5G RAN : physical layer implementation and network slicing

Aymeric de Javel

INSTITUT POLYTECHNIQUE DE PARIS

TELECOM Paris

IP PARIS

NNT : 2022IPPAT031

Thèse de doctorat

5G RAN : implémentation de la couche physique et découpage du réseau

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Réseaux, informations et communications

Thèse présentée et soutenue à Palaiseau le 30/09/2022, par

Aymeric de JAVEL

Composition du Jury :

Madame Thi-Mai-Trang Nguyen Professeure, Université Sorbonne Paris Nord, France	Président
Madame Lina Mroueh Professeure, ISEP, France	Rapporteur
Monsieur Xavier Lagrange Professeur, IMT Atlantique, France	Rapporteur
Monsieur Cédric Adjih Ingénieur de recherche, INRIA, France	Examinateur
Monsieur Jean-Louis Rougier Professeur, Télécom Paris, France	Directeur de thèse
Monsieur Philippe Martins Professeur, Télécom Paris, France	Co-Directeur de thèse
Monsieur Fabrice Bellard Amarisoft, France	Invité
Monsieur Patrice Nivaggioli Cisco, France	Invité

Remerciements

Je tiens à remercier l'ensemble des personnes impliquées de près ou de loin dans le travail présenté dans cette thèse. Cette thèse n'aurait jamais été possible sans eux.

Tout d'abord, mes remerciements vont aux deux rapporteurs de cette thèse, Madame Lina Mroueh et Monsieur Xavier Lagrange, pour le temps précieux qu'ils m'ont accordé.

Par ailleurs, je tiens à remercier mes deux directeurs de thèse, Jean-Louis Rougier et Philippe Martins pour leur temps, leurs nombreux et précieux conseils et leur support au cours de ces trois années. J'ai été très heureux et chanceux de préparer cette thèse sous votre direction. Vos propos toujours bienveillants mais exigeants m'ont permis de fournir le meilleur travail possible. J'ai énormément appris et grandi grâce à vous.

Un immense merci à l'ensemble des membres du laboratoire de Telecom Paris. Merci aux ingénieurs et stagiaires d'avoir partagé de si bons moments à vos côtés, qui m'ont permis de toujours avoir de l'énergie pour avancer. Un merci particulier à Jean-Sébastien, avec qui j'ai traversé au laboratoire la période particulière du Covid.

Merci à Cisco qui a financé ce travail. Merci à l'équipe ADT de m'avoir accueilli en alternance et suivi au cours de ma thèse. Vous m'avez permis de plonger dans le monde professionnel et avez rendu cette thèse possible. Un grand merci à Patrice de m'avoir encadré et suivi au cours de ces années.

Enfin, un grand merci à ma femme, mes enfants, ma famille et mes proches pour votre soutien au cours des ces trois années passionnantes mais difficiles. Je n'aurais pas été capabale de relever ce défi sans votre présence à mes côtes !

Contents

List of Figures

List of Tables

Acronyms

$5QI$ 5G QoS Identifier.

AMF Access and Mobility Management Function.

API Application Programmable Interface.

$AUSF$ Authentication Server Function.

BCH Broadcast Channel.

BG Base Graph.

BGP Border Gateway Protocol.

$BPSK$ Binary Phase Shift Keying.

BSR Buffer Status Report.

BWP Bandwidth Part.

$CBRA$ Contention Based Random Access.

CCE Control Channel Element.

$CFRA$ Contention Free Random Access.

CN Core Network.

$CORESET$ Control Resource Set.

$CPRI$ Common Public Radio Interface.

CPU Central Processing Unit.

CRC Cyclic Redundancy Check.

CU Centralized Unit.

$CUPS$ Control and User Plane Separation.

DCI Downlink Control Information.

$DCNR$ Dual Connectivity with New Radio.

$DL-SCH$ Downlink Shared Channel.

$DMRS$ Demodulation Reference Signal.

DRB Data Radio Bearers.

DU Distributed Unit.

$eCPRI$ enhanced Common Public Radio Interface.

$eMBB$ enhanced Mobile Broadband.

FDD Frequency Division Duplex.

FFT Fast Fourier Transform.

$FPGA$ Field Programmable Gate Array.

GBR Guaranteed Bit Rate.

GPU Graphics Processing Unit.

$GSCN$ Global Synchronization Channel Number.

$GTP-U$ GPRS Tunneling Protocol User plane.

$HARQ$ Hybrid Automatic Repeat Request.

$HTTP$ Hypertext Transfer Protocol.

$iFFT$ inverse Fast Fourier Transform.

IGP Interior Gateway Protocol.

IMS IP Multimedia Subsystem.

IP Internet Protocol.

IQ In-phase Quadrature.

KPI Key Performance Indicator.

$LCID$ Logical Channel ID.

$LDPC$ Low Density Parity Check.

$LFSR$ Linear Feedback Shift Register.

LLR Log-Likelihood Ratio.

LSB Least Significant Bit.

MAC Medium Access Control.

$MANO$ Management and Orchestration.

MCS Modulation and Coding Scheme.

MIB Master Information Block.

$MIMO$ Multiple Input Multiple Output.

$mMTC$ massive Machine Type Communications.

MSB Most Significant Bit.

NAS Non Access Stratum.

$NGAP$ Next Generation Application Protocol.

NSA Non Standalone.

NSSF Network Slice Selection Function.

OFDM Orthogonal Frequency Division Multiplexing.

OFDMA Orthogonal Frequency Division Multiple Access.

OS Operating System.

PBCH Physical Broadcast Channel.

PCF Policy Control Function.

PCI Physical Cell ID.

PDCCH Physical Downlink Control Channel.

PDCP Packet Data Convergence Protocol.

PDSCH Physical Downlink Shared Channel.

PDU Protocol Data Unit.

PHY Physical.

PPP Poisson Point Process.

PRACH Physical Random Access Channel.

PRB Physical Resource Block.

PSS Primary Synchronization Signal.

PUCCH Physical Uplink Control Channel.

PUSCH Physical Uplink Shared Channel.

QoS Quality of Service.

QPSK Quadrature Phase Shift Keying.

RA Random Access.

RAN Radio Access Network.

RAPID Random Access Preamble Identity.

RAR Random Access Response.

RB Resource Block.

RE Resource Element.

REG Resource Element Group.

RF Radio Frequency.

RIC RAN Intelligent Controler.

RIV Resource Indicator Value.

RLC Radio Link Control.

$RNTI$ Radio Network Temporary Identifier.

RRC Radio Resource Control.

RTP Real-time Transport Protocol.

RU Radio Unit.

SA Standalone.

SCS Subcarrier Spacing.

SD Slice Differentiator.

$SDAP$ Service Data Adaptation Protocol.

SDR Software Defined Radio.

SDU Service Data Unit.

SFN Sequence Frame Number.

$SIB1$ System Information Block 1.

$SINR$ Signal over Interference plus Noise Ratio.

SLA Service Level Agreement.

$SLIV$ Start and Length Indicator Value.

SMF Session Management Function.

SMP Symmetrical Multi-Processing.

SR Scheduling Request.

SSB Synchronization Signal Block.

SSS Secondary Synchronization Signal.

SST Slice Service Type.

TA Timing Advance.

TDD Time Division Duplex.

TSN Time Sensitive Network.

UDM Unified Data Management.

UDR Unified Data Repository.

UE User Equipment.

$UECRI$ UE Contention Resolution Identity.

$UL-SCH$ Uplink Shared Channel.

UPF User Plane Function.

$uRLLC$ ultra Reliable Low Latency Communications.

VM Virtual Machine.

$VoIP$ Voice over IP.

VPN Virtual Private Network.

Introduction

While 4G focuses on mobile internet access, 5G addresses a wide range of business use cases (called verticals), ranging from industry 4.0 to smart cities and augmented reality. Those business verticals can be grouped into three prominent use cases: enhanced Mobile Broadband (eMBB), ultra Reliable Low Latency Communications (uRLLC) and massive Machine Type Communications (mMTC). Aggregating those use cases on a single physical infrastructure while respecting the associated constraints is required to optimize deployments and resource consumption. The different use cases have antagonist network requirements. Indeed, while eMBB concentrates on throughput, uRLLC focuses on latency, and mMTC has to support a huge amount of connected devices. Those constraints, and more specifically latency and throughput, are not compatible as increasing capacity often implies a loss in delays, and reciprocally, latency-focused systems are struggling to provide high data rates. In those conditions, it has to be expected that naive base stations will not be able to serve multiple vertical constraints. On the other side, deploying independent physical infrastructures for every vertical will have an insufferable economic and ecological cost.

In this context, a new technology is required to support the heterogeneous terminals on top of a single infrastructure. This technology is called network slicing and describes the ability of a network to operate different verticals and use cases, called network slices, on a single physical infrastructure. A slice represents a logical network that connects terminals with close network constraints to one or multiple data networks. It is associated with a set of resources. Depending on the terminals connecting a slice, a set of requirements and constraints (like throughput, delay, or reliability) is defined.

This technology is expected to provide two critical features for managing different usages. The first one is deployment optimization and flexibility, as it would enable one to deploy multiple verticals on top of a shared physical infrastructure. In addition, it would enable extreme energy consumption and cost gain, which are the two main concerns while deploying mobile networks. On the other side, network slicing must also grant isolation between slices so that different slices Service Level Agreement (SLA) are respected.

Network slicing has received much attention in the literature. On the Core Network (CN) side, network slicing can be achieved using virtualization methods and cloud technologies such as containers. Such virtualization techniques bring a lot of flexibility and agility to the deployment scenarios, options, and parameters. Virtualization enables the operator to easily deploy different logical networks on a single infrastructure while having a minor economic impact. It also provides flexibility, so network instances serving slices with low latency constraints can be deployed closed to base stations on the edge, and other network instances can be gathered in the cloud for better optimization. In this area, research projects mainly focus on orchestration and monitoring as virtualization flexibility can only be achieved with solid orchestration methods. Such methods enable online function deployment and resizing when traffic varies. The second main challenge for deploying CN instances is function placement. Every combination of functions placement will have a cost while deploying a network instance. Deploying functions on the edge might reduce delays but increase deployment complexity, whereas cloud deployments optimize the deployment process and network centralization while increasing delays. Other constraints such as resources' availability can also be considered while computing the deployment cost.

Nevertheless, on the Radio Access Network (RAN) side, it can be pointed out that there is a considerable lack of technologies for network slicing. No algorithms and methods guarantying isolation between slices and requirements fulfillment have been proposed. Especially, systems that handle high throughput and low latency communications are missing. The work introduced in this thesis emerges

from this statement and focuses on the RAN side. Indeed, at the RAN level, network slicing requires the design of new methods and algorithms and the application of virtualization technologies to RAN systems, which has never been done before. In the current state of the art, it can be supposed that network slicing will be achieved by provisioning and associating resources to slices or a group of slices sharing the same requirements in terms of latency and throughput. In this context, resource allocation appears to be critical for network slicing optimization to maximize network efficiency while respecting slices requirements.

This thesis addresses two primary challenges raised by network slicing of the RAN. The first one is that resource allocation requires the definition of network models which consider both diverse and heterogeneous RAN constraints. Beyond capacity and User Equipment (UE) density, which can be easily modeled, latency and reliability are critical parameters for such models and cannot be trivially studied. Many models already exist for resource allocation of the RAN. However, models which take into account network slicing constraints do not exist. Moreover, the impact of network slicing on coverage has not been studied yet, while it is one critical aspect of network reliability. One contribution of this thesis is, therefore, to propose a RAN network slicing model. It takes into account new constraints brought by network slicing, among which coverage is of the utmost importance.

The second main challenge raised by network slicing is the supervision of resource allocation. Indeed, as long as the physical infrastructure is shared between the different slices, it is impossible to guarantee complete control of the network and respect of the requirements even if the resources are carefully allocated. This is a big issue for network slicing as some verticals have ultra-high network control requirements. In this context, supervision systems are required to bring a second level of control over the network. Supervision tools aim to control that the constraints of the different slices can be respected with the current state of the system through the analyze of the data extracted from the network. The data can be provided by network's Application Programmable Interface (API), but this is not reliable concerning potential cyber-attacks. In this thesis, we propose using a data extraction system external to the network (called a probe). Therefore, another contribution of this thesis is the implementation of a prototype of such a probe.

Given that the probe is outside the network, it must decode the different signals of a cell in order to extract data. The data extraction mechanism that has to be implemented corresponds to the receiver's side of a UE's Physical (PHY) layer for decoding downlink transmissions and to the receiver's side of a gNodeB's PHY layer for decoding uplink transmissions. Therefore, the probe can be derived from a 5G PHY layer. The last contribution of this work is the introduction of an open-source 5G physical layer from which the probe is derived. This physical layer is designed with regard to two main features. The first one is network slicing which implies that the PHY layer implementation must be modular so that it can be adapted to different kinds of slices. The second one is functional split. Indeed, in 5G, the different layers of the RAN can be split and dis-aggregated over the network. The PHY layer is therefore built with an architecture which enables future implementation of RAN splitting. This project is named *free5GRAN* and is publicly available on Github [7].

This work starts by introducing the 5G system. Then, it is split into two parts. The first one details the implementation of the 5G physical layer, including the procedures and functions, the algorithms and the software architecture. The second part investigates network slicing itself by reviewing the concepts and associated resources and introducing a model for network slicing and a probe for network supervision.

Chapter 1

Introduction to the 5G system

Contribution

In this chapter, we give an overview of the 5G system. We provide the reader with a global understanding of what 5G is and how the different Quality of Service (QoS) flows are managed across the 5G system. Therefore, this chapter first introduces the whole 5G system and then focuses on the RAN.

1.1 Introduction

In this chapter, an introduction to the 5G system is given. The objective is two-fold. First, it aims to provide the reader with a global understanding of the 5G system: its architecture and main technologies. Furthermore, it also aims at introducing some specific components and functions that will be further discussed later so that the reader has all the keys to understand this work.

