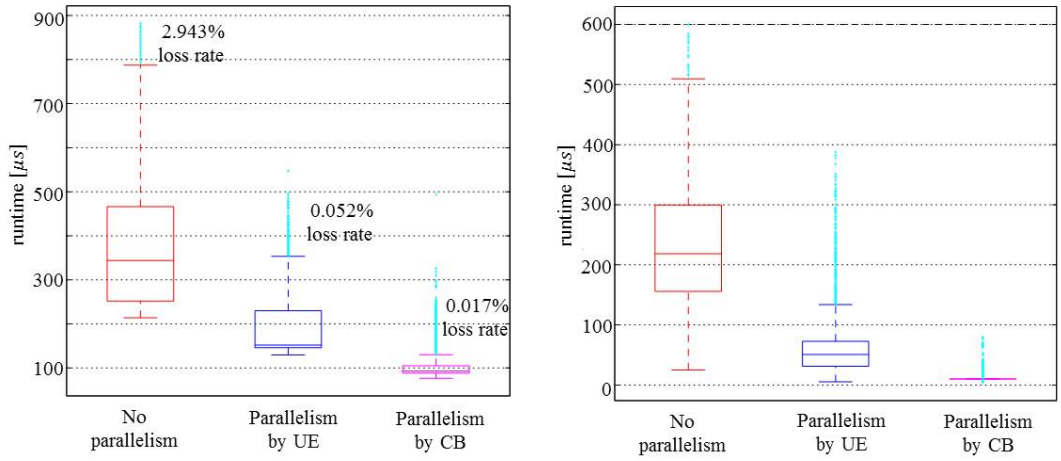
5.3 Performance evaluation

In order to evaluate the performance of the proposed multi-threading processing, we use the above described test-bed which contains a multi-core server hosting the eNB. The various UEs perform file transferring in both uplink and downlink directions.

The test scenario is configured as follows:

- Number of cells: 1 eNB
- Transmission mode: FDD
- Maximum number of RB: 100
- Available physical cores: 16
- Channel coding dedicated cores: 6
- Number of UEs: 3

The performance captor takes multiple timestamps in order to evaluate the runtime of the encoder/decoder itself, as well as, the whole execution time performed by the encoding/decoding function which includes pre- and post-processing delays e.g., code block creation, segmentation, assembling, decoder-bits conditioning (log-likelihood). When a given data-unit is not able to be decoded, i.e., when the maximum number of iterations is achieved without success, data is lost and needs to be retransmitted. This issue is quantified by the KPI referred to as 'loss rate'.

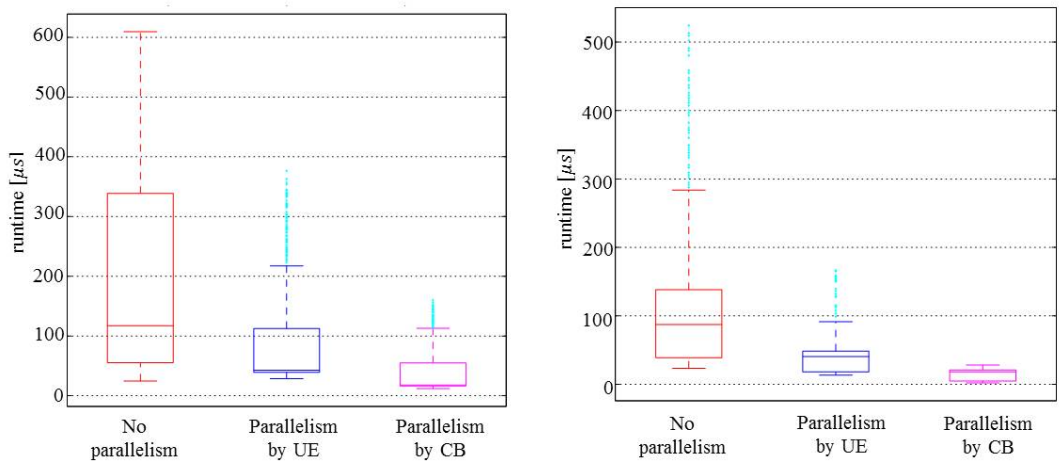


(a) decoding (Rx) (b) decoder (Rx)

Figure 5.5: Decoding runtime (test-bed).

Runtime results are presented in Figures 5.5 and 5.6 for the uplink and downlink directions, respectively.

Decoding function shows a performance gain of $72,6\%$ when executing Code Blocks (CBs) in parallel, i.e., when scheduling jobs at the finest-granularity. Beyond the important latency reduction, runtime values present less dispersion when performing parallelism i.e., runtime values are concentrated around the mean especially when executing CBs in parallel. This fact is crucial when dimensioning cloud-computing infrastructures, and notably data centers hosting virtual network functions with real-time requirements. When considering the gap between CB-parallelism and no-parallelism maximum runtime values, the C-RAN system (BBU-pool) may be moved several tens of kilometers higher in the network.



(a) encoding (Rx) (b) encoder (Rx)

Figure 5.6: Encoding runtime (test-bed).

Chapter 6

Conclusions

Contents

6.1 Main contributions **89**
6.2 Major outcomes **91**
6.3 Research perspectives **92**
6.4 Publications **92**

6.1 Main contributions

In a fist step, we studied in this PhD thesis the performance of Virtualized Network Functions (VNFs) when considering service chaining constraints. We assumed that each VNF is composed of sub-functions to be executed on general purpose hardware of limited capacity, each sub-function requiring a random amount of processing time. We investigated the relevance of resource pooling where available cores in a data center are shared by all active VNFs. We studied by simulation various algorithms for scheduling sub-functions composing active VNFs (namely Greedy, Round Robin and Dedicated Core algorithms). We additionally introduced an execution deadline criterion, which means that VNFs can renege if their sojourn time in the system exceeds by a certain factor their service time. This feature is especially relevant when considering the processing of real-time VNFs. Performance results showed that sub-functions chaining is critical with regard to performance. When sub-functions have to be executed in series, the simple Dedicated Core algorithm is the most efficient. When sub-functions can be executed in parallel, Greedy or Round Robin algorithms have similar performance and outperform the Dedicated Core algorithm. Enabling as much as possible parallelism and avoiding chaining when designing a VNF are fundamental principles to gain from the available computing resources.

To represent the parallelized service mechanism through a simple model, we investigated the $M^{[X]}/M/1$ queuing system with Processor-Sharing discipline, where VNF's jobs arrive in batches to be simultaneously processed by a single server. The computing capacity of this server is considered as the sum of individual capacities of processing units composing a multi-core system. This queuing model enables evaluating virtualized network systems, where the treatment of microservices (VNF's jobs) within requests (batches of VNF's jobs) determines the system performance. We thus addressed the study of the sojourn time of individual jobs as well as the sojourn time distribution of an entire batch. We notably established an exact

expression for the sojourn time distribution of a single job. We further characterized the system performance in various load regimes when using exact asymptotics for the tail of this distribution together with a heavy load convergence result. The sojourn time distribution of an entire batch was finally approximated by means of a simple heuristic when using the exact results of the sojourn time of a single job and assuming that the batch size is not too large. The sojourn time distribution can then be used for dimensioning purposes in order to guarantee that, with a large probability, the service of a job is completed before some lag time.

We further focused our analysis in the performance of virtualized real-time functions as those of the physical layer of the mobile access network, notably the channel coding function when considering a fully centralized Cloud-RAN architecture. An in-depth analysis of the LTE standard and the E-UTRAN protocol stack was performed in order to determine the functional and data decomposition of virtualized RAN functions. Latency was clearly the main key performance indicator used throughout this work, others KPIs such as the loss rate and deadline overtaking were mainly used for dimensioning purposes. To reduce the runtime of the most expensive RAN function in terms of latency, we applied data parallelism and resource pooling in a multi-core system. We have evaluated two methods of parallelism: the first one uses one thread per UE, and the second one goes further, employing one thread per CB. We have implemented a global scheduler to determine the execution order of RAN-jobs. Performance results showed that scheduling sub-functions at the finest granularity can significantly reduce the entire VNF runtime and their variability (i.e., runtime values are more concentrated around the mean). This fact is a step towards the Cloud-RAN implementation, since some tasks of the BBU can be moved higher in the network, which offers more possibilities in the functional split of BBU functions. In the same way, the performance gain enables longer distances between the BBU-pool and the radio elements, as well as, the efficient utilization of computing resources.

In the perspective of running BBU functions on a multi-core system, we investigated the problem of determining the required computing capacity for hosting the base-band processing of various base stations in a Central Office. With this goal in mind, we performed a worst-case analysis. We evaluated by simulation the behavior of a BBU-pool hosting various eNB where the deadline exceedance during the execution of BBU's jobs determine the required computing capacity. We extended our analysis in order to determine the scalability of Cloud-RAN systems. Results reveled that gain persists when the workload rises linearly with the number of cores. It was also shown that the speed up evolution when the computing capacity is increased, is limited by the runtime of the smallest data structure of the channel coding function (namely, a Code Block). This behavior is consistent with Amdah'ls law which says that the performance improvement of parallel processing reaches at some time an upper boundary due to the serial part of the program.

The next main part of this work was dedicated to Cloud-RAN modeling. We introduced a batch queuing model, namely the $M^{[X]}/M/C$ multi-service system, to assess the needed processing capacity in a data center while meeting tight latency requirements in the downlink and uplink directions. In fact, the functional and data decomposition of BBU functions leads to batch arrivals of parallel runnable jobs with non-deterministic runtime. The batch size follows a geometric distribution. The variability of the fronthaul delay and jobs' runtime are captured by the arrival and service distributions, respectively. Since the runtime of a radio subframe becomes the batch sojourn-time, we derived the Laplace transform of this latter quantity, as well as, the probability of exceeding certain threshold to respect LTE radio deadlines. The proposed model was validated by simulation when processing a hundred cells in a multi-core system. We additionally illustrated that the impatience criterion (which reflects RAN time-budgets) is not incident when the probability of deadline exceedance is low enough. Results provided valuable guidelines for sizing and deploying Cloud-RAN systems.

As a proof of concept, we finally implemented on the basis of various open-source solutions, an end-to-end virtualized mobile network which notably includes a virtualized RAN. The platform implements the proposed models and scheduling strategies. The parallel processing of both encoding (downlink) and decoding (uplink) functions is carried out by using multi-threading in a multi-core server within a single process. Latency is considerably reduced since we avoid multi-tasking across different processes. The workload of threads is managed by a global non-preemptive scheduler. Each thread is assigned to a dedicated single core with real-time OS priority and is executed until completion without interruption. The isolation of threads is provided by a specific configuration performed in the OS which prevents from the use of channel coding computing resources for any other job.

Results show important gains in terms of latency, which opens the door for deploying fully centralized cloud-native RAN architectures. *This is the final statement of this thesis: RAN could be considered as a VNF.*

6.2 Major outcomes

In the framework of Network Function Virtualization (NFV), we addressed in this work the performance analysis of virtualized network functions (VNFs), wherein the virtualization of the radio access network (namely, Cloud-RAN) was the driving use-case.

- *VNF modeling by means of stochastic service systems* $M^{[X]}/M/1$ *-PS and* $M^{[X]}/M/C$. These theoretical models revel the behavior of high performance computing architectures based on parallel processing and enable dimensioning the required computing capacity.

- *Job's sojourn-time in an* $M^{[X]}/M/1-PS$ *system.* We established an exact expression for the sojourn time distribution of a single job. *While the mean value was studied in the technical literature by Kleinrock [75], the exact distribution of the sojourn time of a job for exponential service times was so far not known.*

- *Batch's sojourn-time in an* $M^{[X]}/M/C$ *system.* We derived the Laplace transform of the sojourn time of the whole batch which enabled us to obtain an asymptotic estimate of the probability of exceeding a large threshold. This contribution is specially relevant when considering the importance of meeting real-time deadlines in virtualized network systems. In addition, we evidenced that when the service capacity of the system is sufficiently large, the tail of the sojourn time of a batch is dominated by the service time of a single job.

- *C-RAN acceleration by means of multi-threading models.* In a fully centralized Cloud-RAN architecture, latency reduction is particularly relevant because it enables increasing the distance between antennas and BBU functions. This fact improves the concentration level of BBUs in the CO for CAPEX and OPEX savings.

- *Low latency channel coding in open-source.* The proposed multi-threading models were implemented on the basis of open-source code (OAI) for improving the performance of channel coding functions in both uplink and downlink directions.

- *Testbed platform.* As proof of concept, we implemented an end-to-end virtualized mobile network which notably includes the proposed acceleration models.

- *Engineering rules for conceiving future VNFs.* From the study of resource pooling where available cores in a data center are shared by all active VNFs, it stated that avoiding chaining is fundamental when conceiving VNFs in order to achieve low latency performance.

6.3 Research perspectives

The performance analysis of virtualized network functions carried out throughout this work gives rise to new research perspectives, notably:

- *Applying decomposition patterns for achieving parallelism in network service chains.* Rethinking legacy network functions is crucial for achieving virtualized network architectures. Beyond reducing the number of functions to be executed in series, modern IT solutions such as microservices-based architectures are needed when implementing NFV services. Optimal software solutions shall allow future network services to be managed and coordinated across distributed computing servers forming a large virtualized infrastructure or more generally in the cloud.

- *Cloud-native RAN network functions.* The outcomes presented in this study specially concerning runtime reduction enable the deployment of RAN functions in the cloud. Field experiments of fully 'cloudified' mobile networks become the next step.

- *Implementing flexible radio interfaces between distributed and centralized units.* Making RAN functions cloud-native absolutely requires flexible interfaces for carrying radio signals between antennas and centralized computing servers. We are concretely interested in implementing the Functional Split VI presented in Chapter 3 subsection (3.2.4) for moving the whole radio processing (which includes channel encoding/decoding) higher in the backhaul network while keeping only low-PHY functions near to the radio elements.

- *Multi-resource allocation strategies.* The presented models can be extended to multi-resource allocation particularly when considering the required RAM memory during the execution of real-time network functions. Similarly, the required bandwidth capacity of fiber links that connect the building blocks of VNFs hosted in distributed data centers has to be taken into account.

- *Energy-efficient NFV infrastructures.* Reducing the runtime of VNFs leads to important energy savings. This claim may be easily quantified from the study of energy consumption of cloud computing servers.

- *A cooperative allocation of cloud and radio resources.* It could be worth analyzing the performance of coordinated radio and cloud scheduling. Moreover when considering Coordinated Multi-points (CoMPs) technologies where the base band processing of a given radio subframe might be reused, for instance in the case of multi-point transmission. The required computing capacity for processing radio signals may be considerably reduced, while improving the user QoE.

6.4 Publications

International Conference and Journal papers

- "Cloud-RAN modeling based on parallel processing". IEEE Journal on Selected Areas in Communications - Special Issue on Network Softwarization & Enablers (JSAC-2018).

- "Sojourn time in an $M^{[X]}/M/1$ processor sharing queue with batch arrivals". Stochastic Models Journal 2018.

- "Performance Analysis of VNFs for sizing Cloud-RAN infrastructures". IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN-2017).

- "On dimensioning Cloud-RAN architectures". EAI International Conference on Performance Evaluation Methodologies and Tools (ValueTools-2017).
- "Towards the deployment of a fully centralized Cloud-RAN architecture". IEEE Conference on Wireless Communications and Mobile Computing Conference (IWCMC-2017).
- "VNF modeling towards the cloud-RAN implementation". IEEE Conference on Networked Systems (NetSys-2017).
- "Performance analysis of resource pooling for network function virtualization". IEEE Symposium on Telecommunications Network Strategy and Planning (Networks-2016).

Patents

- ID:81333015. Reference number: 20386FR01. Controlled swapping between various RANs belonging to different Mobile Network Operators (MNOs), 2017. It notably enables the user's QoE enhancement as well as CAPEX savings.
- ID:81334285. Reference number: 201297FR01. Radio and cloud multi-site coordination for heterogeneous Cloud-RAN architectures, 2017. It considers a predictive process (based on machine learning) for the scheduling of both radio and cloud resources. Main expected benefits are the convergence of heterogeneous C-RAN functional splits, the QoE improvement and the efficient utilization of radio and cloud resources.

Relevant Conference and Poster Sessions

- "Sojourn time in an $M^{[X]}/M/1$ processor sharing queue with batch arrivals", European Conference on Queuing Theory, Jerusalem, 2018.
- "Virtualized multi-RAT test-bed platform", OAI Workshop, Beijing, 2018.
- "Processing Radio Access Network Functions on General Purpose Computers", Poster Session, Doctoral Summer School - Rescom, 5G and IoT, Lorient, 2016.
- "Towards Cloud-RAN implementation", Poster Session, PhD Students Days, Orange Labs, Paris, 2017.
- "Cloud-RAN modeling", $< I/O >$ Inria-Orange Lab Days, Paris, 2017.

Awards and Recognitions

- "2nd PhD Thesis Award", EDITE de Paris, Sorbonne Université, Paris, 2018.
- "Student Travel Grant", IEEE Conference on Network Function Virtualization and Software Defined Networks, Berlin, 2017.
- "Best Poster Award", PhD Students Days, Orange Labs, Paris, 2017.

Appendices

Appendix A

Sojourn time in an $M^{[X]}/M/1$ Processor Sharing Queue

We study in [6], the $M^{[X]}/M/1$ Processor Sharing queue with batch arrivals. The sojourn time W of a single job and the entire batch are investigated. This queuing model is motivated by the evaluation of Cloud Computing or Virtualized Network systems where the treatment of microservices within requests determines the global system performance.

We first show that the distribution of sojourn time W of a job can be obtained from an infinite linear differential system; the structure of this system, however, makes the explicit derivation of this distribution generally difficult. When further assuming that the batch size has a geometric distribution with some given parameter $q \in [0, 1)$, this differential system can be analyzed via a single generating function $(x, u) \mapsto H_q(x, u)$ which is shown to verify a second-order partial differential equation involving a boundary term at point $u = q$.

Solving this partial differential equation for H_q with required analyticity properties determines the one-sided Laplace transform $H_q^*(\cdot, u)$ for given u. Writing $H_q^*(\cdot, u)$ in terms of a multivariate hypergeometric function enables us to extend its analyticity domain to a cut-plane $\mathbb{C} \setminus [\sigma_q^-, \sigma_q^+]$, for negative constants σ_q^- and σ_q^+. By means of a Laplace inversion of $H_q^*(\cdot, u)$ for a suitable value of u, the complementary distribution function $x \mapsto \mathbb{P}(W > x)$ is then given an explicit integral representation. This enables us to show that the tail of this distribution has an exponential decay with rate $|\sigma_q^+|$, together with a sub-exponential factor. A convergence in distribution is also asserted for W in heavy load condition, the limit distribution exhibiting a sub-exponential behavior itself.

Using our exact results for the sojourn time of a single job, we finally discuss an approximation for the distribution of the sojourn time of an entire batch when assuming that the batch size is not too large.

General considerations

All distributions related to the $M^{[X]}/M/1$ queue are defined in that stationary regime. Let $\mathbb{P}(B = m) = q_m$, $m \geq 1$, define the distribution of the size B of any batch; it is known ([74], Vol.I, §4.5) that the number N_0 of jobs present in the queue has a stationary distribution whose generating function is given by

$$\mathbb{E}(z^{N_0}) = \frac{\mu(1-\varrho^*)(1-z)}{\mu(1-z) - \lambda z(1-\mathbb{E}(z^B))}, \quad |z| < 1, \tag{A.0.1}$$

where $\varrho^* = \lambda\mathbb{E}(B)/\mu$ is the system load; in particular, $\mathbb{P}(N_0 = 0) = 1 - \varrho^*$.