The 5G system can be introduced in many different ways. This chapter introduces it as the network infrastructure designed to support a wide variety of terminals. Indeed, in 4G, the network is built to provide end-users with data services like voice and video streaming or web traffic, whereas, in 5G, the network is built to serve highly heterogeneous terminals like connected vehicles, sensors, or smartphones. The main difference between the different user traffics is not the associated QoS profile but rather the data network to which the service requires access. Indeed, a connected car might require access to the internal manufacturer's network, whereas a sensor for agriculture might connect to the cloud for data collection. Therefore, in 5G, the user traffic is first handled based on the data network to which a terminal requires access and then based on the QoS profile required by the service. When accessing the network, a terminal is given a tunnel that directly connects the terminal to the data network. The network is responsible for setting up the tunnel and supporting different services with different QoS profiles within a single tunnel.

Besides the variety of terminals, multiple use-cases can be addressed in 5G. Those usages are a combination of three fundamental use-cases that are eMBB, uRLLC and mMTC. First, eMBB is the most common usage and corresponds to the natural evolution of 4G networks. It is designed to enable end-users to access the network with high throughput, making applications such as augmented reality possible. Moreover, uRLLC is designed to support critical applications such as remote surgery or autonomous vehicles. Those applications require strong network reliability and low delay communications. Finally, mMTC is a use-case where the network is used to connect millions of devices. The network must be able to serve a huge number of UEs, and the communication must be highly optimized in terms of power consumption. Those three fundamental use-cases delimit a triangle in which all the other use-cases can be found. All the 5G use-cases can be seen as a combination of those three fundamental use-cases. The triangle and the different use-cases are represented in Figure 1.1. All the terminals connected in 5G are associated with a use case.

The first section of this chapter gives an overview of the 5G standalone architecture, the second one describes the RAN splitting concept and the last one introduces Software Defined Radio (SDR).

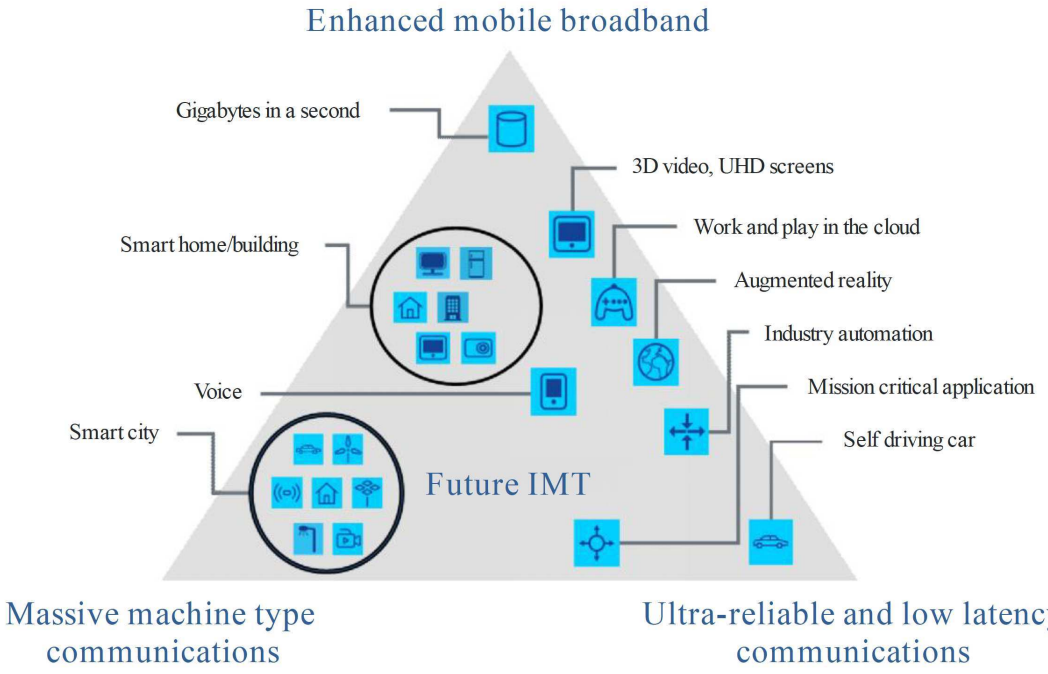


Figure 1.1: 5G usages scenarios (from ITU-R IMT 2020 requirements)

1.2 A quick introduction to virtualization

Before describing the 5G network architecture and functions, it is crucial to understand the core concept of virtualization. Indeed, one of the main evolution from 4G to 5G is the extensive use of virtualization technologies at the different levels of the network (both CN and RAN). The main concepts and terms that are used in this thesis and related to virtualization are:

- Virtualization. First, the virtualization concept itself refers to the software implementation of functions usually implemented in hardware. The main impact of virtualization is that functions are no longer correlated to the underlying hardware as the software implementations are designed to be deployed on generic hardware platforms.
- Microservice. When functions are virtualized, they are commonly developed as microservice software applications. A microservice architecture refers to the split of one big function into indivisible sub-functions that are implemented in separate software applications and work with each other to implement the big function behavior.
- Container. Containers are one way to manage microservice applications deployment, orchestration and life-cycle management. A container is a standalone environment containing all the assets the embedded application requires. In addition, this environment is seamlessly portable across devices that support containers. There are several container management systems for cloud environments, among which *Kubernetes* is the most used. It enables to define groups of resources and services, and automates containers deployment, replication, and life cycle management to provide a given service using a set of resources.
- Cloud-native networking. It refers to the virtualization of network functions usually implemented in physical nodes. For example, in Internet Protocol (IP) networks, cloud-native networking refers to the virtualization of the routing and switching functions usually implemented in routers and switches.

1.3 5G standalone system overview

The 5G system is made of two main parts that are the CN and the RAN. Those two parts work together to build end-to-end tunnels between UEs and data networks. The RAN is responsible for the radio connection with UEs and the CN is responsible for network control and user traffic routing. In this section, an introduction to the 5G CN and RAN is given. Figure 1.2 represents the overall 5G system architecture (only the main components of the CN are represented).

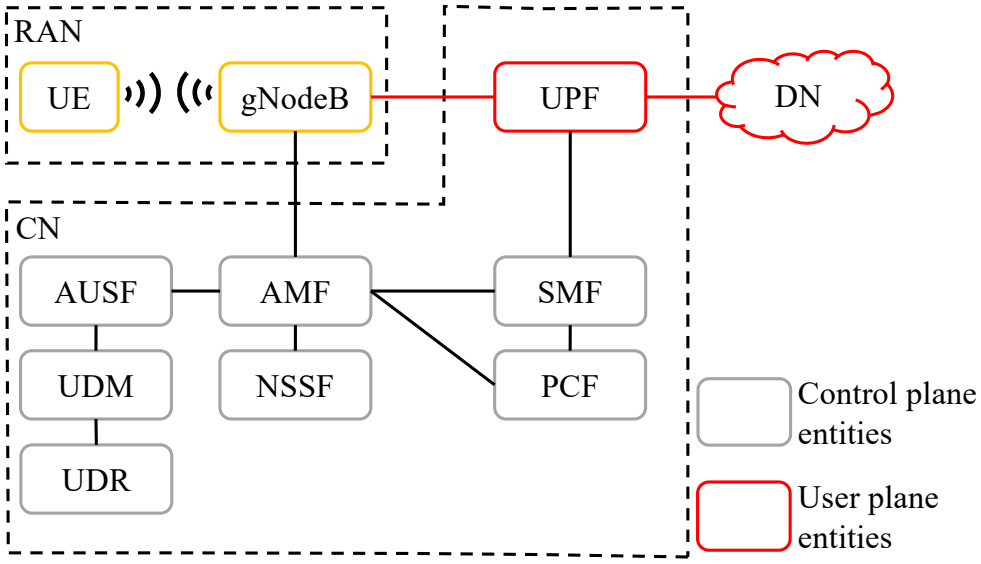


Figure 1.2: 5G network architecture

The rise of virtualization techniques in the past years has introduced a significant evolution in 5G. Indeed, in 4G, the network infrastructure is hardware-centric, meaning that the functions must be grouped and implemented in hardware devices, and the architecture aims to distribute the different functions alongside the network to provide the best service quality. The common infrastructure must adjust to the different services. In 5G, the network functions are implemented in software and can be deployed with a much higher level of flexibility and optimization, and some functions can be dedicated to specific services. The infrastructure can, therefore, be adjusted to the different services.

1.3.1 Core Network

On the CN's side, different functions are required for network control and user traffic routing. There are three main differences between the 4G and the 5G CN:

- The first is the virtualization of the 5G CN. The different functions are microservice software applications. This increases the flexibility of the CN as it can be easily deployed and managed.
- The second one is that the different functions are split between the network control and the user traffic handling functions, which is called Control and User Plane Separation (CUPS). This separation is defined in the latest version of the 4G CN and is native in 5G. The CUPS architecture enables sharing the control functions between the different services and dedicating user plane functions to services. Therefore, the infrastructure keeps a consistent overview of the network, but the user traffic functions can be adjusted to the different services.
- The last main evolution is that two connection types between the CNs elements have been defined. The first is the reference point connection type, which is the natural evolution of the 4G CN, where each connection from one CN function to another has its interface. The second one, which is new in 5G, is the service-based architecture where all the network control functions are connected to a single network and communicate with each other through REST API calls.

The only user plane function is the User Plane Function (UPF). It is responsible for user traffic forwarding from the data network to the UE and from the UE to the data network. The user traffic is routed through the network from the gNodeB to the UPF using GPRS Tunneling Protocol User plane (GTP-U) tunnels. In the downlink direction, UPF must be able to match incoming traffic from the data network to the corresponding QoS flow. In uplink direction, UPF selects the data network on which the packets from one QoS flow must be transmitted. Multiple UPFs can be deployed in a single network to dedicate this function to one type of service and one data network.

The network control functions are:

- Access and Mobility Management Function (AMF): This is the entry point of the network for all the UEs. It is responsible for:
  - Network connection: setting up the Non Access Stratum (NAS) connection between the UE and the AMF for future communications.
  - Network registration: manage registration procedure with the UE. This procedure enables the authentication of the UE to the network and the authentication of the network to the UE.
  - Mobility: manage the evolution of the UE within the network. This guarantees that the network always knows where the UE is located for paging.
  - Next Generation Application Protocol (NGAP) session management: it forwards the NAS communications between the different components of the network and the UE. This is specially used for Protocol Data Unit (PDU) session establishment, which is the tunnel used by UEs for user traffic transfer.
- Authentication Server Function (AUSF): It is responsible for handling authentication procedure. It acts as a proxy between the AMF and the Unified Data Management (UDM).
- UDM: It is responsible for managing all the user data and for generating the authentication vectors used by the AUSF for network authentication.
- Unified Data Repository (UDR): It is the network's database.
- Session Management Function (SMF): It is responsible for all the procedures related to user traffic, including:
  - UPF selection.
  - IP address allocation.
  - PDU session and GTP-U tunnel establishment and management. PDU session and GTP-U tunnel are used for user traffic routing through the transport network until the data network and is also responsible for QoS flows management.
- Policy Control Function (PCF): It is responsible for managing network policies like QoS, service access etc. It interacts with the SMF and AMF for determining which services can be accessed by UEs. Moreover, it is responsible for helping the network components to select the QoS profiles that can be used by UEs.
- Network Slice Selection Function (NSSF): It is responsible for selecting the slice to which a UE can access.

One principal aspect of the 5G CN is that it is the first telecommunication technology designed as a cloud-native network. Indeed, the microservice architecture, as well as the service-based interface, make that the functions can be managed like any other cloud application:

- The microservice architecture makes it easy to virtualize the CN into small software functions. Those functions can be deployed and managed using cloud technologies like containers and *Kubernetes*.

- Moreover, the service-based interface enables to consider the communications between the different components of the CN to be like standard and widely used application-to-application communications. Many network technologies already exist to support those communications.

Deploying the 5G CN as a cloud-native network brings two main assets. The first one is increased flexibility and optimization. Indeed, it is much easier to manage a set of cloud applications than a set of hardware devices (as it was the case for previous generations). The resources can be shared between the different components of the network, which enables a better optimization. The second main advantage is that it becomes possible to deploy on-the-fly dedicated 5G CN functions. This is especially precious regarding the number of heterogeneous use-cases that the 5G network could serve: dedicating CN instances to a given use-case and efficiently managing the associated resources is a crucial point to make those use-cases possible.

1.3.2 Radio Access Network

The RAN is responsible for radio communications between UEs and gNodeB. Given that the 5G network offers many functionalities, it requires a large set of protocols for transmitting user data and control information.

1.3.2.1 RAN protocol stack

The 5G RAN protocol stack is not the same for control and user plane. Figure 1.3 represents the protocol stack for the control plane whereas Figure 1.4 represents the protocol stack for user traffic. It can be seen that on the UE's and gNodeB's sides, the protocol stack remain the same from PHY to Packet Data Convergence Protocol (PDCP).

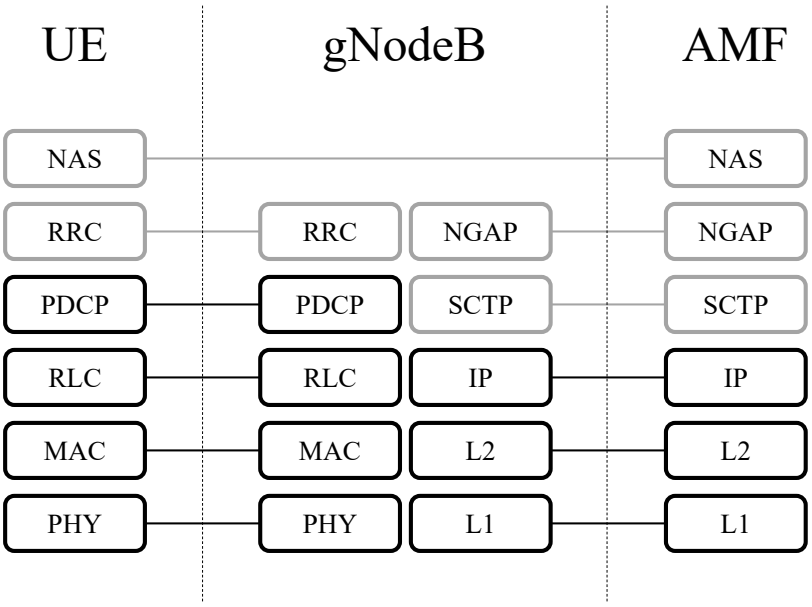


Figure 1.3: 5G RAN protocol stack for network control

Table 1.1 introduces the different layers of the control and user plane protocol stacks.