As motivated above, we here aim at characterizing the sojourn time W of a job entering the $M^{[X]}/M/1$ queue with batch arrivals and PS discipline; both the stationary distribution and its tail behavior at infinity will be derived. As reviewed below, this distribution has been determined for this queue in the case $B = 1$; the case of a general batch size $B \geq 1$ has been considered but for the derivation of the first moment $\mathbb{E}(W)$ only. To our knowledge, the distribution of sojourn time W for a batch size $B \geq 1$ is newly obtained in the present study.

State-of-the-art

We briefly review available results on the distribution of sojourn times in the PS queue. The average sojourn time $w(x) = \mathbb{E}(W \,|\, S = x)$ of a job, given that it requires some service time $S = x$, has been addressed in [74] for some more general class of service time distributions than the exponential one; the analysis proceeds from an integral equation verified by the function $w = w(x)$, $x \geq 0$. For an exponentially distributed service time S, in particular, it is shown that the conditional mean $w(x)$ reduces to

$$w(x) = \frac{x}{1-\varrho^*} + \frac{\mathbb{E}(B^2) - \mathbb{E}(B)}{\lambda\mathbb{E}(B)^2}\left[\frac{(1-(1-\varrho^*)^2)(1-e^{-\mu(1-\varrho^*)x})}{2(1-\varrho^*)^2}\right]; \tag{A.0.2}$$

deconditioning $w(x)$ with respect to an exponentially distributed variable $S = x$ with parameter μ then gives the unconditional mean sojourn time

$$\mathbb{E}(W) = \frac{1}{\mu(1-\varrho^*)}\frac{\mathbb{E}(B^2) + \mathbb{E}(B)}{2\,\mathbb{E}(B)}. \tag{A.0.3}$$

For exponentially distributed service times and a batch size $B = 1$, the unconditional Laplace transform $s \mapsto \mathbb{E}(e^{-sW})$ of the sojourn time W can be derived via the explicit resolution of an infinite linear differential system in some Hilbert space L^2 of summable squared sequences; this resolution is performed by applying the classical spectral theory of self-adjoint linear operators in this Hilbert space ([119], Corollary 3 and references therein). Given the load $\varrho = \lambda/\mu$, the inversion of this Laplace transform then provides the integral representation

$$\mathbb{P}(W > x) = (1-\varrho)\int_0^\pi \frac{\sin\theta}{(1+\varrho-2\sqrt{\varrho}\cos\theta)^2}\frac{e^{h_0(\theta,x)}}{\cosh\left(\frac{\pi}{2}\cot\theta\right)}\,\mathrm{d}\theta, \quad x \geq 0, \tag{A.0.4}$$

for the complementary distribution function (c.d.f.) of the sojourn time W (see [119], Theorem 1), with exponent $h_0(\theta, x) = \cot\theta\left(2\Psi_0 - \frac{\pi}{2} + \theta\right) - (1+\varrho-2\sqrt{\varrho}\cos\theta)x$ and where the angle $\Psi_0 = \Psi_0(\theta)$ is defined by

$$\Psi_0 = \mathrm{Arctan}\left[\frac{\sqrt{\varrho}\sin\theta}{1-\sqrt{\varrho}\cos\theta}\right], \quad \theta \in [0,\pi]. \tag{A.0.5}$$

Using formula (A.0.4), the distribution tail of sojourn time W can be obtained in the form

$$\mathbb{P}(W > x) \sim c_0(\varrho)\left(\frac{\pi}{\mu x}\right)^{\frac{5}{6}} \exp\left[-(1-\sqrt{\varrho})^2\mu x - 3\left(\frac{\pi}{2}\right)^{\frac{2}{3}}\varrho^{\frac{1}{6}}(\mu x)^{\frac{1}{3}}\right] \tag{A.0.6}$$

for large x and fixed $0 < \varrho < 1$, with coefficient

$$c_0(\varrho) = \frac{2^{\frac{2}{3}}}{\sqrt{3}\varrho^{\frac{5}{12}}}\frac{1+\sqrt{\varrho}}{(1-\sqrt{\varrho})^3}\exp\left(\frac{1+\sqrt{\varrho}}{1-\sqrt{\varrho}}\right);$$

this tail therefore shows an exponential trend with decay rate $(1-\sqrt{\varrho})^2\,\mu$ corrected by a sub-exponential factor. An estimate similar to (A.0.6) was first obtained in [76] for the waiting time $\widetilde{W}$ of the $M/M/1$ queue with Random Order of Service discipline, $\widetilde{W}$ being defined as the time spent by a job in the queue up to the beginning of its service; it has been eventually shown [120] that the distribution of $\widetilde{W}$ and that of the sojourn time W of the $M/M/1$ with PS discipline are related by the remarkable relation

$$\forall\, x \geq 0, \quad \mathbb{P}(W > x) = \frac{1}{\varrho}\,\mathbb{P}(\widetilde{W} > x). \tag{A.0.7}$$

For exponentially distributed service times and a batch size $B = 1$ again, the tail behaviors at infinity of the distributions of waiting time $\widetilde{W}$ and sojourn time W are therefore of identical nature.

Main contributions

For the presently considered $M^{[X]}/M/1$ Processor Sharing queue with batch arrivals,

A) we first show that the conditional distribution functions G_n of W, given the queue occupancy $n \geq 0$ (in number of jobs) at the batch arrival instant, verify an infinite-dimensional linear differential system ([30], Section 2);

B) assuming further that the batch size is geometrically distributed with parameter $q \in [0,1)$, the resolution of this differential system is reduced to that of a partial differential equation (PDE) for the generating series $H_q(x,u) = \sum_{n\geq 0} G_n(x)u^n$, $x \in \mathbb{R}^+$, $|u| < 1$ (the index q for H_q is meant to stress the dependence on this parameter). While linear and of second order, this PDE for function H_q is non-standard in that it involves an unknown boundary term at point $u = q$ ([30], Section 3);

C) using analyticity properties to determine the boundary term at $u = q$, the unique solution H_q to the PDE is derived via its one-sided Laplace transform $H_q^*(\cdot,u)$ in the form

$$H_q^*(s,u) = \frac{u-q}{u}\,\frac{\mathfrak{L}_q(s,u)}{(u-U_q^+(s))(u-U_q^-(s))} + \frac{q}{u}\,\frac{\mathfrak{L}_q(s,0)}{sq+\varrho+q}$$

for $s \geq 0$ and given u, where $\mathfrak{L}_q(s,u)$ is defined as the definite integral

$$\mathfrak{L}_q(s,u) = \int_u^{U_q^-(s)} \left(\frac{\zeta - U_q^+(s)}{u - U_q^+(s)}\right)^{C_q^+(s)-1} \left(\frac{\zeta - U_q^-(s)}{u - U_q^-(s)}\right)^{C_q^-(s)-1} \frac{\mathrm{d}\zeta}{(1-\zeta)^2}$$

involving the two roots $U_q^{\pm}(s)$ of some quadratic equation $P_q(s,u) = 0$ in u, and with exponents $C_q^{\pm}(s) = -(U_q^{\mp}(s) - q)/(U_q^{\pm}(s) - U_q^{\mp}(s))$. Once expressed in terms of a multivariate hypergeometric function, this solution $H_q^*(\cdot,u)$ can be analytically extended to a cut-plane $\mathbb{C} \setminus [\sigma_q^-, \sigma_q^+]$ of variable s, for some negative constants σ_q^- and σ_q^+ depending on q and system parameters ([30], Section 4);

D) by means of a Laplace inversion, the distribution function of W can finally be given an integral representation which generalizes the specific formula (A.0.4) to batches with any size (provided their distribution is geometric). Its tail behavior

$$\mathbb{P}(W > x) \sim \left(\frac{A_q}{x}\right)^{\frac{5}{6}} \exp\left[\sigma_q^+ x - B_q x^{\frac{1}{3}}\right]$$

for large x and some constants A_q, B_q depending on ϱ and q, exhibits an exponential decay with rate $|\sigma_q^+|$, together with a sub-exponential factor which generalizes the estimate (A.0.6) already known for $q = 0$ ([30], Section 5);

E) furthermore, a heavy load convergence theorem is derived for time W, when properly scaled; the limit distribution $V^{(0)}$ is determined explicitly and exhibits, in particular, a sub-exponential tail given by

$$\mathbb{P}(V^{(0)} > y) \sim \sqrt{\pi} y^{\frac{1}{4}} \exp(-2\sqrt{y})$$

for large y, independently of parameter q ([30], Section 6);

F) using the latter results for the sojourn time of a single job, we finally discuss an approximation for the distribution of the sojourn time of an entire batch. This approximation proves reasonably accurate in the case when the batch size is not too large ([30], Section 7).

Sojourn time of a batch

Light load

First consider the case when $\varrho = 0$ with an isolated batch. Given $B = m$, the sojourn time Ω simply equals the workload of this batch, composed of a sum of m i.i.d. exponentially distributed variables with identical mean $1/\mu = 1$; as a consequence, we have $\mathbb{E}(e^{-s\Omega}|B = m) = (s+1)^{-m}$ for all $s \geq 0$ hence

$$\mathbb{E}(e^{-s\Omega}) = \sum_{m \geq 1} (1-q) q^{m-1} \frac{1}{(s+1)^m} = \frac{1-q}{1-q+s}, \quad s \geq 0,$$

and the Laplace inversion readily gives

$$\mathbf{D}_q(x) = e^{-(1-q)x}, \quad x \geq 0, \tag{A.0.8}$$

and, in particular, $M_q = \mathbb{E}(\Omega) = 1/(1-q)$. Interestingly, (A.0.8) shows that the distribution of the sojourn time Ω of a batch is identical to that of a single job of this batch, as obtained in ([30] (6.1) A.0.9[1]) for $\varrho = 0$. This can be interpreted by saying that all jobs having the same duration in distribution, the distribution of the maximum sojourn duration among them is also that of a single job.

The expression (A.0.9) for function $\mathbf{G}_q$ with $\varrho = 0$ enables us to make the approximation (2.6.13) explicit in the form

$$\mathfrak{A}_q(x) = \frac{e^{-(1-q)x}}{1-q+q\,e^{-(1-q)x}}, \quad x \geq 0. \tag{A.0.10}$$

The mean associated with $\mathfrak{A}_q$ is, in particular,

$$\mathfrak{M}_q = -\frac{\log(1-q)}{q} \cdot \frac{1}{1-q}$$

which is asymptotic to the exact average M_q for small q, but is larger than M_q for increasing q. Similarly (A.0.10) entails that $\mathfrak{A}_q(x) \sim \mathbf{D}_q(x)/(1-q)$ is greater than $\mathbf{D}_q(x)$ for large x, all the more than q is close to 1. We thus conclude that, for light load ϱ, approximation (2.6.13) provides an upper bound for the distribution $\mathbf{D}_q$.

[1] When $\varrho \downarrow 0$, we have $W \Longrightarrow W^{(0)}$ in distribution where

$$\mathbb{P}(W^{(0)} > x) = e^{-(1-q)x}, \quad x \geq 0. \tag{A.0.9}$$

Results

The performance of Cloud Computing systems has been analyzed by evaluating the sojourn time W of individual jobs within requests. A single $M^{[X]}/M/1$ Processor Sharing queuing model has been proposed for this evaluation, assuming that the incoming batches have a geometrically distributed size. The Laplace transform of the sojourn time of a single job has been derived through the resolution of a partial differential equation with unknown boundary terms; analyticity arguments have enabled us to solve this equation, and to provide an exact integral representation for the distribution of W. Exact asymptotics for the tail of this distribution, together with a heavy load convergence result, have been further obtained to characterize the system performance in various load regimes. An independence argument is finally discussed for estimating the distribution of the sojourn time Ω of a whole batch, with assessment on the basis of numerical observations.

The precise account of the temporal correlation between jobs pertaining to a given batch is an open issue. As an extension to the present analysis for the sojourn time of single jobs, nevertheless, we believe that the exact derivation of the distribution of the batch sojourn time Ω can be envisaged using the same assumptions, i.e., exponentially distributed service times and geometrically distributed batch size. The derivation of a system of differential equations for conditional distributions related to Ω and its resolution through an associated linear, second-order partial differential equation with boundary terms could then be addressed in a derivation framework similar to that of the present study.

Appendix B

Cloud-RAN applications

The most popular C-RAN use cases rely on areas with huge demand, such as high density urban areas with macro and small cells, public venues, etc.

Various C-RAN applications are currently considered by academia and industry, the most popular ones are listed below:

- *Network slicing* offers a smart way of segmenting the network and of supporting customized services, e.g, a private mobile network. Slices can be deployed with particular characteristics in terms of QoS, latency, bandwidth, security, availability etc. This scenario and the strict C-RAN performance requirements are studied in [121].

- *Multi-tenancy 5G networks* handle various Virtual mobile Network Operators (VNOs), different Service Level Agreements (SLAs) and Radio Access Technologies (RATs). A global 5G C-RAN scenario is given in [122]. Authors propose a centralized Virtual Radio Resource Managements (VRRMs) so-called "V-RAN enabler" to orchestrate the global environment. The management entity estimates the available radio resources based on the data rate of different access technologies and allocate them to the various services in the network by the OAI scheduler.

- *Coexistence of heterogeneous functional splits* for supporting fully or partially centralization of RAN network functions. An in-depth analysis of the performance gain when performing different functional splits is presented in [83]. Results show that the performance decreases when lower-layer RAN functions are kept near to antennas. This work recommends full centralization to take advantage of the "physical" inter-cell connectivity for deploying advanced multi-point cooperation technologies to improve the QoE.

- *Intelligent Networks* enable the automated deployment of eNBs for offering additional capacity in real time [122]. These intelligent procedures bring an enormous economic benefit for network operators, which today massively invest to extend their network capacity.

Other rising Cloud-RAN applications have been summarized in [83]. It includes massive Internet of Things (IoT) applications, broad band communications, which are delay-sensitive (e.g., virtual reality, video replay at stadiums, etc.), low-latency and high-reliability applications such as assisted driving and railway coverage, since C-RAN enables fast hand-over for UEs moving with high speed.

Appendix C

Fronthaul capacity reduction for Functional Split I

I/Q compression by Redundancy removal

When removing the redundancy in the spectral domain [99], the I/Q data rate can be significantly reduced.

Current LTE implementation, I/Q signal is spectrally broader than necessary where redundancy is mainly due to oversampling.

For instance, for an eNB of 20 MHz, the useful number of sub-carriers is given by $N_{sc} = N_{RB} * N_{sc-pRB}$ = 100RB∗12sub-carriers= 1200. Since the bandwidth of a sub-carrier is 15 KHz, the required cell bandwidth is $1200 * 15 = 18$ MHz. The remaining 2 MHz are used for filter edge roll-off. Nevertheless, in current practice 30.72 MHz bandwidth is used. This is due to the Inverse Fast Fourier Transform (IFFT) size which is based on a power of two. Thus, the smallest size power of two which is larger than 1200 is 2048. Hence, the transmitted bandwidth (namely, $f_s = N_{FFT} * BW_{sc}$) is 2048*15 KHz = 30.72 which produces a redundancy of 10 MHz. Table C.1 shows the sampling frequency according to the commercial cell bandwidth. We also present the useful bandwidth, $BW_{uf} = N_{sc} * BW_{sc}$, and the resulting overhead.

Table C.1: Useful RAN bandwidth.

BW_{cell}	1.4	3	5	10	15	20
N_{RB}	6	12	25	50	75	100
N_{sc}	72	144	300	600	900	1200
N_{FFT}	128 (2^7)	256 (2^8)	512 (2^9)	1024 (2^{10})	1536	2048 (2^{11})
f_s	1.92	3.84	7.68	15.36	23.04	30.72
BW_{uf}	1.08	2.16	4.5	9	13.5	18
Overhead	0.84	1.68	3.18	6.36	9.54	12.72

BW_{cell}, f_s, BW_{uf} and the *Overhead* are in MHz

Guo et al. propose in [99] adding zeros to the original signal for up-sampling it with rate f_s. This signal is passed through a low-pass filter, for finally down-sampling it with rate f_{ds}. The efficiency of the process is evaluated by using the so-called down-sampling factor F, which is given by $F = \frac{f_{ds}}{f_s} \leq 1$. The resulting bandwidth is limited to $[-f_{ds}/2, -f_{ds}/2]$.

I/Q compression by non-uniform quantization

I/Q compression can be performed by implementing a non-uniform quantizer [99]. Fronthaul compression and optimization are being studied by the vRAN Fronthaul Group in the framework of the Telecom Infra Project (TIP) [123].

Since Orthogonal Frequency Division Multiplexing (OFDM) uses multiple narrow-band subcarriers, a large Peak-to-average power ratio (PAPR) is produced mainly in the downlink direction. In the uplink, in addition to Single Carrier Frequency Division Multiple Access (SC-FDMA) technology, an Automatic Gain Control (AGC) is used in order to reduce the difference in power between users. However, either in downlink or uplink directions, LTE signals have a large dynamic amplitude range [99] In LTE, time domain samples are transmitted using a constant number of bits per complex component (I/Q), namely $M = 15$ bits. Thus, there are $2^M = 32768$ available signal levels which can be unnecessary for representing high amplitude signals. Nevertheless, if all I/Q signals are compressed to target M^c bits (where $M^c < M$, then components with low amplitude can be strongly impacted.

The proposed solution is to implement an adaptive quantization process. I/Q signals are segmented in small blocks of N_s samples. Each block i uses a particular M_i^c number of bits. The scaling factor, i.e., $\frac{M}{M_i^c}$ is transmitted as overhead in each block. It can be easily deduced that minimizing the block length enables reducing the subsequent quantization error. However, it will increase the overhead.

Evidently, there is a correlation between the resolution M and the resulting I/Q data rate. A higher resolution improves the signal quality while increasing the required fronthaul capacity.

Appendix D

Resource Allocation Grid

The simulation tool performs the radio resource grid allocation while selecting the modulation order Q_m and more concretely the MCS index I_{MCS} based on the emulated radio conditions, i.e., the SINR. It is shown in Table D.1.

Table D.1: Key Features of Modulation and Coding when $N_{RB} = 100$.

CQI	Modulation	Q_m	MCS index I_{MCS}	SINR [dB]	TBS [bits]	k[1]
1-6	QPSK	2	0 - 9	< 3	2792-15840	0.08-0.47
7- 9	16-QAM	4	10 - 16	< 9	15840-30576	0.24-0.46
10- 15	64-QAM	6	17 - 28	< 20	0.30-0.75	

In the same way, we define the number of PRB for a UE in function of the emulated traffic in the cell and of the data load per user. Note that for an LTE Bandwidth of 20 MHz the available number of PRB is 100 [106].

We can easily identify the I_{TBS} from the I_{MCS} looking up the table 7.1.7.1-1 presented in the LTE Physical layer specification 3GPP TS 36.213 version 12.4.0 Release 12 [106]. Used values are shown in Table D.2.

Table D.2: MCS and TBS correlation.

MCS index I_{MCS}	Q_m	TBS index I_{TBS}
0-9	2	$I_{TBS} = I_{MCS}$
10- 16	4	$I_{TBS} = I_{MCS} - 1$
17- 28	6	$I_{TBS} = I_{MCS} - 2$

Based on the TBS index, I_{TBS}, and the number of Physical RB, N_{RB}, we determine the TBS from the table 7.1.7.2.1.1-1 of the LTE PHY specification 36.213 version 12.4.0 Release 12 [106]. It is for transport blocks not mapped to two or more layer-spatial multiplexing, i.e., SISO configuration. A fragment is shown in Table D.3

Table D.3: An example of TBS as a function of N_{RB}.