1.3.2.2 Air interface

1.3.2.2.1 Operating bands: In 5G, the operating frequency range is split into two parts. The first one, Frequency Range 1 (FR1), ranges from 410MHz to 7.125GHz, and the second one noted FR2 ranges from 24.25GHz to 52.6GHz. FR1 is the natural evolution of the frequency range defined for 4G, whereas FR2 brings millimeter waves to 5G. Multiple bands are defined within those frequency ranges and have a specific configuration (like operating frequency, multiplexing mode, or radio configuration). The operating bands are defined in Tables 5.2-1, and 5.2-2 of TS38.104 [8].

Layer	Description
PHY	The physical layer is responsible for radio communications from the gNodeB to the UE. This includes channel coding, modulation mapping, Orthogonal Frequency Division Multiplexing (OFDM) modulation, and signal transmission.
MAC	The Medium Access Control (MAC) layer has two main purposes. The first one is to ensure the data transmission is successful and to perform re-transmission if it is not the case by implementing Hybrid Automatic Repeat Request (HARQ). The second one is to multiplex and prioritize the different traffics that are being transmitted.
RLC	Radio Link Control (RLC) is responsible for segmentation and concatenation of packets for transmission. It is also responsible for allocating the packets to one or another radio bearer. A radio bearer is the equivalent of a tunnel on the radio interface and can be dedicated to a specific QoS profile. Finally, depending on the transmission mode, it can implement automatic repeat request for packet re-transmissions in case of errors.
PDCP	It is responsible for packets compression, ciphering and integrity protection.
RRC	Radio Resource Control (RRC) is used by the gNodeB for PHY layer control. It enables the configuration of the different PHY layer functionalities like beamforming or Multiple Input Multiple Output (MIMO). Furthermore, RRC signaling is used for different purposes like RRC connection management or paging.
NAS	NAS is the protocol that handles the communications between the UE and the AMF. It carries the user's authentication, registration, and mobility management.
SDAP	It is responsible for multiplexing the different QoS flows on the RAN side.
IP	This is the actual user traffic. In 5G, other kinds of traffic are possible, like ethernet or raw packets, but the most usual one remains IP.

Table 1.1: RAN protocol stack layers

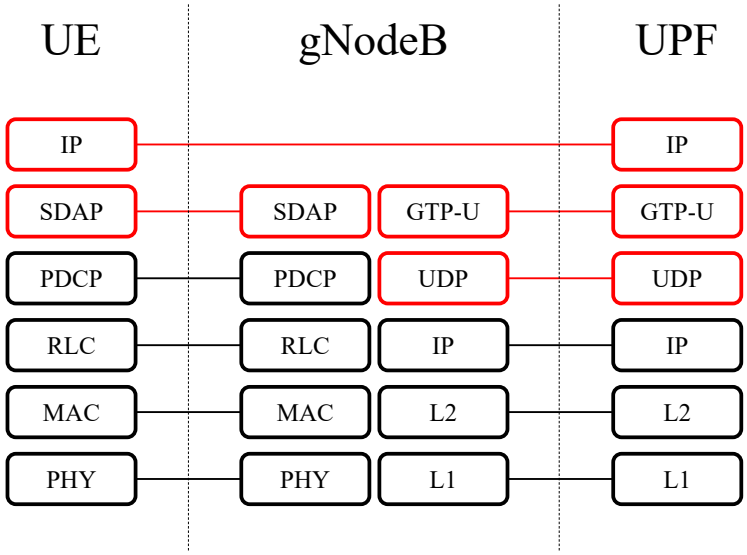


Figure 1.4: 5G RAN protocol stack for user traffic

1.3.2.2.2 OFDM: The 5G physical layer uses two resources, time and frequency bandwidth, to transmit data from a transmitter to a receiver. In order to share the resources between the UEs, the available time and bandwidth resources are split into small elements that are called Resource Element (RE):

- In the frequency domain, 5G uses OFDM. The core concept of OFDM is to divide the available bandwidth into sub-bandwidths that are orthogonal to each other and are called subcarriers. Given that the subcarriers are orthogonal to each other, they can be placed contiguously so that the spacing between subcarriers is also the width of a subcarrier. In OFDM systems, the width of a subcarrier is denoted the Subcarrier Spacing (SCS). As a subcarrier can be small regarding the cell's bandwidth, the Resource Block (RB) is defined as a set of 12 contiguous subcarriers.

- In the time domain, the resources are split into radio frames, subframes, slots, and symbols. A radio frame is 10ms long and comprises 10 subframes that are 1ms long. The OFDM symbol is the smallest time-domain unit. Its duration is nearly equal to one period of a signal at the SCS frequency. To avoid interference due to spread delay, all the OFDM symbols are added a cyclic prefix in front of them, which is a copy of the end of the symbol. There are 14 symbols per slot. Depending on the SCS, the number of slots per subframe varies.

The RE is defined as being one subcarrier over one OFDM symbol. The available resources can be seen as a time and frequency grid (called OFDM grid), as presented in Figure 1.6. The RE is one element of this grid.

1.3.2.2.3 Flexible numerology: In 4G, SCS is fixed to 15kHz whereas in 5G it can be either 15kHz, 30kHz, 60kHz, 120kHz or 240kHz. This is called flexible numerology and is the major evolution of the radio interface from 4G to 5G. Each time the SCS is multiplied by two, the symbol duration is divided by two. Different SCS and symbol duration can answer different use-cases, where higher SCS can be used for short-delay communications and lower SCS can be used for standard communications. Especially, low SCS narrows frequency selectivity and decreases computation complexity at the receiver. It can be used for UEs with energy consumption constraints. On the opposite, 5G systems may be mainly deployed in Time Division Duplex (TDD), and the downlink to uplink switching can be done at the slot level or even at the symbol level in case of dynamic scheduling. This means that the shorter the symbol is (and thus, the higher the SCS), the finer the downlink to uplink switching can be, and this is an optimization that can be done for low latency communications.

As different numerologies cannot be mixed on a single bandwidth, Bandwidth Part (BWP) has been defined. A BWP is a sub-bandwidth of the cell bandwidth with a given SCS and symbol duration.

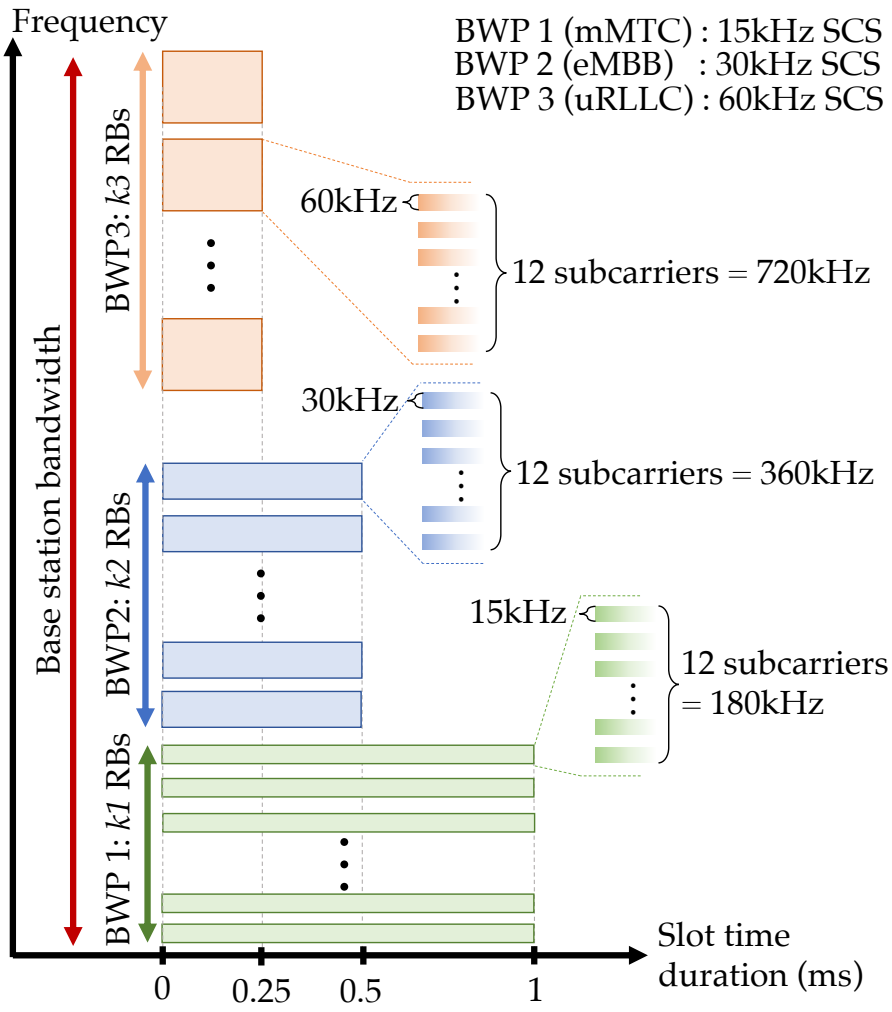


Figure 1.5: Bandwidth Parts, Resource Blocks and subcarrier Spacing

Each UE can be allocated to one or more BWP, but can be active on only one BWP simultaneously. UEs sharing the same constraints in terms of latency and energy consumption might share the same BWP. Figure 1.5 and 1.6 represent BWPs and radio frame structure in FR1 bands (bands below 7.125Ghz). In those diagrams, there is three BWPs allocated to three verticals that are mMTC, eMBB and uRLLC. For mMTC, a SCS of 15kHz is applied to narrow the frequency selectivity. For uRLLC, the maximum SCS, 60kHz is applied and finally, for eMBB, an intermediate SCS of 30kHz is applied. In the first one, we present the base station bandwidth which is split into the three BWPs. Each BWP is composed of an integer number of RBs (i.e. 12 subcarriers). Given that the symbol duration is inversely proportional to the SCS and that a slot is made of 14 symbols, each time the SCS is mutliplied by two, the slot duration is divided by two. Thus, the slot duration is 1ms for 15kHz SCS, 0.5ms for 30kHz and 0.25ms for 60kHz. In the second diagram, we present the REs of the three BWPs. For the eMBB and mMTC BWPs, the DL/UL switching is long and set to 0.5ms. There is 1 slot (for eMBB) and 0.5 slot (for mMTC) of downlink data followed by a guard period and then by uplink symbols on the next slot / half slot. Between eMBB and uRLLC BWPs, there is a guard band as the DL/UL switching is not the same. This guard band prevents downlink symbols of the two BWPs from interfering with uplink symbols of the other one. For the uRLLC BWP, the DL/UL switching is as quick as possible as it is done at the symbol level. One downlink symbol is followed by an uplink symbol, with a guard symbol between them. The guard symbol prevents downlink symbols spread delay from interfering with uplink symbols.

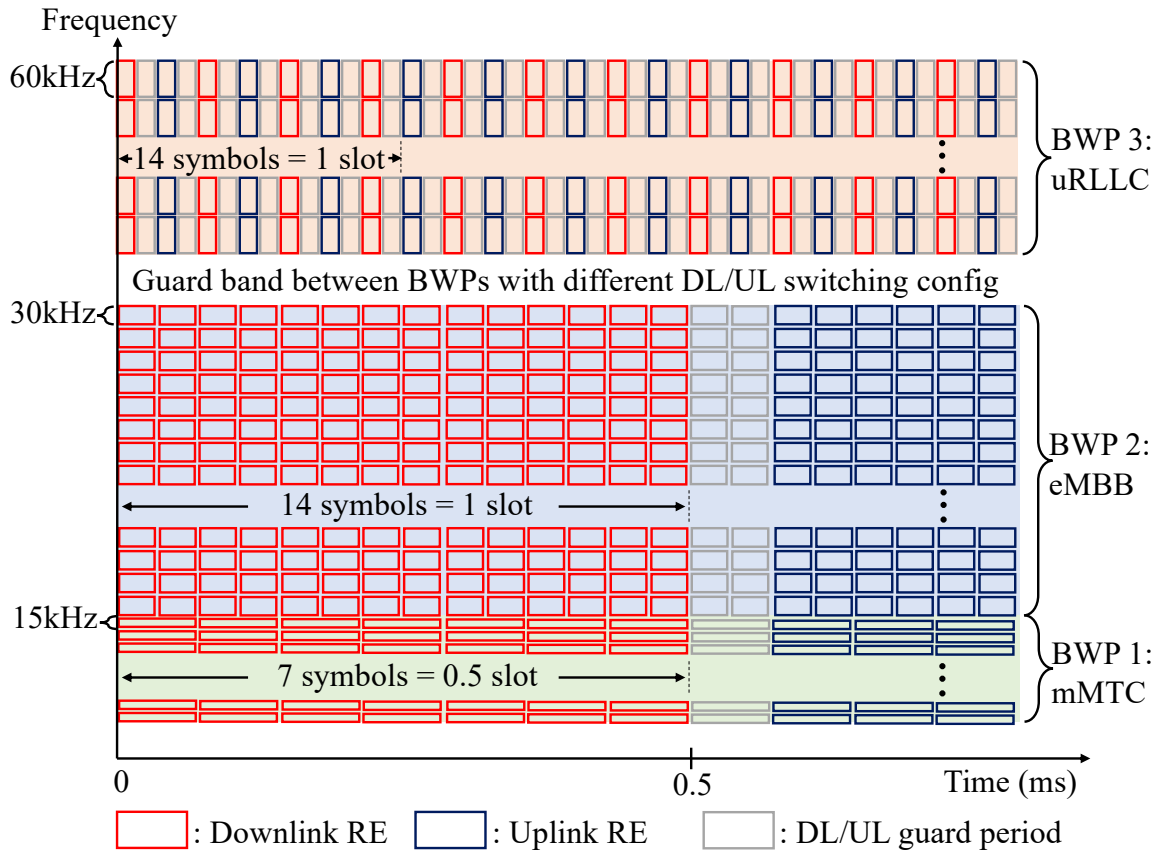


Figure 1.6: 5G radio frame structure

1.3.2.3 5G PHY layer channel coding

1.3.2.3.1 Polar codes: Polar codes in a channel coding technique introduced by Arikan in 2009 [9]. It is a simple method that consists in polarizing the channel. It is used in 5G for broadcast and control channels: Broadcast Channel (BCH) and Downlink Control Information (DCI). The specificity of this method is that it is theoretically possible to reach Shannon capacity. A good introduction to this method is given in [10].