I_{TBS}	N_{RB}					
	4	5	6	7	8	9
12	904	1128	1352	1608	1800	2024
13	1000	1256	1544	1800	2024	2280
14	1128	1416	1736	1992	2280	2600

Appendix E

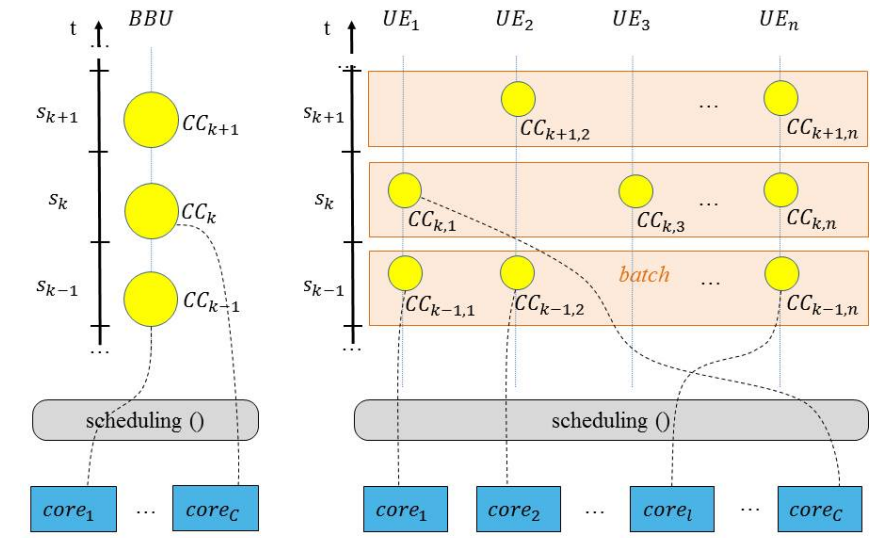
Scheduling Strategy

In the framework of parallel computing on multi-core platforms, we define a global scheduler which selects the next job to be processed among those which are ready to be executed. The scheduler allocates a single processing unit (core) to the next job in the queue in a FIFO order. We use non-preemptive scheduling, in this way, a running job cannot be interrupted and it is executed until completion. As shown in Figure E.1, we consider data decomposition in such a way that the channel coding sub-function is performed on different sub-sets of data at the same time, i.e., each parallel task works on a portion of data. Note that functional parallelism is not applicable on the PHY layer of BBUs, since modulation, channel coding and IFFT sub-functions must be executed in series.

In the framework of software-based BBUs, each radio subframe generates a job for channel coding processing, i.e., a new job requires service every millisecond. When parallel computing is not applied, as shown in Figure E.1(a), a channel coding job denoted by CC_k for the k-th subframe is executed on a single core even when more cores are available. If the channel coding job CC_k arrives when the previous one CC_{k-1} is not finished, the scheduler allocates to it the next free core. In the case that all cores are busy, the job is queued. This scenario is not desirable when executing base-band functions, mainly when closed-loop processes are active, e.g., the HARQ process defined in LTE.

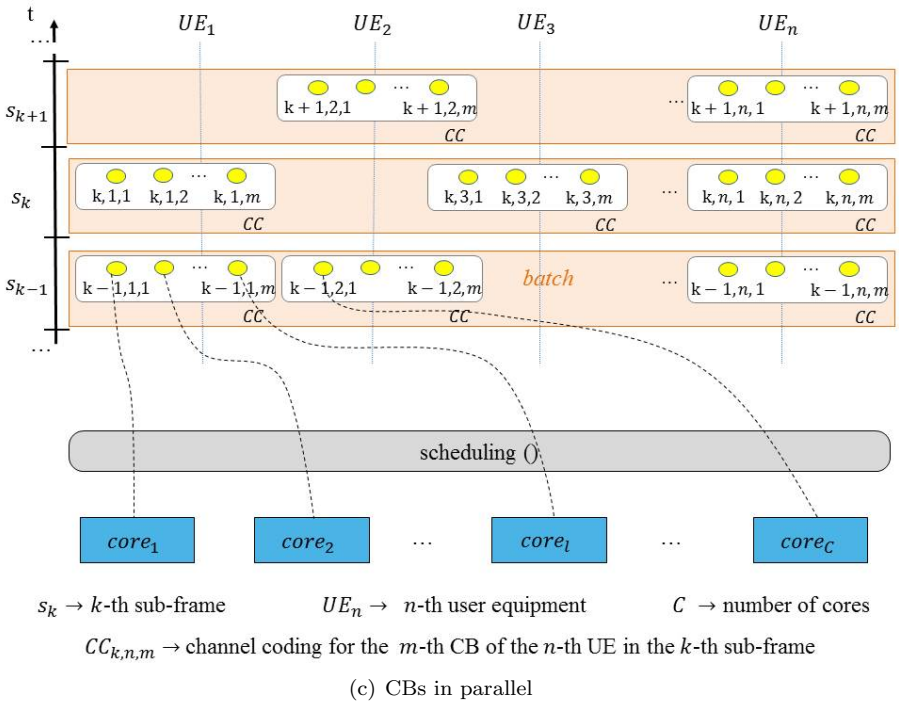
In order to speed up the channel coding sub-function, we decompose each channel coding job CC_k in a suite of sub-jobs. Each parallel sub-job works on a portion of BBU data. The sub-job belonging to the n-th UE allocated in the k-th subframe is denoted by $CC_{k,n}$. Channel coding jobs arrive in batches at the computing platform every millisecond. (See Figure E.1(b) for an illustration.) The batch size (i.e., the number of active jobs) is variable and corresponds to the number of UEs allocated into a subframe. The service time of a job in this batch varies in function of the TBS which is determined by the radio scheduler in the MAC layer. Batches are processed in a First-Come-First-Serve discipline. The global scheduler selects jobs and assigns them one core to each.

We now consider a finer granularity in the decomposition of channel coding sub-functions. Thus, each job $CC_{k,n}$ of the n-th UE is split in a set of m parallel sub-jobs, where the m-th sub-job is denoted by $CC_{k,n,m}$. This is illustrated in Figure E.1(c). The resulting number of parallel channel coding jobs, i.e., the batch size, varies every millisecond (or more generally, according to the periodicity of radio scheduling). Thus, the batch size is the product of active UEs and the number m of CBs. Note that m can take different values for each UE. As in the preceding threading model, incoming batches join the tail of a single queue and each job gets access to a particular core under FIFO criterion.



(a) No parallelism

(b) UEs in parallel



(c) CBs in parallel

Figure E.1: Global scheduler.

Appendix F

Analysis of the Markov chain considering an $M^{[X]}/M/C$ system

To completely determine the sojourn time of a batch in an $M^{[X]}/M/C$ queuing system, it is necessary to know the number of jobs (y_n) belonging to the tagged batch when the n-th job enters service.

The Markov chain (y_n) describing the number of jobs of a tagged batch at time (τ_n^+) for $n = 1, \ldots, b$ is such that

$$y_n = \begin{cases} y_{n-1} & \text{with probability } y_{n-1}/C, \\ y_{n-1} + 1 & \text{with probability } (C - y_{n-1})/C. \end{cases} \tag{F.0.1}$$

By definition $y_n \in \{1, \ldots, C\}$ and the Markov chain is absorbed at state C. We show the following result, where we used Stirling numbers of the second kind and polynomials $\mathcal{A}_{n,p}(x)$ defined by Equations (4.2.1) and (4.2.2), respectively

Lemma F.0.1. *The conditional transition probabilities of the Markov chain (y_n) are given for $k \geq \ell$ by*

$$\mathbb{P}(y_n = k \mid y_1 = \ell) = \frac{(C-\ell)!}{(C-k)!C^{n-1}} \sum_{m=0}^{n-1} \binom{n-1}{m} S(m, k-\ell)\ell^{n-1-m} = \frac{1}{C^{n-1}} \binom{C-\ell}{k-\ell} \mathcal{A}_{n-1,k-\ell}(\ell). \tag{F.0.2}$$

Proof. The transition matrix of the Markov chain (y_n) is

$$P = \begin{pmatrix} \frac{1}{C} & \frac{C-1}{C} & 0 & . & \\ 0 & \frac{2}{C} & \frac{C-2}{C} & 0 & . \\ & & \ddots & & \\ & & & 0 & 1 \end{pmatrix}.$$

Let us first consider the case $y_1 = 1$, we have

$$\mathbb{P}(y_n = k \mid y_1 = 1) = e_1^t P^{n-1} e_k,$$

where e_n is the column vector with all entries equal to 0 except the n-th one equal to 1, and e_n^t is the transpose of vector e_n.

To compute the matrix P^n, we diagonalize the matrix P. For this purpose, we note that

the eigenvalues of matrix P are obviously the positive reals $x_j = \frac{k}{C}$ for $j = 1, \ldots, C$. Simple computations show that the eigenvector associated with the eigenvalue x_j is the vector v_j with entries

$$v_j(\ell) = \frac{(j-1)\ldots(j-\ell+1)}{(C-1)\ldots(C-\ell+1)}$$

for $\ell = 1, \ldots, j$ (with $v_1 = 1$) and $v_j(\ell) = 0$ for $\ell = j+1, \ldots, C$. Note that we have for $\ell = 1, \ldots, j$

$$v_j(\ell) = \frac{(j-1)!(C-\ell)!}{(j-\ell)!(C-1)!}.$$

The vectors v_j for $j = 1, \ldots, C$ are obviously linearly independent and form a basis of $\mathbb{R}^C$. To compute $P^n e_k$, we determine the representation of e_k on the basis (v_j). By setting

$$e_k = \sum_{j=1}^{C} u_j^{(k)} v_j,$$

we easily check that $u_j^{(k)} = 0$ for $j = k+1, \ldots, C$ and then

$$\begin{cases} v_k(k) u_k^{(k)} = 1 \\ \sum_{j=\ell}^{k} v_j(\ell) u_j^{(k)} = 0 \quad \text{for } 1 \le \ell < k. \end{cases}$$

We then deduce that

$$u_k^{(k)} = \frac{(C-1)!}{(C-k)!(k-1)!} \tag{F.0.3}$$

and

$$\sum_{j=\ell}^{k-1} \frac{(j-1)!}{(j-\ell)!} u_j^{(k)} = -\frac{(k-1)!}{(k-\ell)!} u_k^{(k)} = -\frac{(C-1)!}{(C-k)!(k-\ell)!}.$$

The above equation can be rewritten as

$$\sum_{j=\ell}^{k-1} \binom{j}{\ell} \frac{u_j^{(k)}}{j} = -\frac{(C-1)!}{(C-k)!(k-\ell)!\ell!}$$

for $\ell = 1, \ldots, k-1$. In matrix form, this equation reads $U_k \widetilde{u}^{(k)} = \beta^{(k)}$, where the matrix U_k with coefficients equal to $\binom{j}{\ell}$ for $\ell = 1, \ldots, k-1$, $\ell \le j \le k-1$ and other coefficients equal to 0, is an upper triangular Pascal matrix, β_k is the vector with entries

$$\beta_k(\ell) = -\frac{(C-1)!}{(C-k)!(k-\ell)!\ell!}$$

for $\ell = 1, \ldots, k-1$ and $\widetilde{u}^{(k)}$ is the vector with entries equal to $u_j^{(k)}/j$ for $j = 1, \ldots, k-1$.

It is classical that the inverse of the matrix U_k is the upper triangular matrix with coefficients $(-1)^{j+\ell}\binom{j}{\ell}$ for $\ell = 1, \ldots, k-1$, $\ell \le j \le k-1$ and other coefficients equal to 0.

Hence, $\widetilde{u}^{(k)} = U_k^{-1}\beta^{(k)}$ and then for $\ell = 1, \ldots, k-1$

$$\begin{aligned} u_\ell^{(k)} &= -\ell \sum_{j=\ell}^{k-1} (-1)^{j+\ell} \binom{j}{\ell} \frac{(C-1)!}{(C-k)!(k-\ell)!\ell!} \\ &= \frac{(-1)^{\ell+1}(C-1)!}{(C-k)!(\ell-1)!} \sum_{j=\ell}^{k-1} \frac{(-1)^j}{(k-j)!(j-\ell)!} \\ &= (-1)^{k+\ell} \frac{(C-1)!}{(C-k)!(\ell-1)!(k-\ell)!} \end{aligned}$$

since

$$\sum_{j=\ell}^{k-1} \frac{(-1)^j}{(k-j)!(j-\ell)!} = \frac{(-1)^\ell}{(k-\ell)!} \sum_{j=0}^{k-\ell-1} (-1)^j \binom{k-l}{j} = \frac{(-1)^\ell}{(k-\ell)!}((1-1)^{k-\ell} - (-1)^{k-\ell}).$$

It is worth noting that the above expression is also valid for $\ell = k$ in view of Equation (F.0.3).

We eventually come up with the representation

$$e_k = \sum_{j=1}^{k} (-1)^{k+\ell} \frac{(C-1)!}{(C-k)!(j-1)!(k-j)!} v_j$$

for $k = 1, \ldots, C$. It follows that

$$P^{n-1} e_k = \sum_{j=1}^{k} (-1)^{k+j} \frac{(C-1)!}{(C-k)!(j-1)!(k-j)!} \left(\frac{j}{C}\right)^{n-1} v_j$$

and then for $n \geq 1$

$$\mathbb{P}(y_n = k \mid y_1 = 1) = \sum_{j=0}^{k} (-1)^{k+j} \frac{C!}{(C-k)!j!(k-j)!} \left(\frac{j}{C}\right)^n.$$

The above expression can be rewritten as

$$\mathbb{P}(y_n = k \mid y_1 = 1) = \frac{C!}{(C-k)!} \frac{S(n,k)}{C^n},$$

where $S(n,k)$ denotes the Stirling number of the second kind. By using the identity [114]

$$\sum_{m=0}^{n} \binom{n}{m} S(m,k) = S(n+1, k+1),$$

Equation (F.0.2) follows for $\ell = 1$. By using definition (4.2.2), Equation (F.0.2) is established for $\ell = 1$.

It is worth noting that that from the identity [114],

$$x^n = \sum_{k=0}^{n} S(n,k) \frac{\Gamma(x+1)}{\Gamma(x-k+1)},$$

with Γ denoting the Eurler's function, we have $\sum_{k=0}^{C} \mathbb{P}(y_n = k \mid y_1 = 1) = 1$.

In the same way, we have for any $\ell \leq k$

$$\mathbb{P}(y_n = k \mid y_1 = \ell) = e_\ell^t P^{n-1} e_k = \frac{(C-\ell)!}{(C-k)!} \sum_{j=\ell}^{k} \frac{(-1)^{k+j}}{(j-\ell)!(k-j)!} \left(\frac{j}{C}\right)^{n-1}$$
$$= \frac{(C-\ell)!}{(C-k)!C^{n-1}} \sum_{m=0}^{n-1} \binom{n-1}{m} S(m, k-\ell)\ell^{n-1-m}$$

and Equation (F.0.2) follows by using definition (4.2.2). □

To conclude this section, let us determine the mean value of the random variable y_n.

Lemma F.0.2. *The mean value* $\mathbb{E}(y_n \mid y_1 = 1)$ *is given by*

$$E(y_n \mid y_1 = \ell) = C\left(1 - \left(1 - \frac{\ell}{C}\right)\left(1 - \frac{1}{C}\right)^{n-1}\right). \tag{F.0.4}$$

Proof. By using the recurrence relation (F.0.1), we have

$$E(y_n \mid y_{n-1}) = 1 + \left(1 - \frac{1}{C}\right) y_{n-1}$$

and hence

$$E(y_n \mid y_1 = \ell) = \sum_{m=0}^{n-2} \left(1 - \frac{1}{C}\right)^m + \left(1 - \frac{1}{C}\right)^{n-1} \ell.$$

Equation (F.0.4) then easily follows.

From the above Lemma, we deduce the identity

$$\frac{1}{C^{n-1}} \sum_{k=0}^{C} \binom{C-\ell}{k-\ell} k\mathcal{A}_{n-1,k-\ell}(\ell) = C\left(1 - \left(1 - \frac{\ell}{C}\right)\left(1 - \frac{1}{C}\right)^{n-1}\right). \tag{F.0.5}$$

□

Bibliography

[1] James E Smith and Ravi Nair. The architecture of virtual machines. *Computer*, 38(5):32–38, 2005.

[2] Network Functions Virtualisation ETSI. Architectural Framework. Technical report, Technical Report ETSI GS NFV 002 V1.1.1, 2013.

[3] Pascal Potvin, Hanen Garcia Gamardo, Kim-Khoa Nguyen, and Mohamed Cheriet. Hyper heterogeneous cloud-based IMS software architecture: A proof-of-concept and empirical analysis. In *Smart City 360*, pages 250–262. Springer, 2016.

[4] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *Nsdi*, volume 11, pages 24–24, 2011.

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

[6] Fabrice Guillemin, Veronica Quintuna-Rodriguez, and Alain Simonian. Sojourn time in a M[X]/M/1 processor-sharing queue with batch arrivals. In *Stochastic Models Journal*. IEEE, 2018.

[7] Randolph Nelson, Don Towsley, and Asser N. Tantawi. Performance analysis of parallel processing systems. *IEEE Transactions on software engineering*, 14(4):532–540, 1988.

[8] Gerald J Popek and Robert P Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, 1974.

[9] David Clinton. Teach Yourself Linux Virtualization and High Availability. 2017.

[10] Gareth Roy, Andrew Washbrook, David Crooks, Gang Qin, Samuel Cadellin Skipsey, Gordon Stewart, and David Britton. Evaluation of containers as a virtualisation alternative for HEP workloads. In *Journal of Physics: Conference Series*, volume 664, page 022034. IOP Publishing, 2015.

[11] Redhat. Containers, What is a Linux Container. `https://www.redhat.com/en/topics/containers/whats-a-linux-container`, n.d. Accessed on 17.06.2018.

[12] James Lewis and Martin Fowler. Microservices, a definition of this new architectural term. `https://www.martinfowler.com/articles/microservices.html`, n.d. Accessed on 17.06.2018.

[13] Shahir Daya, Nguyen Van Duy, Kameswara Eati, Carlos M Ferreira, Dejan Glozic, Vasfi Gucer, Manav Gupta, Sunil Joshi, Valerie Lampkin, Marcelo Martins, et al. *Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach*. IBM Redbooks, 2016.

[14] Ivan Dwyer. Microservices, Patterns for building modern applications Iron.IO White Paper. `https://www.iron.io/docs/Iron.io_Whitepaper_Microservices%20Patterns%20for%20Building%20Modern%20Apps.pdf`, 2015. Accessed on 21.03.2018.

[15] Tomas Cerny, Michael J Donahoo, and Michal Trnka. Contextual understanding of microservice architecture: current and future directions. *ACM SIGAPP Applied Computing Review*, 17(4):29–45, 2018.

[16] ETSI. Network Functions Virtualization. `http://www.etsi.org/technologies-clusters/technologies/nfv`, 2012. Accessed on 02.05.2018.

[17] Farah Slim, Fabrice Guillemin, Annie Gravey, and Yassine Hadjadj-Aoul. Towards a dynamic adaptive placement of virtual network functions under onap. In *Third International NFV-SDN'17-O4SDI-Workshop on Orchestration for Software-Defined Infrastructures)*, 2017.

[18] Hendrik Moens and Filip De Turck. VNF-P: A model for efficient placement of virtualized network functions. In *Network and Service Management (CNSM), 2014 10th International Conference on*, pages 418–423. IEEE, 2014.

[19] 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.