1.3.2.3.1.1 Polar transform: The first step toward channel polarization is polar transform. It is a procedure that consists in multiplying an input bit sequence (noted c) of size N by a matrix G_N. This matrix is generated from a generator matrix G_2:

$$G_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

Where G_N is the n-th Kronecker power of G_2, with $N = 2^n$. For example:

$$G_4 = \begin{pmatrix} 1 \cdot \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} & 0 \cdot \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \\ 1 \cdot \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} & 1 \cdot \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

The output of polar transform when $N = 4$ for message c ($\{c_0, c_1, c_2, c_3\}$) is $c \cdot G_4 = \{c_0 + c_1 + c_2 + c_3, c_1 + c_3, c_2 + c_3, c_3\}$. The sum is made modulo 2, which corresponds to a XOR operation.

This polar transform operation can be done for any $N = 2^n$.

As represented in Figure 1.7, polar transform can also be understood as a binary tree. The leaves of the tree correspond to the input message c and at each parent node (noted p), the two child nodes (noted n_l for left child and n_r for right child) are combined such as $p = [n_l + n_r, n_r]$. The output of

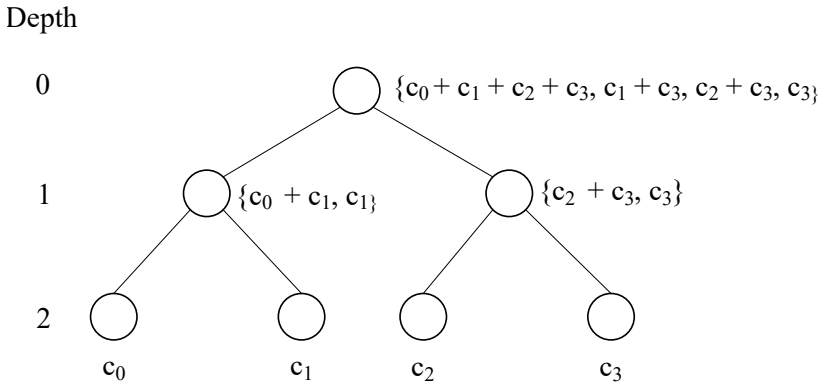


Figure 1.7: Polar transform binary tree representation

the polar transform is obtained at the root node (node which does not have a parent). The depth of one node is the layer in the binary tree, starting from the root node. Figure 1.7 shows the binary tree representation of polar transform operation for $N = 4$. It can be seen that the combination obtained at the root node is equal to $c \cdot G_4$. Binary tree representation is only a graphical way of understanding the polar transform, but it is critical to understand the decoding algorithm.

1.3.2.3.1.2 Channel polarization: Channel polarization comes from the statement that when applying polar transform, the channel is polarized. Indeed, when decoding the received channel after going through a noisy channel, it is observed that some bits are nearly never well decoded and others are nearly always decoded. After polar transform, the channel becomes either very bad or very good.

An intuitive way to explain the polarization is that it can be observed in the previous paragraph that when applying polar transform, some input bits are transmitted with much higher energy than others. In the previous example, bit 3 is transmitted with 4 times more energy than bit 0 as c_3 is present in the 4 output bits of polar transform, whereas bit c_0 is only present once.

It is possible to determine what are the positions where the channel will become very bad and what are the positions where the it will become very good. This can be done by performing simulations with different kinds of radio channels. For a given sequence of N bits, the most reliable and less reliable positions can be determined based on those simulations. When the input sequence size is N, all the input bits are placed in positions where the channel will become very good but also in positions where the it becomes very bad.

The concept of polar coding is to use polar transform for input sequences whose size (noted K) is smaller than N. Therefore, the K input bits can be placed in the most reliable positions of the channel, and the other $N - K$ bits are frozen (set to 0). Therefore, the reliability of the transmission depends on the ratio between N and K.

In 5G, polar coding is defined in TS28.212 section 5.3.1 [11] and the array of the most reliable positions is given in Table 5.3.1.2-1 for an input bit sequence of size $N = 1024$ (which is the maximum input size): the bits positions are sorted in order of reliability. The array of the most reliable bit for other sequence lengths can be deduced from this table.

1.3.2.3.2 Low Density Parity Check: Low Density Parity Check (LDPC) is a method that assigns a set of parity check bits to an input sequence to strengthen the input bit sequence. The parity bits can be used at the receiver's side to recover the initial input sequence. The output of the LDPC coder contains the systematic bits and the computed parity bits. It was first introduced in 1960 by Gallager [12]. In 5G, LDPC are used for encoding the data channels that are the Downlink Shared Channel (DL-SCH) and Uplink Shared Channel (UL-SCH).

1.3.2.3.2.1 Parity check concept: A parity bit is a bit which is computed depending on a sequence of input bits so that the final sequence (input and parity bits) is even. For example, if the

input size if 3 and there is one parity bit, then $[0, 1, 0]$ would be given a parity bit 1 (as $0+1+0+1 = 0 \pmod 2$) and $[0, 1, 1]$ would be given a parity bit 0 (as $0 + 1 + 1 + 0 = 0 \pmod 2$).

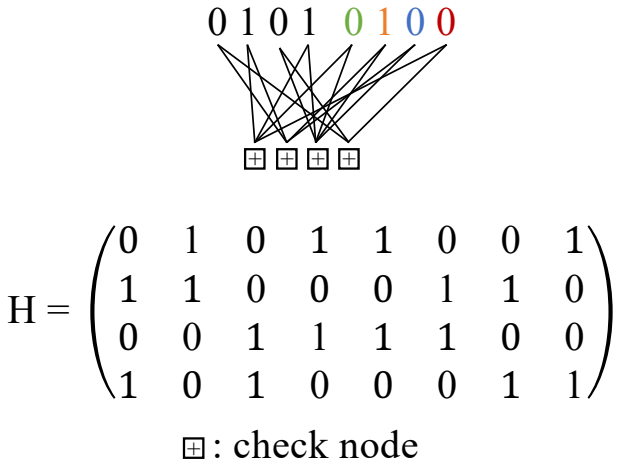


Figure 1.8: Tanner graph and associated parity check matrix

Given an input sequence of size K and a number of output bits N, $N-K$ parity bits are computed. The linkage between input bits and parity bits (i.e., which parity bit depends on which input bits) is a specific research topic as the way the association is made impacts the performances in terms of robustness and complexity. The ratio K/N is called the code rate. Figure 1.8 represents a possible association between input bits and parity bits with $K = 4$ and $N = 8$ (i.e. there is 4 parity bits, i.e. code rate is 0.5). The 4 first black bits are input bits, and the last 4 colored bits are the parity ones. Check nodes represent the link between input and parity bits: the sum of all the bits connected to a single check node must equal 0. For example, in Figure 1.8, the first check node requires that input bits 1 and 3 and parity bits 0 and 3 are even. The graph is called the Tanner graph, and the associated matrix (noted H) is called the parity check matrix. The matrix lines represent the check nodes, and the columns represent the bits (input and parity bits). $H_{i,j} = 1$ when bit j is connected to check node i. The parity bits w are generated such that $H \cdot (c, w)^T = 0$, with c being the input bits sequence.

1.3.2.3.2.2 Low density parity check concept: LDPC is a specific kind of parity check coding method. Its specificity resides in the way the parity check matrix is built. Indeed, a naive way to increase the robustness of parity-check codes is to increase the density of the edges in the Tanner graph (by increasing the number of relations between the input bits and parity bits). The main issue is that the higher the density of the graph, the highest the complexity.

The major improvement of LDPC compared to parity check coding is its capability to provide high robustness while keeping the number of edges in the graph low. It provides a very good balance between robustness and complexity. This is achieved by building the parity check matrix with two main ideas:

- The matrix is random (or pseudo-random) so that the relations between input bits and parity bits do not follow a pattern. As the number of edges is low, there are a few ones in the matrix, which is, therefore, a sparse matrix.
- In usual parity-check codes, the parity bits are not interconnected whereas, in LDPC, the parity bits are interconnected with a low number of relations.

A good introduction to LDPC codes is given in [13].

1.3.2.3.2.3 5G LDPC: As detailed in TS38.212 section 5.3.2, the method used in 5G is derived from the basic LDPC concept and is called Quasi-Cyclic LDPC (QC-LDPC). This method is slightly different from the usual LDPC. Indeed, the bits (both input and parity bits) are grouped

(the size of a group is called the lifting size and is noted Z_c). The parity check matrix is built based on $Z_c \times Z_c$ zeros sub-matrices and $Z_c \times Z_c$ right-shifted identity sub-matrices. The cyclic term refers to the shifted identity sub-matrices where the association between bits and check nodes is sequential (bit 0 is only connected check node 0, bit 1 to check node 1, etc) with an initial offset due to the shift. 5G LDPC is "quasi-cyclic" as the parity check matrix is not only composed of shifted identity sub-matrices but also of zeros sub-matrices.

Given that the receiver must be able to decode the channel, both transmitter and receiver must use the same parity check matrix defined in the standard. The generation of this matrix depends on K, N, Z_c, and the Base Graph (BG) to be used. Depending on the transport block size and code rate, two different BGs are used in 5G. For high transport block size and high code rates, BG 1 is used, otherwise BG 2 is used. H is not defined as-is but is generated using a two steps procedure:

- First, a primary matrix called H_{BG} is built based on the BG and the set index (noted i_{LS}), which depends on Z_c. H_{BG} is a $m \times n$ (46×68 for BG 1 and 42×52 for BG 2) matrix and is generated using Tables 5.3.2-2 (for BG 1) and 5.3.2-3 (for BG 2) from TS38.212. First, the matrix is initialized with -1. Then, for each row i and column j present in the tables, the associated $H_{BG}(i,j)$ is set to the value corresponding to the i_{LS} parameter.
- Then, H matrix can be derived from H_{BG}. First, elements $H_{BG}(i,j) = -1$ are replaced by $Z_c \times Z_c$ zeros sub-matrices and elements with $H_{BG}(i,j) \geq 0$ by a $Z_c \times Z_c$ identity matrix right shifted $H_{BG}(i,j)$ times.

Let us consider a toy example where $Z_c = 3$ and H_{BG} is given by:

$$H_{BG} = \begin{pmatrix} 0 & 2 & -1 \\ 1 & -1 & 2 \end{pmatrix}$$

The associated parity check matrix H is :

$$H = \begin{pmatrix} I_{Z_c} & I_{Z_c}^{(-2)} & 0_{Z_c} \\ I_{Z_c}^{(-1)} & 0_{Z_c} & I_{Z_c}^{(-2)} \end{pmatrix}$$

Where I_n and 0_n are the identity matrix and zero matrix of size n. $A^{(s)}$ is the matrix A left shifted s times (and thus $A^{(-s)}$ the matrix A right shifted s times).

Therefore, H is given by:

$$H = \begin{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

1.3.3 QoS management

The network's QoS management is a fundamental aspect of the 5G system to support the variety of terminals that might access the network. In 5G NR, the traffic is routed based on the PDU session. The PDU session is an end-to-end concept as it goes from the UE to the UPF. Within a PDU session, the user traffic can be either IP, ethernet, or even custom datagrams traffic to fit specific customer's requirements. One PDU session is associated with a single UPF and therefore defines a gateway from the mobile network towards the data network. Within a PDU session, multiple QoS profiles can be multiplexed depending on the services the UE is accessing.

1.3.3.1 CN

On the CN side, the PDU session is implemented by associating one PDU session with one GTP-U tunnel, which goes from the gNodeB to the UPF. Given that multiple QoS profiles can be multiplexed within a single PDU session, the components involved in GTP-U tunnel routing must be able to support different QoS profiles.

1.3.3.2 RAN

On the RAN side, the QoS is managed at the different levels of the stack:

- First the Service Data Adaptation Protocol (SDAP) protocol is responsible for mapping packets from one PDU session to the corresponding QoS flow. Each QoS flow is tagged with a 5G QoS Identifier (5QI) which gives the associated constraints. 5QI are defined in Table 5.7.4-1 of TS23.501 [6].
- Then, the RLC layer associates QoS flows to Data Radio Bearers (DRB). Multiple QoS flows can be mapped onto the same DRB. After multiplexing flows onto a DRB, they cannot be distinguished by the lower layers, and therefore, mapping different QoS flows onto a single DRB can only be done if the flows have close requirements. Especially, Guaranteed Bit Rate (GBR) and non-GBR traffics should not be mapped onto the same DRB.
- Finally, the MAC layer allocates the resources and configures the transport blocks based on the DRB of the RLC Service Data Unit (SDU). For example, for sensitive DRB, a low modulation with a high code rate can be chosen, but it requires more radio resources for the same data. Multiple RLC SDUs with different DRB can be multiplexed in a single transport block if the constraints associated with each DRB can be met together.

It can be seen that the RLC layer is critical for QoS management. Indeed, this layer is responsible for informing the MAC layer about how to handle the traffic on the radio link by assigning a DRB to each QoS flow. The higher the number of DRB, the higher the complexity at the MAC layer, but the finer the QoS management and reversely. Therefore, the RLC must find the best trade-off between complexity and QoS management granularity.