[20] Milad Ghaznavi, Aimal Khan, Nashid Shahriar, Khalid Alsubhi, Reaz Ahmed, and Raouf Boutaba. Elastic virtual network function placement. In *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, pages 255–260. IEEE, 2015.

[21] Simon Oechsner and Andreas Ripke. Flexible support of VNF placement functions in OpenStack. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–6. IEEE, 2015.

[22] Roberto Riggio, Abbas Bradai, Tinku Rasheed, Julius Schulz-Zander, Slawomir Kuklinski, and Toufik Ahmed. Virtual network functions orchestration in wireless networks. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 108–116. IEEE, 2015.

[23] 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.

[24] David Dietrich, Ahmed Abujoda, and Panagiotis Papadimitriou. Network service embedding across multiple providers with nestor. In *IFIP Networking Conference (IFIP Networking), 2015*, pages 1–9. IEEE, 2015.

[25] 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.

[26] Stuart Clayman, Elisa Maini, Alex Galis, Antonio Manzalini, and Nicola Mazzocca. The dynamic placement of virtual network functions. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9. IEEE, 2014.

[27] Sevil Mehraghdam, Matthias Keller, and Holger Karl. Specifying and placing chains of virtual network functions. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 7–13. IEEE, 2014.

[28] Mayank Mishra and Anirudha Sahoo. On theory of VM placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 275–282. IEEE, 2011.

[29] Guilherme Thompson. *Stochastic Models for Resource Allocation in Large Distributed Systems*. PhD thesis, Université Pierre et Marie Curie, 2017.

[30] Fabrice Guillemin and Guilherme Thompson. Analysis of a trunk reservation policy in the framework of fog computing. *arXiv preprint arXiv:1604.03823*, 2016.

[31] Christine Fricker, Fabrice Guillemin, Philippe Robert, and Guilherme Thompson. Analysis of an offloading scheme for data centers in the framework of fog computing. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 1(4):16, 2016.

[32] Annie Gravey, J Boyer, K Sevilla, and Josee Mignault. Resource allocation for worst case traffic in ATM networks. *Performance Evaluation*, 30(1-2):19–43, 1997.

[33] Sanjoy K Baruah, Neil K Cohen, C Greg Plaxton, and Donald A Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.

[34] Hans Kroner, Gerard Hebuterne, Pierre Boyer, and Annie Gravey. Priority management in ATM switching nodes. *IEEE Journal on selected areas in communications*, 9(3):418–427, 1991.

[35] Raj Jain, Dah-Ming Chiu, and William R Hawe. *A quantitative measure of fairness and discrimination for resource allocation in shared computer system*, volume 38. Eastern Research Laboratory, Digital Equipment Corporation Hudson, MA, 1984.

[36] Patrick Poullie, Thomas Bocek, and Burkhard Stiller. A survey of the state-of-the-art in fair multi-resource allocations for data centers. *IEEE Transactions on Network and Service Management*, 15(1):169–183, 2018.

[37] Keshav Srinivasan. An Engineering Approach to Computer Networking, 1997.

[38] Peter Marbach. Priority service and max-min fairness. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 266–275. IEEE, 2002.

[39] Manolis Katevenis. Fast switching and fair control of congested flow in broadband networks. *IEEE Journal on selected Areas in Communications*, 5(8):1315–1326, 1987.

[40] Ellen L. Hahne. Round-robin scheduling for max-min fairness in data networks. *IEEE Journal on Selected Areas in communications*, 9(7):1024–1039, 1991.

[41] Ellen L Hahne. Round robin scheduling for fair flow control in data communication networks. Technical report, Massachusetts Inst of Tech Cambridge Lab for Information and Decision Systems, 1986.

[42] Leonard Kleinrock. Time-shared systems: A theoretical treatment. *Journal of the ACM (JACM)*, 14(2):242–261, 1967.

[43] Karl Sigman. Processor Sharing Queues. `http://www.columbia.edu/~ks20/4404-Sigman/4404-Notes-PS.pdf`, 2013. Accessed on 17.06.2018.

[44] Madhavapeddi Shreedhar and George Varghese. Efficient fair queuing using deficit round-robin. *IEEE/ACM Transactions on networking*, 4(3):375–385, 1996.

[45] Albert G Greenberg and Neal Madras. How fair is fair queuing. *Journal of the ACM (JACM)*, 39(3):568–598, 1992.

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

[47] Hoon Kim, Keunyoung Kim, Youngnam Han, and Sangboh Yun. A proportional fair scheduling for multicarrier transmission systems. In *Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th*, volume 1, pages 409–413. IEEE, 2004.

[48] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. Quincy: fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 261–276. ACM, 2009.

[49] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems*, pages 265–278. ACM, 2010.

[50] Apache Hadoop. Hadoop: Fair Scheduler. `https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/FairScheduler.html`, n.d. Accessed on 21.06.2018.

[51] Jeff Barr. Taking Massive Distributed Computing to the Common Man – Hadoop on Amazon EC2/S3. `https://cloud.google.com/hadoop/`, n.d. Accessed on 22.06.2018.

[52] Google Cloud. Apache Spark and Apache Hadoop on Google Cloud Platform Documentation. `https://cloud.google.com/hadoop/`, n.d. Accessed on 22.06.2018.

[53] Robert Sheldon. Oracle's cloud analytics platform comprises several tools. `https://searchoracle.techtarget.com/feature/Oracles-cloud-analytics-platform-comprises-several-tools`, n.d. Accessed on 22.06.2018.

[54] Wei Wang, Baochun Li, and Ben Liang. Dominant resource fairness in cloud computing systems with heterogeneous servers. In *INFOCOM, 2014 Proceedings IEEE*, pages 583–591. IEEE, 2014.

[55] Ali Ghodsi, Vyas Sekar, Matei Zaharia, and Ion Stoica. Multi-resource fair queueing for packet processing. *ACM SIGCOMM Computer Communication Review*, 42(4):1–12, 2012.

[56] Corinne Touati, Eitan Altman, and Jérôme Galtier. Utility Based Fair Bandwidth Allocation. In *In Proceedings of the IASTED International Conference on Networks, Parallel and Distributed Processing and Applications (NPDPA 2002*. Citeseer, 2002.

[57] Haïkel Yaïche, Ravi R Mazumdar, and Catherine Rosenberg. A game theoretic framework for bandwidth allocation and pricing in broadband networks. *IEEE/ACM Transactions On Networking*, 8(5):667–678, 2000.

[58] Veronica Karina Quintuna Rodriguez and Fabrice Guillemin. Performance analysis of resource pooling for network function virtualization. In *Telecommunications Network Strategy and Planning Symposium (Networks), 2016 17th International*, pages 158–163. IEEE, 2016.

[59] ONF Solution Brief. OpenFlow-enabled SDN and Network Functions Virtualization. *Open Netw. Found*, 2014.

[60] Ivan Stojmenovic and Sheng Wen. The Fog computing paradigm: Scenarios and security issues. In *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*, pages 1–8. IEEE, 2014.

[61] Tarik Taleb, Marius Corici, Carlos Parada, Almerima Jamakovic, Simone Ruffino, Georgios Karagiannis, and Thomas Magedanz. EASE: EPC as a service to ease mobile core network deployment over cloud. *Network, IEEE*, 29(2):78–88, 2015.

[62] Kostas Pentikousis, Yan Wang, and Weihua Hu. Mobileflow: Toward software-defined mobile networks. *Communications Magazine, IEEE*, 51(7):44–53, 2013.

[63] I Chih-Lin, Jinri Huang, Ran Duan, Chunfeng Cui, Jesse Xiaogen Jiang, and Lei Li. Recent progress on C-RAN centralization and cloudification. *Access, IEEE*, 2:1030–1039, 2014.

[64] Dirk Wubben, Peter Rost, Jens Steven Bartelt, Massinissa Lalam, Valentin Savin, Matteo Gorgoglione, Armin Dekorsy, and Gerhard Fettweis. Benefits and impact of cloud computing on 5G signal processing: Flexible centralization through cloud-RAN. *Signal Processing Magazine, IEEE*, 31(6):35–44, 2014.

[65] Alexander William Dawson, Mahesh K Marina, and Francisco J Garcia. On the benefits of RAN virtualisation in C-RAN based mobile networks. In *Software Defined Networks (EWSDN), 2014 Third European Workshop on*, pages 103–108. IEEE, 2014.

[66] Navid Nikaein. Processing radio access network functions in the cloud: Critical issues and modeling. In *Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services*, pages 36–43. ACM, 2015.

[67] Rui Wang, Honglin Hu, and Xiumei Yang. Potentials and challenges of C-RAN supporting Multi-RATs toward 5G mobile networks. *Access, IEEE*, 2:1187–1195, 2014.

[68] Michael Haugh. Examining Factors of the NFV-I Impacting Performance and Portability. 2015.

[69] Uwe Dötsch, Mark Doll, Hans-Peter Mayer, Frank Schaich, Jonathan Segel, and Philippe Sehier. Quantitative analysis of split base station processing and determination of advantageous architectures for LTE. *Bell Labs Technical Journal*, 18(1):105–128, 2013.

[70] Liqiang Liu and Vidyadhar G Kulkarni. Balking and reneging in M/G/s systems exact analysis and approximations. *Probability in the Engineering and Informational Sciences*, 22(03):355–371, 2008.

[71] Ishwari Singh Rajput and Deepa Gupta. A priority based round robin CPU scheduling algorithm for real time systems. *International Journal of Innovations in Engineering and Technology*, 1(3):1–11, 2012.

[72] R Pike and A Gerrand. Concurrency is not parallelism. *Heroku Waza*, 2012.

[73] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating system concepts essentials*. John Wiley & Sons, Inc., 2014.

[74] Leonard Kleinrock. *Queueing Systems*, volume I: Theory. Wiley Interscience, 1976.

[75] Leonard Kleinrock, Richard R Muntz, and E Rodemich. The processor-sharing queueing model for time-shared systems with bulk arrivals. *Networks*, 1(1):1–13, 1971.