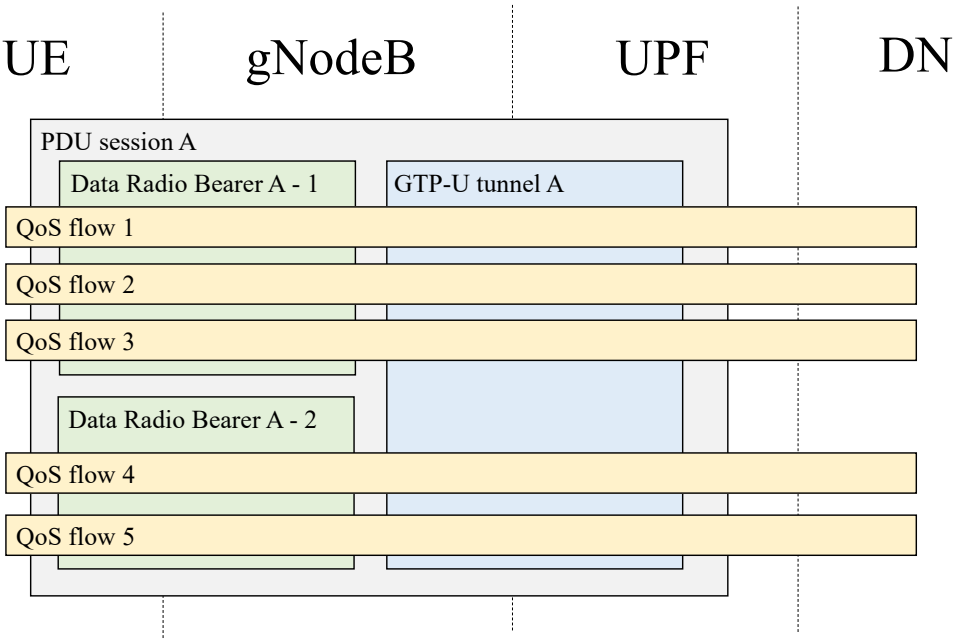


Figure 1.9: QoS management in 5G

Figure 1.9 represents how QoS flows are multiplexed and managed in 5G.

1.3.4 Deployment schemes

Given that moving from one generation to the next is a highly challenging process, and to smooth the expenses along time, the 5G network can be deployed in two modes that are Standalone (SA) and Non Standalone (NSA). In 5G SA, the 5G RAN communicates with a 5G CN whereas in NSA, it is possible to connect a 5G RAN to a 4G CN or reciprocally a 4G RAN to a 5G CN. Even if

the long-term objective is to have all the 5G networks deployed in SA, using the NSA deployment scheme with the 5G RAN and 4G CN is a first way to deploy the 5G network, while still leveraging 4G infrastructure. The interest of such deployment is that 5G base stations can be deployed, and the operator can propose increased performances to its customers and use new frequency bands without considering the migration of the CN. When the operator is ready, it can finally move from 5G NSA to SA by deploying the 5G CN.

5G NSA is called non-standalone as it cannot operate without a 4G cell: the 4G cell is responsible for network control, and the 5G cell carries user traffic. Therefore, UEs start the cell search, and registration procedure on a 4G cell, and the CN decides to move the UE from the 4G base-station to the 5G base-station for user traffic. 5G NSA is also depicted as Dual Connectivity with New Radio (DCNR).

Different options have been introduced for 5G NSA among which options 3 and 3a are the most likely to be used for 4G to 5G evolution. In both cases, the control traffic goes from the UE to the 4G eNodeB and from the eNodeB to the 4G CN. The difference is for the user traffic. As represented in Figure 1.10, in option 3, user traffic goes from the UE to the gNodeB and reaches the CN through the 4G eNodeB. On the other side, as represented in Figure 1.11, in option 3a the user traffic goes directly from the gNodeB to the CN. In both cases, the UE keeps a user traffic link with the 4G eNodeB, even if the network will prioritize the 5G cell for user traffic.

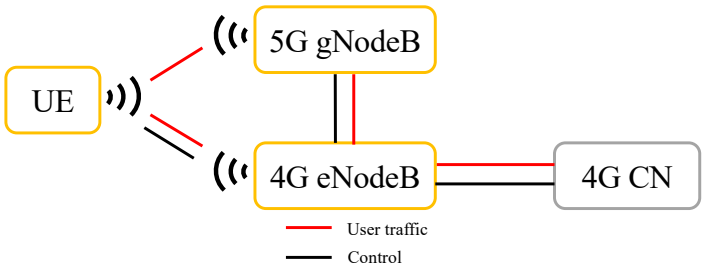


Figure 1.10: 5G NSA option 3

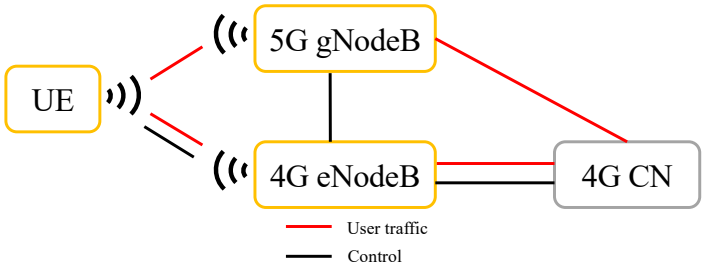


Figure 1.11: 5G NSA option 3a

Figure 1.12: 5G SA

When the operator has deployed the 5G CN, the 5G RAN can be operated in SA mode, as represented in Figure 1.12.

1.4 5G RAN functional split

1.4.1 General concept

The introduction of the functional split is a significant evolution of the RAN from 4G to 5G. It consists of dis-aggregating the RAN functions over the network. Then, the functions are grouped and deployed on different devices connected through standardized interfaces. This new paradigm emerges in the context of the network's virtualization, where all the functions are moved from hardware to software implementations. This concept is also depicted as open-RAN. While functional split refers to the dis-aggregation of the RAN layers and is defined in TR38.801 section 11.1 [14], open-RAN refers to the implementation aspects of the dis-aggregated RAN like the deployment, monitoring, and interfaces. The principal motivations for functional split and open-RAN are the flexibility, the optimization, the centralization and the ability to deploy multi-vendors RAN stacks.

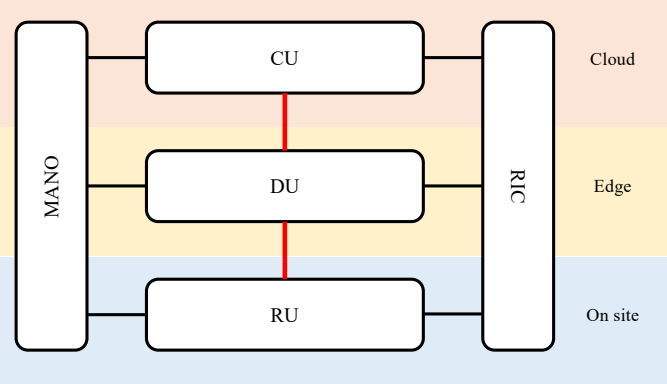


Figure 1.13: Dis-aggregated RAN architecture

Figure 1.13 represents the architecture of a dis-aggregated RAN.

The RAN functions are grouped and placed in the Centralized Unit (CU), Distributed Unit (DU) and Radio Unit (RU) which represent three levels of centralization. The CU is the most centralized and might be deployed in the cloud, whereas the DU is the intermediate between the cell's site and the CU. Finally, the RU is the radio unit that contains the lowest layers. It is directly connected to the antenna system and must therefore be onsite.

The Management and Orchestration (MANO) is a component that is responsible for managing and orchestrating the different components. Depending on a set of constraints and available resources, it decides where to place the CU and DU and how to split the functions. This is not a real-time function.

Finally, the RAN Intelligent Controler (RIC) is the component that contains the network's intelligence and protected algorithms. Indeed, the CU, DU and RU contain the different standardized functions but the RAN also contain manufacturer-specific functions and algorithms that will be deployed in the RIC. For example, the MAC layer scheduler, the handover algorithm, the radio neighborhood and UE's context and session management functions can be placed into the RIC. In a more general perspective, the RIC contains functions and APIs enabling better control of the network elements. This includes network functions like the one listed above and new ones like AI-powered network monitoring. Furthermore, in Figure 1.13, there is only one RIC but depending on the constraints and placement, there can be multiple RICs collocated with the CU and DU. Given that it might contain some highly critical functions, the RIC is a near real-time component.

The CU can be split into two different components: the CU-CP and CU-UP. The first one contains the functions for control plane processing, and the second contains the functions for user plane processing.

Therefore, open-RAN is the meeting point between the network's virtualization and the control and user plane separation. As a result, it can be leveraged to bring to the RAN the flexibility already existing at the CN's side. This flexibility is a crucial asset to support the heterogeneous services and associated tunnels implied by the various terminals connected in 5G.

1.4.2 Possible splits

A fundamental question in open-RAN is how the functions are grouped and where to split between the CU, DU and RU. This is a critical aspect as this has a huge impact on the underlying network. Indeed, the lowest the split, the largest the required bandwidth, and the highest the latency constraints. On the other side, the lowest the split, the highest the flexibility and optimization capabilities. A very good global picture of the possible splits and the associated constraints is given in [15]. Figure 1.14 gives an abstraction of this poster, where the network capacity is given for a 100MHz cell, with 256-QAM modulation, 8 layers and 32 antennas.

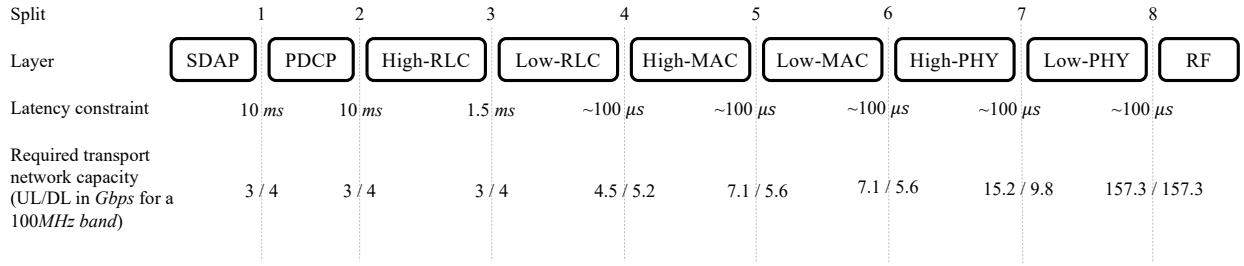


Figure 1.14: Open-RAN split options and constraints

1.4.3 Enhanced Common Public Radio Interface

The definition of an interface between the different components of an open-RAN base station is fundamental. The critical trade-off of this interface is to be flexible enough to support the different splits while being well-standardized to enable multi-vendors base stations. Ericsson, Huawei, NEC, and Nokia have addressed this issue by defining enhanced Common Public Radio Interface (eCPRI), which is defined in [1]. The eCPRI interface is multi-vendor and supports the different possible splits.

Figure 1.15 represents the eCPRI protocol stack where the left part represents the eCPRI protocol itself, and the right part represents other protocols that can be used in the context of open-RAN.

Two main deployment schemes exist for the user plane:

- eCPRI over ethernet: the user plane messages are directly encapsulated in ethernet frames.
- eCPRI over IP: the messages are encapsulated in UDP packets.

Different message types are defined for the eCPRI user plane. The three major types are:

- In-phase Quadrature (IQ) Data: This type of user plane message carries IQ samples. Depending on the split, the samples can be either in the time or frequency domain. For example, it can transfer IQ samples between the channel mapper and the inverse Fast Fourier Transform (iFFT). The header transfers information about the sequence to which belongs a message (when the sequence has been split over multiple messages) and an identifier that enables to distinguish multiple streams (like multiple users, MIMO layers etc). The user data contains the IQ samples to be transmitted.
- Bit sequence: this is used to transfer user plane data for all the splits above the modulation (i.e., split 7-3). For example, it can be used by the MAC layer to transfer transport blocks to the PHY layer. Once again, the header contains the sequence number and a stream identifier, and the user data contains the bits to be transmitted.
- Real-Time Control Data: this is used to carry internal base-station control data. It can be either metadata to be transferred with the bit sequence or IQ samples, or it can be data for sharing information from one layer to another. For example, it can be used to inform the MAC layer that a new QoS profile is being used on the network. This message is not standardized, meaning the content and format may vary from vendor to vendor. The header contains an identifier that refers to the kind of control message being transmitted and a sequence number, and the user data contains the control payload.

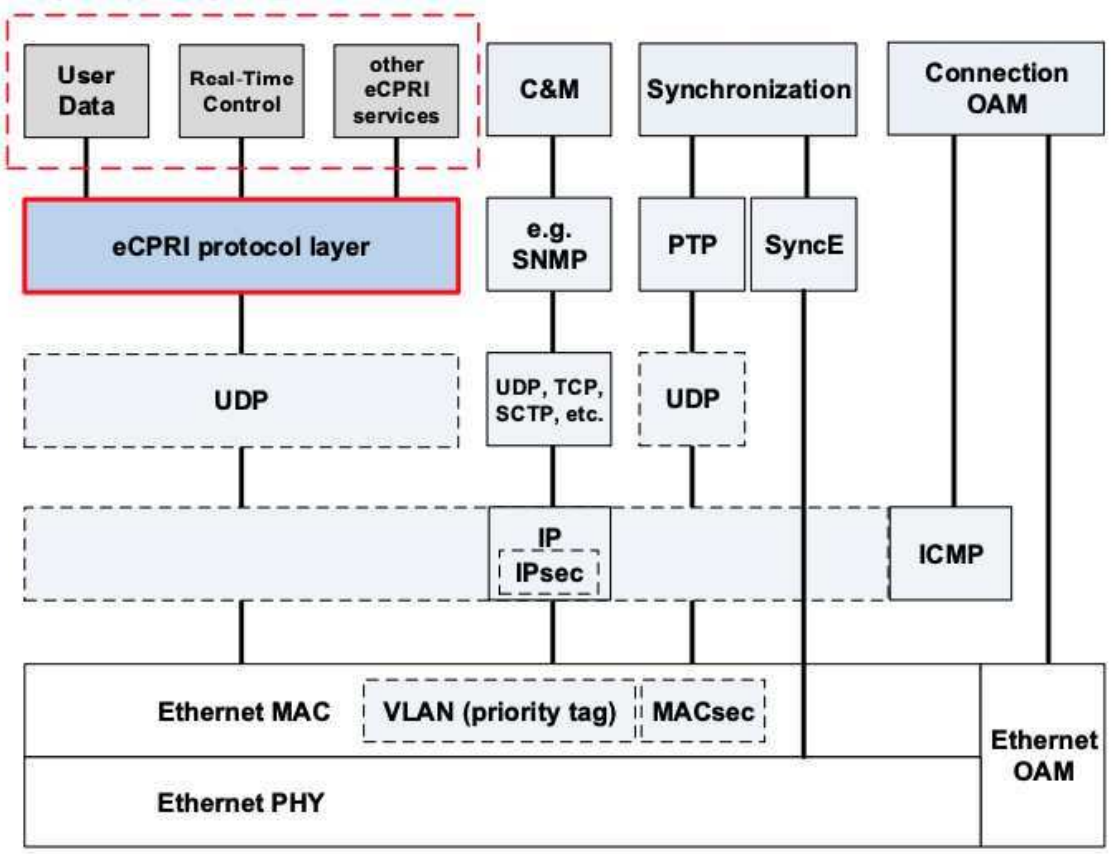


Figure 1.15: eCPRI protocol stack from [1] Figure 6

1.5 Software Defined Radio

Choosing a split among the possible ones depends on the use case and constraints. For example, in operator networks, splits 7.2x and 3 are likely to be used as it provides a good balance between centralization, transport network constraints, and resource sharing.

For the sake of this work, another use case is considered: lab experimentation and prototyping. Here are the constraints and objectives of this use case:

- The purpose is to build software prototypes of RAN stack (L1, L2, and L3).
- The number of base stations to be deployed on the network and the deployment area are much smaller than the operators' network and can even be reduced to one cell deployed in a testing room.
- This prototype should support RAN splitting.

It is decided that the two main challenges must be addressed one after the other. The first is to build a 5G base station, and the second one is to split this base station to create an open-RAN one.

Therefore, for the first developments, which consist in implementing a 5G base station, it is decided not to implement the functional split within the base-station protocol stack and split 8 is the easiest option in this case. It enables building the whole base station in one software while not considering functional split issues as a first step. Moreover, all the functions can be implemented in software to better disseminate the knowledge by open-sourcing the code.

The second challenge, which will consist in splitting the base station, can be anticipated by designing a base-station architecture that can be easily split.

SDR is one way to implement split 8. This is a new Radio Frequency (RF) paradigm where all the signal processing functions are implemented in software. The baseband signal is transmitted to and from the RF device, which carries frequency transposition, digital to analog conversion, and signal

transmission. SDR is a great option for experimentation as it can support all the radio technologies (from 2G to 5G). Moreover, given that software implementation is usually easier than hardware implementation, SDR enables quicker development and iterations even for the lowest functions of the stack.

Nevertheless, SDR is facing two main bottlenecks. The first one is that software implementations are likely to provide less good performances than hardware ones. Furthermore, it implies the highest constraints on the underlying transport network. Therefore, split 8 and SDR appears to be a great option for building prototypes and cloud-RAN deployments but might not fit all the market segments.

1.6 Conclusion

In this chapter, an introduction to the 5G system has been given. First, the overall system's architecture was detailed. Then, the 5G RAN was described, including the protocol stack, the air interface, the channel coding techniques, and QoS management. Finally, an introduction to RAN splitting was given. It is established that SDR is the technology that will be used in this work to implement the physical layer. This technology implements the open-RAN split 8, implying the highest constraints on the link between the RF device and the base station. However, it provides the highest degree of flexibility and enables a full-software implementation of the physical layer, which is great in the context of network slicing and RAN splitting.

Part I

5G PHY layer implementation

Introduction

The 5G system introduces many new technologies, from the CN to the RAN. Those technologies are critical enablers for various 5G use cases such as industry 4.0, augmented reality, and smart cities. Virtualization is the fundamental building block for such new technologies. On the CN side, virtualization methods are mature as they mostly come from the data center and cloud world. On the other side, virtualization must be adapted to new RAN constraints (like network slicing and RAN splitting).

In this context, 5G RAN research and education frameworks are required. Such frameworks should enable one to deeply understand the technology by clearly exposing the key components and clarifying the threshold between standard definitions and manufacturers' implementations. Moreover, such a research framework should also enable one to develop new RAN technologies, algorithms, and methods by having a modular architecture and providing APIs. Finally, its modular architecture should also ease RAN virtualization.

In this part, *free5GRAN*, an open-source 5G physical layer, is introduced. It is designed to be an easy-to-understand, easy-to-use, and highly modular framework. It can be used by qualified engineers and researchers for new technology testing and developments and by beginners who will go through the main components of a 5G RAN stack. For people willing to test and develop new RAN technologies, *free5GRAN* provides a modular architecture and a rich library, and the existing code-base can easily be leveraged to build new projects and experiments. Moreover, for beginners willing to have a clear view of the RAN architecture and understand the development trade-offs, *free5GRAN* exposes the key components of a RAN stack and provides an implementation of the system from scratch. Other open-source projects such as srsLTE [16] and OpenAirInterface [17] already exist but are mostly designed to deploy efficient testbeds. *free5GRAN* aims at developing a software and documentation framework to understand the 5G system, like the approach used in the Linux From Scratch project [18].

Beyond prototyping and knowledge dissemination, the modularity of this physical layer intends to ease the customization of the different functions for the implementation of network slicing. Furthermore, its software architecture is designed to envision the use of *free5GRAN* in open-RAN deployments.

This part is split in four chapters. Chapters 2 and 3 give a global understanding of the physical layer, its procedures and main functions. Chapter 4 reviews the different algorithms used by the PHY layer. Finally, chapter 5 describes the *free5GRAN* code structure and software architecture.

Physical layer main procedures

The 5G PHY layer is a set of functions and algorithms used to implement procedures which enable the gNodeB and the UE to communicate with each other.

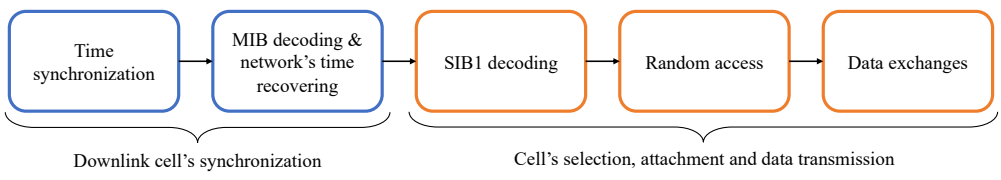


Figure 1.16: Procedures for UE's cell selection and attachment

Figure 1.16 represents the steps required for a UE to select and attach to a cell. The first step is downlink time synchronization which enables the UE to decode information from the gNodeB. The second step is to recover the network's clock and extract basic cell information, which is done through the reception of Master Information Block (MIB). Decoding System Information Block 1 (SIB1) enables the UE to extract the basic cell's RRC configuration. Once those three steps are done, the UE can determine whether or not it has to connect to the cell. If the cell is selected, the Random Access (RA) process enables the UE to establish a radio connection with the gNodeB and perform uplink synchronization. Finally, the gNodeB and the UE can exchange data.

Quick introduction to RRC protocol

The 5G physical layer processing is made according to many parameters. Those parameters are configured by default or transmitted from the gNodeB to the UE. RRC protocol has been defined for managing those parameters:

- It is first responsible for determining the best parameters to be used for a given UE at a given time, depending on constraints like channel quality or traffic requirements.
- It is also responsible for configuring the UE with those parameters.

Figure 1.17 represents the overall RRC configuration process. First, the gNodeB's RRC layer computes the best configuration for the UE. Then the configuration is communicated to the UE using the previous or default PHY configuration. When the UE's RRC layer receives the new configuration, it applies it to the UE PHY layer.

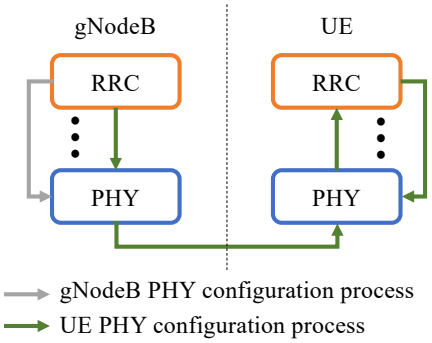


Figure 1.17: RRC configuration process

The RRC protocol is standardized in TS38.213 [19], and a good introduction is given in [20]. In this part, the core operation of the RRC protocol is not detailed, but some of the main parameters are introduced. Moreover, it is often explained where they can be found and how they are initialized to a default value.

Chapter 2

Downlink synchronization procedures

Contribution

In this chapter, we give an overview of the downlink synchronization procedures. A review of the Synchronization Signal Block (SSB) is proposed, and we explain how Primary Synchronization Signal (PSS) and Secondary Synchronization Signal (SSS) are used to determine the symbols borders. Then, we detail the broadcasting of the MIB, which contains the basic cell's information. First, we describe the MIB payload and fields, then we explain how it is encoded and placed into the SSB, and finally, we detail how UEs use it to recover the network's timing information. The contribution of this chapter is to expose some key information acquired during the implementation of the 5G physical layer in order to complete the detailed information contained in the standard. It gives a global understanding of the procedures and references to the standard, where further details can be found.

2.1 Introduction

Downlink synchronization between the gNodeB and UEs is the first mechanism that must be implemented at the UEs' side. It is required because the different components of the cell do not have the same internal clock. Therefore, downlink synchronization is required to synchronize the internal clock of UEs with that of gNodeB.

First, UEs must determine what are the symbols, slots, and frames borders in order to recover the OFDM grid which enables UEs to extract and decode data. Furthermore, in Orthogonal Frequency Division Multiple Access (OFDMA) systems, the resources are allocated in time and frequency, and synchronization also enables UEs to recover the network's timing before receiving resources allocations. Finally, as the internal clock of the gNodeB and the UE are different, the transmission frequency of the two components might be slightly different, and synchronization is used to determine and compensate the frequency offset between the gNodeB and the UE.

This chapter introduces the downlink synchronization procedure. The first section investigates how UEs can determine symbols borders to recover OFDM grid. The second section details how the broadcasting of basic cell information is made and how UEs use it to recover the network's clock.

2.2 Global objective and method

Downlink synchronization between the gNodeB and the UE is two-fold. The first step, called symbol synchronization, enables the UE to recover the symbols' borders. After symbol synchronization, the UE can recover the OFDM grid in order to extract and decode data but is not yet fully synchronized with the cell as the network's clock, given by the current Sequence Frame Number (SFN) and slot ID, and radio frames start have not been determined. The second step is radio frame synchronization and enables the UE to determine the radio frames and slots borders and recover the network's time.

Figure 2.1 represents the different signals, variables, and parameters involved in the downlink synchronization process.

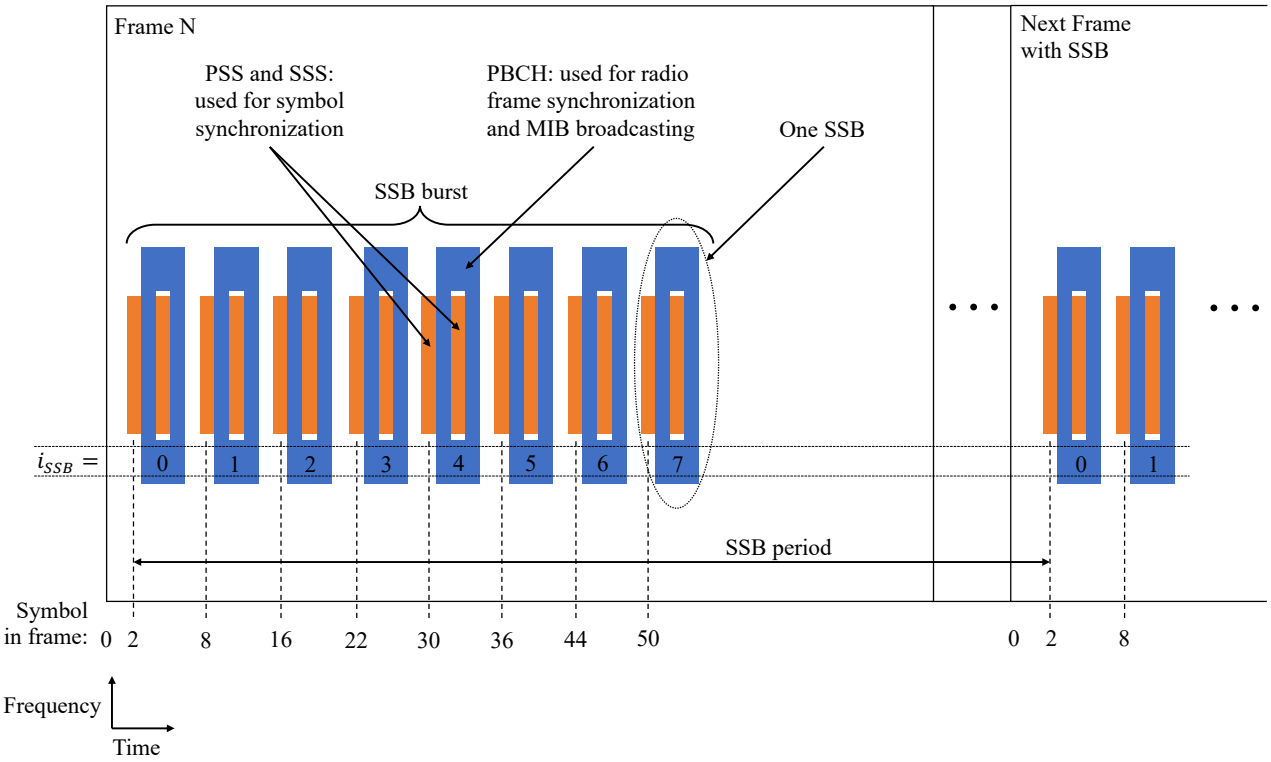


Figure 2.1: Synchronization signals representation for SSB pattern C with an operating band between 3GHz and 7.125GHz

The synchronization process relies on SSB, which is a block of 3 signals: PSS, SSS and Physical Broadcast Channel (PBCH). SSB is transmitted periodically by the gNodeB and used by UEs for initial, periodic and frequency synchronization. PSS and SSS are used for symbol synchronization. The gNodeB uses PBCH to broadcast basic information to UEs called MIB. It is also used by UEs for radio frames and slots synchronization and to recover the network's time. Figure 2.2 represents the SSB: PSS is located in the first SSB symbol, SSS is located in the third one, and PBCH is located in the second, third and fourth symbols.