[76] L Flatto et al. The waiting time distribution for the random order service $M/M/1$ queue. *The Annals of Applied Probability*, 7(2):382–409, 1997.

[77] Frank WJ Olver. *NIST handbook of mathematical functions hardback and CD-ROM*. Cambridge University Press, 2010.

[78] Navid Nikaein, Raymond Knopp, Florian Kaltenberger, Lionel Gauthier, Christian Bonnet, Dominique Nussbaum, and Riadh Ghaddab. OpenAirInterface 4G: an open LTE network in a PC. In *International Conference on Mobile Computing and Networking*, 2014.

[79] Veronica Quintuna Rodriguez and Fabrice Guillemin. VNF modeling towards the Cloud-RAN implementation. In *Networked Systems (NetSys), 2017 International Conference on*, pages 1–8. IEEE, 2017.

[80] Veronica Quintuna Rodriguez and Fabrice Guillemin. Towards the deployment of a fully centralized Cloud-RAN architecture. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2017 13th International*, pages 1055–1060. IEEE, 2017.

[81] Veronica Quintuna Rodriguez and Fabrice Guillemin. Performance analysis of VNFs for sizing cloud-RAN infrastructures. In *Network Function Virtualization and Software Defined Networks (NFV-SDN), 2017 IEEE Conference on*, pages 1–6. IEEE, 2017.

[82] China Mobile Research Institute. C-RAN, The Road Towards Green RAN, White Paper. `https://pdfs.semanticscholar.org/eaa3/ca62c9d5653e4f2318aed9ddb8992a505d3c.pdf`, 2011. Accessed on 21.03.2018.

[83] Gabriel Brown. Cloud-RAN, The Next-Generation Mobile Network Architecture, Huawei White Paper. `http://www-file.huawei.com/-/media/CORPORATE/PDF/mbb/cloud-ran-the-next-generation-mobile-network-architecture.pdf?la=en`, 2017. Accessed on 21.03.2018.

[84] Ericsson. Cloud-RAN. *White Paper*, 2015.

[85] Ericsson Telefonica. Cloud-RAN Architecture for 5G. *White Paper*, 2015.

[86] Navid Nikaein, Mahesh K Marina, Saravana Manickam, Alex Dawson, Raymond Knopp, and Christian Bonnet. OpenAirInterface: A flexible platform for 5G research. *ACM SIGCOMM Computer Communication Review*, 44(5):33–38, 2014.

[87] Navid Nikaein, Raymond Knopp, Florian Kaltenberger, Lionel Gauthier, Christian Bonnet, Dominique Nussbaum, and Riadh Ghaddab. OpenAirInterface: an open LTE network in a PC. In *Proceedings of the 20th annual international conference on Mobile computing and networking*, pages 305–308. ACM, 2014.

[88] Amarisoft. V-RAN LTE. `https://www.amarisoft.com/v-ran-2/`, 2018. Accessed on 21.03.2018.

[89] David López-Pérez, Akos Ladányi, Alpár Jüttner, Herve Rivano, and Jie Zhang. Optimization method for the joint allocation of modulation schemes, coding rates, resource blocks and power in self-organizing LTE networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 111–115. IEEE, 2011.

[90] Navid Nikaein, Raymond Knopp, Lionel Gauthier, Eryk Schiller, Torsten Braun, Dominique Pichon, Christian Bonnet, Florian Kaltenberger, and Dominique Nussbaum. Closer to cloud-RAN: RAN as a service. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 193–195. ACM, 2015.

[91] Erik Dahlman, Stefan Parkvall, and Johan Skold. *4G: LTE/LTE-advanced for mobile broadband*. Academic press, 2013.

[92] Islam Alyafawi, Eryk Schiller, Torsten Braun, Desislava Dimitrova, Andre Gomes, and Navid Nikaein. Critical issues of centralized and cloudified LTE-FDD radio access networks. In *Communications (ICC), 2015 IEEE International Conference on*, pages 5523–5528. IEEE, 2015.

[93] Leonard Kleinrock. *Queueing Systems*, volume II: Computer Applications. Wiley Interscience, 1976.

[94] Q Zheng, Y Chen, S Abeyratne, S Dreslinski, and T Mudge. "Revisiting Cloud RAN from a Computer Architecture Point of View", July 2016. [Online; posted July-2016].

[95] Aleksandra Checko, Henrik L Christiansen, Ying Yan, Lara Scolari, Georgios Kardaras, Michael S Berger, and Lars Dittmann. Cloud RAN for mobile networks technology overview. *IEEE Communications surveys & tutorials*, 17(1):405–426, 2015.

[96] Dario Sabella, Peter Rost, Yingli Sheng, Emmanouil Pateromichelakis, Umer Salim, Patricia Guitton-Ouhamou, Marco Di Girolamo, and Giovanni Giuliani. RAN as a service: Challenges of designing a flexible RAN architecture in a cloud-based heterogeneous mobile network. In *Future Network and Mobile Summit (FutureNetworkSummit), 2013*, pages 1–8. IEEE, 2013.

[97] 3GPP, 3rd Generation Partnership Project. *"Study on new radio access technology Radio access architecture and interfaces"*, 3 2017. V14.0.

[98] Jialong Duan, Xavier Lagrange, and Frederic Guilloud. Performance Analysis of Several Functional Splits in C-RAN. In *Vehicular Technology Conference (VTC Spring), 2016 IEEE 83rd*, pages 1–5. IEEE, 2016.

[99] Bin Guo, Wei Cao, An Tao, and Dragan Samardzija. LTE/LTE-A Signal Compression on the CPRI Interface. *Bell Labs Technical Journal*, 18(2):117–133, 2013.

[100] Dirk Wubben, Peter Rost, Jens Steven Bartelt, Massinissa Lalam, Valentin Savin, Matteo Gorgoglione, Armin Dekorsy, and Gerhard Fettweis. Benefits and impact of cloud computing on 5G signal processing: Flexible centralization through cloud-RAN. *IEEE signal processing magazine*, 31(6):35–44, 2014.

[101] Ericsson AB, Huawei Technologies Co. Ltd., NEC Corporation, Nokia. eCPRI Specification V1.0, 2017.

[102] IEEE, Draft Standard for Radio Over Ethernet Encapsulations and Mappings. Standard, IEEE, January 2018.

[103] xRAN Forum. Fronthaul working group, White paper, 2017.

[104] I Chih-Lin, Yannan Yuan, Jinri Huang, Shijia Ma, Chunfeng Cui, and Ran Duan. Rethink fronthaul for soft RAN. *IEEE Communications Magazine*, 53(9):82–88, 2015.

[105] Mauricio Iturralde, Anne Wei, Tara Ali Yahiya, and André-Luc Beylot. Resource allocation for real time services using cooperative game theory and a virtual token mechanism in LTE networks. In *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*, pages 879–883. IEEE, 2012.

[106] LTE, Evolved Universal Terrestrial Radio Access, Physical layer procedures (3GPP TS 36.213 version 12.4.0 Release 12). Standard, European Telecommunications Standards Institute, 2015.

[107] LTE, Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network(E-UTRAN), Overall description, Stage 2 (3GPP TS 36.300 version 13.6.0 Release 13). Standard, European Telecommunications Standards Institute, February 2017.

[108] Sourjya Bhaumik et al. CloudIQ: A framework for processing base stations in a data center. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 125–136. ACM, 2012.

[109] Syed Faraz Hasan. *Emerging trends in communication networks*. Springer, 2014.

[110] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM, 1967.

[111] Veronica Quintuna Rodriguez and Fabrice Guillemin. On dimensioning Cloud-RAN systems. In *ValueTools, 11th EAI International Conference on Performance Evaluation Methodologies and Tools*. EAI, 2017.

[112] V. Quintuna Rodriguez and F. Guillemin. Cloud-RAN Modeling Based on Parallel Processing. *IEEE Journal on Selected Areas in Communications*, 36(3):457–468, March 2018.

[113] M.V. Cromie, M.L. Chaudhry, and W.K. Grassman. Further results for the queueing systems $M^X/M/c$. *J. Opl Res. Soc.*, 30(8):755–763, 1979.

[114] John Riordan. *Combinatorial identities*. Wiley series in probability and mathematical statistics. Wiley, New York, 1968.

[115] William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, January 1968.

[116] b com. Wireless Edge Factory. `https://b-com.com/en/bcom-wireless-edge-factory`, n.d. Accessed on 24.06.2018.

[117] David R Butenhof. *Programming with POSIX threads*. Addison-Wesley Professional, 1997.

[118] Daniel P Bovet and Marco Cesati. *Understanding the Linux Kernel: from I/O ports to process management*. " O'Reilly Media, Inc.", 2005.

[119] Fabrice Guillemin and Jacqueline Boyer. Analysis of the $M/M/1$ queue with processor sharing via spectral theory. *Queueing Systems*, 39(4):377–397, 2001.

[120] Sem C Borst, Onno J Boxma, John A Morrison, and R NúñEz Queija. The equivalence between processor sharing and service in random order. *Operations Research Letters*, 31(4):254–262, 2003.

[121] Navid Nikaein, Eryk Schiller, Romain Favraud, Kostas Katsalis, Donatos Stavropoulos, Islam Alyafawi, Zhongliang Zhao, Torsten Braun, and Thanasis Korakis. Network store: Exploring slicing in future 5G networks. In *Proceedings of the 10th International Workshop on Mobility in the Evolving Internet Architecture*, pages 8–13. ACM, 2015.

[122] Sina Khatibi, Luísa Caeiro, Lúcio S Ferreira, Luis M Correia, and Navid Nikaein. Modelling and implementation of virtual radio resources management for 5G Cloud RAN. *EURASIP Journal on Wireless Communications and Networking*, 2017(1):128, 2017.

[123] TIP Project. Telecoom Infra Project (TIP), vRAN Fronthaul. `https://telecominfraproject.com/vran-fronthaul/`, n.d. Accessed on 27.02.2018.