SSB is a four symbols by 240 subcarriers signal located anywhere in the cell's bandwidth: its center frequency is given by the Global Synchronization Channel Number (GSCN). There are five SSB patterns (A, B, C, D, and E) that depend on the operating band and give the SSB's SCS and first symbol index.

The SSB transmission period can be either 5, 10, 20, 40, 80, 160ms, and the default value is 20ms. SSB can be transmitted several times in one period if beamforming is activated. The number of possible iterations is noted L_{max}, which corresponds to the maximum number of beams. Each SSB iteration is sent on a specific beam, and the receiver can compare the received power of the different beams. It is used by UEs to select the best beam. A SSB burst is the set of all the SSB iterations per SSB period. The index of an SSB iteration within an SSB burst is noted $i_{SSB} \in [0, L_{max} - 1]$.

Table 2.1 shows the different SSB cases. More information about the time and frequency position of the four signals can be found in TS38.211 section 7.4.3. For example, for NR band $n78$ (band between $3.3GHz$ and $3.8GHz$), the SSB pattern is case C, the SCS is 30kHz, and L_{max} is 8. In this case, 8 beams will be activated in the cell and consequently there is 8 SSB transmissions per radio frame. For example, the 7-th beam (associated with $i_{SSB} = 6$) will be sent at symbol 44 of the radio frame. This information is used by the UE to determine the start of the radio frame after SSB decoding. In this case, the UE knows that the radio frame starts 44 OFDM symbols before the start of the received SSB. The slot number within the frame and symbol number within the slot can be derived from the symbol number: the slot number is $\lfloor 44/14 \rfloor = 3$, and the symbol number within the slot is $44 \pmod{14} = 2$.

The PSS and SSS sequences are generated based on the Physical Cell ID (PCI) noted $N_{ID}^{cell} \in$

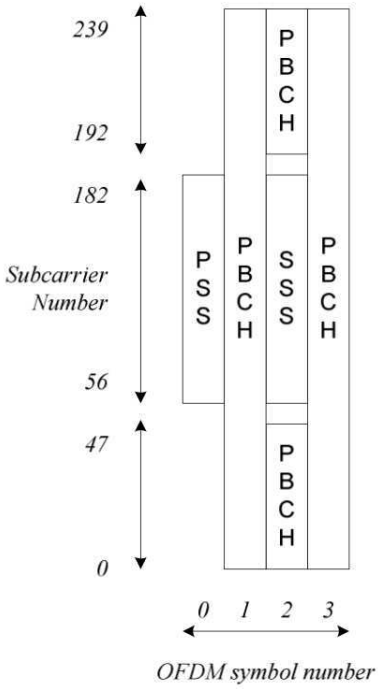


Figure 2.2: Time and frequency position of SSB, from TS38.300, Figure 5.2.4-1 [2]

$[0, 1007]$, from which two parameters $N_{ID}^{(1)} \in [0, 335]$ and $N_{ID}^{(2)} \in [0, 2]$ can be computed such as:

$$N_{ID}^{cell} = 3 \cdot N_{ID}^{(1)} + N_{ID}^{(2)}$$

The PCI is used in order to avoid the cell's confusion at the cell's edges. The physical channels are scrambled with a sequence computed from the PCI. The PCI is determined by the UE by searching for the PSS (for $N_{ID}^{(2)}$) and SSS (for $N_{ID}^{(1)}$).

2.3 Symbol synchronization

The first step in downlink synchronization is symbol synchronization, which enables the UE to determine what are the symbols' borders in the received signal. It uses PSS and SSS signals located into the SSB.

2.3.1 Primary Synchronization Signal

PSS is a 127 subcarriers signal located from subcarrier 56 to 182 with respect to subcarrier 0 of the SSB (which corresponds to $GSCN - 120$ subcarriers) in the first SSB symbol. PSS signal is a Binary Phase Shift Keying (BPSK) sequence noted $d_{pss}(n)$ ($n \in [0, 126]$) which values are 1 and -1. Three possible sequences can be generated depending on $N_{ID}^{(2)}$ ($N_{ID}^{(2)} \in [0, 2]$). Each sequence is a cyclic permutation of a single root sequence and the first sequence is the root sequence itself. Each sequence element is placed in subcarriers 56 to 182 in the first SSB symbol. PSS sequence generation is defined is appendix A.1.1 (from TS38.211 section 7.4.2.2.1 [21]).

PSS is a m-sequence, which is a specific kind of Linear Feedback Shift Register (LFSR). LFSR are deterministic pseudo-random sequences among which m-sequences are a subset with the lowest possible repetition. In 4G, Zadoff-Chu sequences were used for PSS and SSS but it is not highly robust while facing frequency offset between the gNodeB and the UE, as described in [22]. In 5G, Zadoff-Chu sequences are replaced by m-sequences which have better performances regarding frequency offset while still having strong auto-correlation properties. Having three possible PSS sequences is a trade-off between the number of cells using the same frequency without confusion and the frequency

Pattern	**SCS**	**SSB First symbol index (f is the cell frequency) in frame**		
		$f \leq 3GHz$	$3GHz < f \leq 7.125GHz$	$f > 7.125GHz$
		$L_{max} = 4$	$L_{max} = 8$	$L_{max} = 64$
Case A	$15kHz$	$\{2, 8, 16, 22\}$	$\{2, 8, 16, 22, 30, 36, 44, 50\}$	
Case B	$30kHz$	$\{4, 8, 16, 20\}$	$\{4, 8, 16, 20, 32, 36, 44, 48\}$	
Case C	$30kHz$	$\{2, 8, 16, 22\}$	$\{2, 8, 16, 22, 30, 36, 44, 50\}$	
Case D	$120kHz$			$\{4, 8, 16, 20\}$ $+28 \times n$ $n \in \{0, 1, 2, 3, 5, 6, 7, 8, 10, 11, 12, 13, 15, 16, 17, 18\}$
Case E	$240kHz$			$\{8, 12, 16, 20, 32, 36, 40, 44\}$ $+56 \times n$ $n \in \{0, 1, 2, 3, 5, 6, 7, 8\}$

Table 2.1: SSB cases and starting symbol in the frame for non-shared spectrum (from TS38.213 section 4.1 [5])

Parameter or variable	**Definition**
i_{SSB}	Index of the SSB iteration
L_{max}	Maximum number of iterations per SSB burst
N_{ID}^{cell}	Cell's identifier also called PCI
$N_{ID}^{(1)}$	Parameter derived from N_{ID}^{cell} for SSS generation
$N_{ID}^{(2)}$	Parameter derived from N_{ID}^{cell} for PSS generation
v	Two Least Significant Bit (LSB)s of the SFN in BCH payload

Table 2.2: Parameters and variables used in section 2.3

offset robustness. It could be possible to increase the number of cells using the same frequency while keeping strong frequency robustness by increasing the sequence size, but this would increase the search complexity at the UE's side.

2.3.2 Secondary Synchronization Signal

As PSS, SSS is a 127 subcarriers signal located from subcarrier 56 to 182 in the third SSB OFDM symbol. SSS is a BPSK sequence noted $d_{sss}(n)$ ($n \in [0, 126]$) and depends on $N_{ID}^{(1)}$ ($N_{ID}^{(1)} \in [0, 335]$). $d_{sss}(n)$ is defined in A.1.2 (from TS38.211 7.4.2.3.1 [21]). SSS sequences are Gold sequences. A Gold sequence is a combination of two m-sequences that have a specific characteristic: the number of sequences with low mutual correlation is high compared to the sequence's size. In 5G, there is 336 possible SSS sequences, whose length is 127.

2.3.3 Implementation

2.3.3.1 gNodeB

Given that the gNodeB is the time and frequency reference for the cell, it has to broadcast the SSB. The base-station first computes $N_{ID}^{(1)}$ and $N_{ID}^{(2)}$ based on N_{ID}^{cell}.

It then generates d_{pss} and d_{sss} and places them into the SSB. PBCH also has to be placed, but this will be detailed in section 2.4, as this is not an initial synchronization requirement. The SSB's time and frequency position within the OFDM grid depends on GSCN and i_{SSB}.

2.3.3.2 UE

On the UE's side, the cell search and synchronization procedure is detailed in TS38.213 section 4.1 [5]. The first step is to determine what are the SSB symbols borders and what is the $N_{ID}^{(2)}$ value used by the gNodeB. It is considered that the UE knows the cell's GSCN, so that the UE can center its receiving frequency with the center frequency of the SSB. If it were not the case, the synchronization procedure would also have to be done in the frequency domain. A correlation is made between the received signal and the three possible sequences to identify the right PSS sequence. Based on the identified sequence, the following information is obtained:

- The $N_{ID}^{(2)}$ parameter used by the gNodeB.
- The index of the first sample of the PSS, which corresponds to the highest correlation peak. This information is used to determine the four SSB symbols' borders (PSS, SSS and PBCH).

The second step determines the $N_{ID}^{(1)}$ value by identifying the right SSS sequence. First, the received SSS signal is located based on the synchronization index determined during PSS search. It is then possible to correlate the 336 possible SSS sequences against the received SSS. The highest correlation peak gives the gNodeB's $N_{ID}^{(1)}$ value and UE can finally compute the PCI.

On the UE's side, the symbol synchronization process is done periodically in parallel to other PHY functions. The periodicity of the time synchronization is implementation-specific. In *free5GRAN*, it is done every 20ms.

In order to get rid of receiver and transmitter clock de-synchronization and mitigate the Doppler effect, frequency tracking must be done. Different methods exist, and the one used in *free5GRAN* is joint frame boundary and frequency offset estimation (detailed in [23]). This method detects a fine frequency offset between $-SCS/2$ and $+SCS/2$. It is considered that the frequency offset is never outside those bounds. Otherwise, other methods such as frequency domain frequency tracking should be implemented. This method computes the frequency offset for each SSB symbol, noted f_{offset}^{symb} and the final frequency offset f_{offset} is the average of the four offsets. Frequency offset f_{offset}^{symb} between transmitter and receiver can be deduced from $d\Phi$ and the SCS with the equation:

$$f_{offset}^{symb} = SCS \cdot \frac{d\Phi}{2 \cdot \pi}$$

Where $d\Phi$ is the phase offset between the cyclic prefix and the corresponding part of the symbol. Further details and code implemented in *free5GRAN* for frequency offset computation are given in appendix A.1.3. Finally, received signal $s(k)$ can be corrected by applying the computed frequency offset:

$$s_{corrected}(k) = s(k) \cdot e^{-2 \cdot j \cdot \pi \cdot f_{offset} \cdot \frac{k}{S_{rate}}}$$

Where S_{rate} is the signal sampling rate, and k is the sample index in the received signal. This frequency correction can be applied continuously by the UE to the received signal.

At the end of this procedure, the UE knows at which sample in the received signal the first SSB symbol starts. It enables the UE to extract the time domain SSB signal from the received signal as being the four symbols following the PSS synchronization index.

2.4 Radio frame synchronization and Master Information Block decoding

Once the symbol synchronization has been performed, the UE recovered the SSB symbols' borders and can extract and decode the PBCH. This is the second downlink synchronization procedure which enables the UE to perform radio frames and slots synchronization. After this procedure, the UE is fully synchronized with the cell, as the symbols, slots, and radio frames borders and the network's time are known. Furthermore, MIB payload, which is carried by the PBCH, contains information that

Parameter	Description
SFN	Identifier of the current frame (i.e the network's time)
PDCCH configuration	8 bits information used to determine Coreset 0 and Search Space 0: 4 MSB is the index of one of the Table from TS 38.213 Table 13.1 to 13.4 and 4 LSB is the index in TS 38.213 Table 13.11
Common SCS	SCS used until UE uses a specific BWP, i.e. for SIB1 transmission and RA procedure.
Cell barred	Whether or not the cell is accessible
DMRS type A position	Symbol index where Demodulation Reference Signal (DMRS) is located
k SSB	Offset (in 15kHz subcarriers) between the first subcarrier of the SSB and the first subcarrier of the nearest lower RB in the cell's band.
Intra freq reselection	Indicates where to search for other cell if cell is barred

Table 2.3: MIB payload

enables the UE to find, extract and decode the SIB1, which is the first RRC configuration message notified to the UE. The MIB is broadcasted from the gNodeB within the SSB using the PBCH. PBCH is a specific channel dedicated to broadcasting as it is optimized for low-throughput but high robustness communications. PBCH is the most robust 5G data channel so that it can be received everywhere in the cell, even when radio conditions are awful.

The content of the MIB payload is detailed in Table 2.3.

The MIB payload is described in appendix A.2.1, from TS38.212 section 7.1.1 [11].

The total number of PBCH IQ samples to be placed in the SSB is 576. The subcarriers are from subcarrier 0 to subcarrier 239 in the second and fourth SSB symbols, subcarrier 0 to 47 and subcarrier 192 to 239 in the third one. Within the 576 available positions, 144 are reserved for PBCH DMRS transmission and 432 are occupied by the actual PBCH payload.

2.4.1 MIB payload processing functions

Once the MIB payload has been built, it has to be encoded in order to be transmitted. Figure 2.3 represents the processing functions performed by the gNodeB for encoding or by the UE for decoding MIB payload. The different steps are detailed in appendix A.2.2. The channel mapping block contains to the low-PHY functions like channel mapping, OFDM modulation and signal transmission.

2.4.2 Demodulation Reference Signal

DMRS is a signal sent with every physical channel. It is a reference signal that mitigates channel impairments. This impairment is corrected by channel estimation and equalization. The method used in *free5GRAN* is described in section 4.8.

Even if the DMRS generation depends on the physical channel, the signal is always of the same type. It is a Quadrature Phase Shift Keying (QPSK) signal generated based on a pseudo-random sequence. As for SSS, DMRS is following a Gold sequence. The sequence is initialized depending on parameters like the PCI or the UE's identifier.

For each physical channel in the uplink or downlink direction, the corresponding DMRS sequence must be generated and placed together with the physical channel. The DMRS signal generation and placement for PBCH is defined in TS38.211 sections 6.4.1 and 7.4.1 [21].

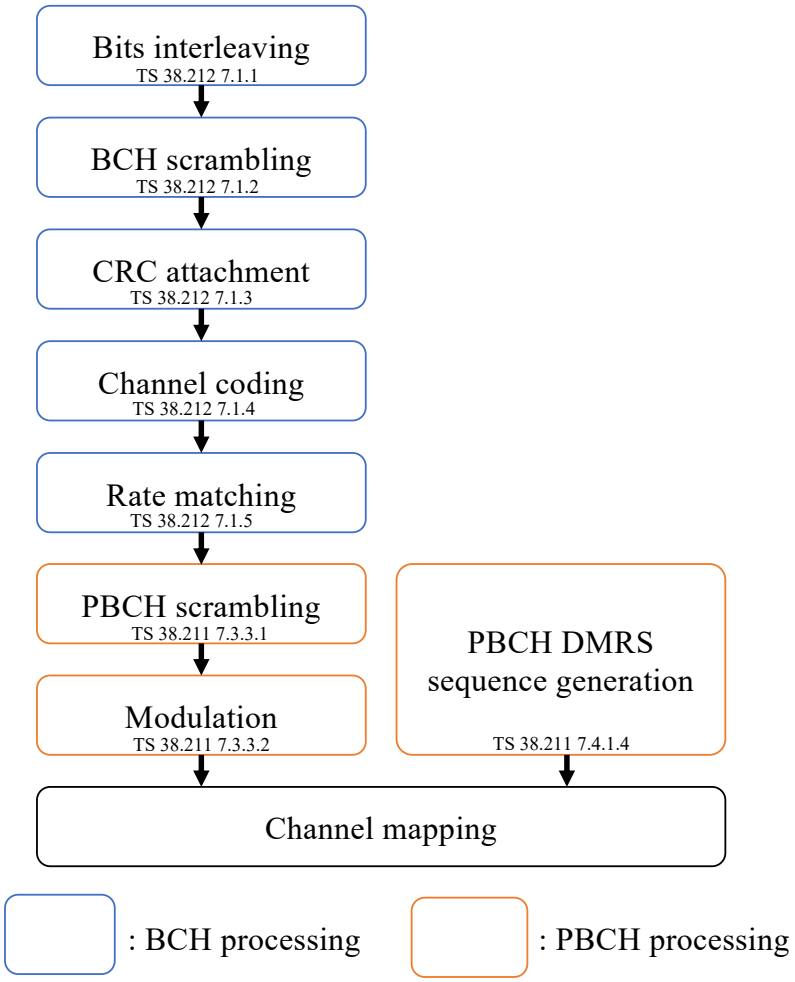


Figure 2.3: MIB payload processing functions

2.4.3 Computing channels positions *(TS38.211 section 7.4.3.1.3)*

Before sending the computed PBCH and DMRS IQ samples to the channel mapper, it is required to compute the position of each sample within the SSB.

- For the DMRS, the 144 IQ samples positions are within the PBCH resources, every four subcarriers, with an offset from subcarrier 0 of the SSB noted $v = N_{ID}^{cell} \pmod 4$. For example, if $N_{ID}^{cell} = 250$, then $v = 2$, so the first DMRS IQ sample will be placed at subcarrier 2 of the second SSB symbol, and the next ones are placed every four subcarriers. Given that PBCH is placed from subcarrier 0 to subcarrier 239 in the second and fourth SSB symbols, and from subcarrier 0 to 47 and subcarrier 192 to 239 in the third one, the final DMRS positions are $\{2, 6, 10, ..., 238\}$ for the second and fourth SSB symbols and $\{2, 6, 10, ..., 46, 194, 198, ..., 238\}$ for the third one.
- The 432 PBCH IQ samples positions are the available positions within PBCH resources, after placing the DMRS. With the same example ($N_{ID}^{cell} = 250$), the PBCH positions are $\{0, 1, 3, 4, 5, 7, ..., 239\}$ for the second and fourth SSB symbols and $\{0, 1, 3, 4, 5, 7, ..., 47, 192, 193, 195, ..., 239\}$ for the third one.

After computing the positions, PBCH and PBCH DMRS IQ samples can be transmitted.

2.4.4 Implementation

2.4.4.1 gNodeB

The gNodeB MIB encoding implementation basically follows the processing steps defined in section 2.4.1.

2.4.4.2 UE

Before decoding the physical channel, channel estimation and equalization must be performed. This is the case for all the physical channels. The channel estimation function is detailed in section 4.8.

Furthermore, given that MIB decoding is an initial cell search process, the UE is blind and does not know much about the cell. The only information decoded at this level is N_{ID}^{cell}. All the other parameters used by the gNodeB are not known by the UE. UE must perform a blind search of those parameters.

Extracting those parameters will enable the UE to perform radio frame synchronization and determine the current cell's SFN and slot ID. The parameters identified by the blind search process are the following:

- i_{SSB} (defined in section 2.2): the UE does not know the index of the current SSB iteration. This information will enable it to know when slots and frames start. To find this information, UE will loop over all the possible i_{SSB} values. As this value is used for DMRS generation, UE's channel estimation and equalization process should only be successful for the right i_{SSB} value. Once the UE has determined what is the i_{SSB} used by the gNodeB, it can determine slots and frames borders by looking into Table 2.1. For example, let assume that in band $n78$ (SSB case C) the best beam received by the UE has $i_{SSB} = 6$. The start of the current radio frame is 44 symbols before the first symbol of the current SSB. As there is 14 symbols per slot, the 44th symbol corresponds to symbol 2 of slot 3 ($3 \cdot 14 + 2 = 44$). Radio frames and slots synchronization have been done.

- The two LSBs of the SFN (Used in appendix A.2.2.2): as the UE does not know the network's clock yet, it cannot determine what are the LSBs of the SFN and thus cannot perform BCH de-scrambling. It will have to test all the 4 possible two LSBs of the SFN. For each possible values, the BCH payload is de-scrambled and the correct value is the one which equals the LSBs value after de-scrambling. Once the correct value is determined, the correct SFN is decoded and the UE has recovered the network's time. The two LSBs of the SFN give a value between 0 and 3 which is noted v.

The UE procedure to extract and decode MIB is detailed in Algorithm 1.

2.5 Conclusion

In this chapter, an overview of the downlink synchronization mechanisms was given. The SSB was introduced as the signals' block, enabling UEs to determine symbols and frames borders and recover the network's clock. From now on, it is considered that UEs are synchronized with the cell so that they can recover the cell's OFDM grid, and their clock is set to the network's clock. This synchronization procedure is made periodically, and UEs do not lose the synchronization with the cell. After decoding basic cell information, UEs are ready to extract and decode downlink data. They can perform uplink synchronization and start transmitting uplink data.

Algorithm 1 UE MIB extraction and decoding

▷ **Recover** SSB OFDM grid from SSB signal
$SSB\_grid = ofdm\_demodulation(SSB\_signal)$
▷ Compute channels positions
$[pbch\_positions, dmrs\_positions] = compute\_pbch\_and\_dmrs\_positions(N_{ID}^{cell})$
▷ Channel de-mapping
$[pbch\_samples, dmrs\_samples] = channel\_demap(SSB\_grid, pbch\_positions, dmrs\_positions)$
▷ **Blind search** i_{SSB}: loop over all the possible values
for $i_{SSB} \in [0, L_{max} - 1]$ **do**
    ▷ Compute corresponding DMRS sequence
    $dmrs\_sequence = compute\_dmrs\_sequence(i_{SSB}, N_{ID}^{cell})$
    ▷ Perform **channel estimation** and **equalization**
    $coefficients = channel\_estimation(dmrs\_positions, dmrs\_samples, dmrs\_sequence)$
    $pbch\_samples = channel\_equalization(pbch\_samples, pbch\_positions, coefficients)$
    ▷ **Decode** PBCH (demodulation and de-scrambling)
    $bch\_bits = pbch\_decoding(pbch\_samples)$
    ▷ **Decode** BCH (rate recovering, channel decoding, CRC validation)
    $[validated, scrambled\_mib\_bits] = bch\_decoding(bch\_bits)$
    ▷ If CRC **validated** (i.e. i_{SSB} has been found)
    if *validated* **then**
        ▷ Get frames and slots borders from i_{SSB}
        $[frames\_borders, slots\_borders] = extract\_frames\_and\_slots\_borders(i_{SSB})$
        ▷ Blind search the 2 LSBs of the SFN (v): loop over all the possible values
        for $v \in [0, 3]$ **do**
            ▷ **Decode** MIB and retrieve SFN
            $mib\_bits = mib\_decoding(bch\_bits)$
            $SFN = retrieve\_sfn\_from\_mib(mib\_bits)$
            ▷ If SFN LSBs correspond to v, then v has been found
            if $v\_matches\_sfn(v, SFN)$ **then**
                $return(mib\_bits)$
            end if
        end for
    end if
end for

Chapter 3

Downlink and uplink data transmissions

Contribution

In this chapter, we explain how downlink transmissions are made. Furthermore, we explain the uplink synchronization procedure and uplink data transmission mechanisms. Once again, the main contribution is to expose the critical steps for implementing a physical layer and propose an abstract of the standard. We also explain the way the different procedures and components interact with each other.

3.1 Introduction

Once UEs are synchronized with the cell, data can be transmitted between the gNodeB and UEs. However, the time and frequency domain resources are shared between the different UEs. Therefore, a mechanism has to be implemented between the gNodeB and UEs so that the gNodeB can notify UEs of the resources allocated for data transmissions. Furthermore, given that the different UEs are not in the same location within the cell, uplink synchronization must be done so that the uplink transmissions arrive simultaneously at the base station. The uplink synchronization process is integrated into the RA procedure, performed by UEs to access the cell.

The first section of this chapter introduces the mechanism for resource allocation and data extraction and decoding. The second section details how initial cell's RRC configuration is broadcasted by the gNodeB to UEs. Then, the third section explains how the RA procedure is made by UEs to access the cell and perform uplink synchronization. Finally, the last section gives an overview of the procedure for downlink and uplink data transmissions after RRC connection setup.

3.2 Data communication

This section details the overall operation of the 5G PHY downlink and uplink data communications. Understanding those two procedures is a prerequisite to understand how to transmit and decode SIB1 for radio connection setup, and it is also used for all the future downlink and uplink data exchanges between the gNodeB and the UE.

3.2.1 Overall concept

The data transmission in both directions involves two physical channels that are Physical Downlink Control Channel (PDCCH) and Physical Downlink Shared Channel (PDSCH) for the downlink and PDCCH and Physical Uplink Shared Channel (PUSCH) for the uplink. The PDSCH and PUSCH are the channels that carry the data. The gNodeB uses the PDCCH to notify resource allocation for a UE. The UE decodes the PDCCH to determine where PDSCH/PUSCH is located. Downlink data

Term or notation	Definition
CORESET	RE grid where PDCCH can be placed
Search space	PDCCH and CORESET configuration
$N_{RB}^{CORESET}$	Number of RBs in the Control Resource Set (CORESET)
$N_{symb}^{CORESET}$	Number of OFDM symbols in the CORESET
CCE	Logical CORESET resource
REG	Physical CORESET resource
REG bundle	Physical CORESET resource
Aggregation level	Number of CCEs used for PDCCH transmission
PDSCH mapping type	Way PDSCH IQ samples are mapped to physical resources
$dmrs-AdditionalPosition$	Number of additional symbols with DMRS
l_0	First symbol with DMRS in resources allocation
l_d	Duration of the resources allocation

Table 3.1: Terms and notations used in section 3.2

reception and uplink data transmission (in a UE perspective) are consequently based on those three channels:

- First, the UE continuously searches for PDCCH elements in specific areas of the OFDM grid. Given that there are multiple UEs in the cell (and thus multiple potential PDCCH elements to be transmitted simultaneously), the PDCCH location is defined by a search zone within the OFDM grid. Inside this zone, multiple PDCCH elements can be sent. The different positions of the possible PDCCH elements within the search zone are called PDCCH candidates. The shape (time and frequency positions) of this search zone within the grid is given by an object called the CORESET. The search space gives the periodicity and timing information of the CORESET. To extract and decode PDCCH, the UE has to determine which CORESET and search space must be used depending on the current procedure step. Then, it uses those information to locate the different possible PDCCH candidates in the grid and tries to decode them. If any data is transmitted from the gNodeB to the UE, decoding one of the candidates must be successful. After extracting PDCCH, DCI can be extracted. DCI is the transport block payload that carries information about the time and frequency position of the PDSCH or PUSCH. Furthermore, DCI informs the UE about whether or not the current resources grant is for downlink or uplink communication.

- If the DCI carries downlink grant, the UE uses it to find and extract PDSCH IQ samples in the OFDM grid. It can then perform PDSCH and DL-SCH (transport channel carried by PDSCH) decoding and extract the data transmitted by the gNodeB.

- Otherwise, if the DCI carries uplink resources grant, the UE can notify MAC layer that uplink resources have been granted. The transport block is UL-SCH (transport channel carried by PUSCH) and PUSCH encoded. Finally, the PUSCH IQ samples are placed in the OFDM grid according to the resource allocation (given by the DCI), and the generated signal is transmitted to the gNodeB. The gNodeB also uses the DCI to wait for uplink transmission and can recover the transmitted uplink transport block by performing PUSCH and UL-SCH decoding.

Figure 3.1 represents the different channels and notations involved in the data transmission process.

3.2.2 DCI and PDCCH encoding

The first step to communicate between a gNodeB and a UE is to notify where the resources for downlink or uplink data are located in the OFDM grid. That information is contained in a transport payload called DCI, carried by the PDCCH. Depending on the configuration and current procedure, there are different DCI payload formats. It will be discussed later as it depends on the procedure step

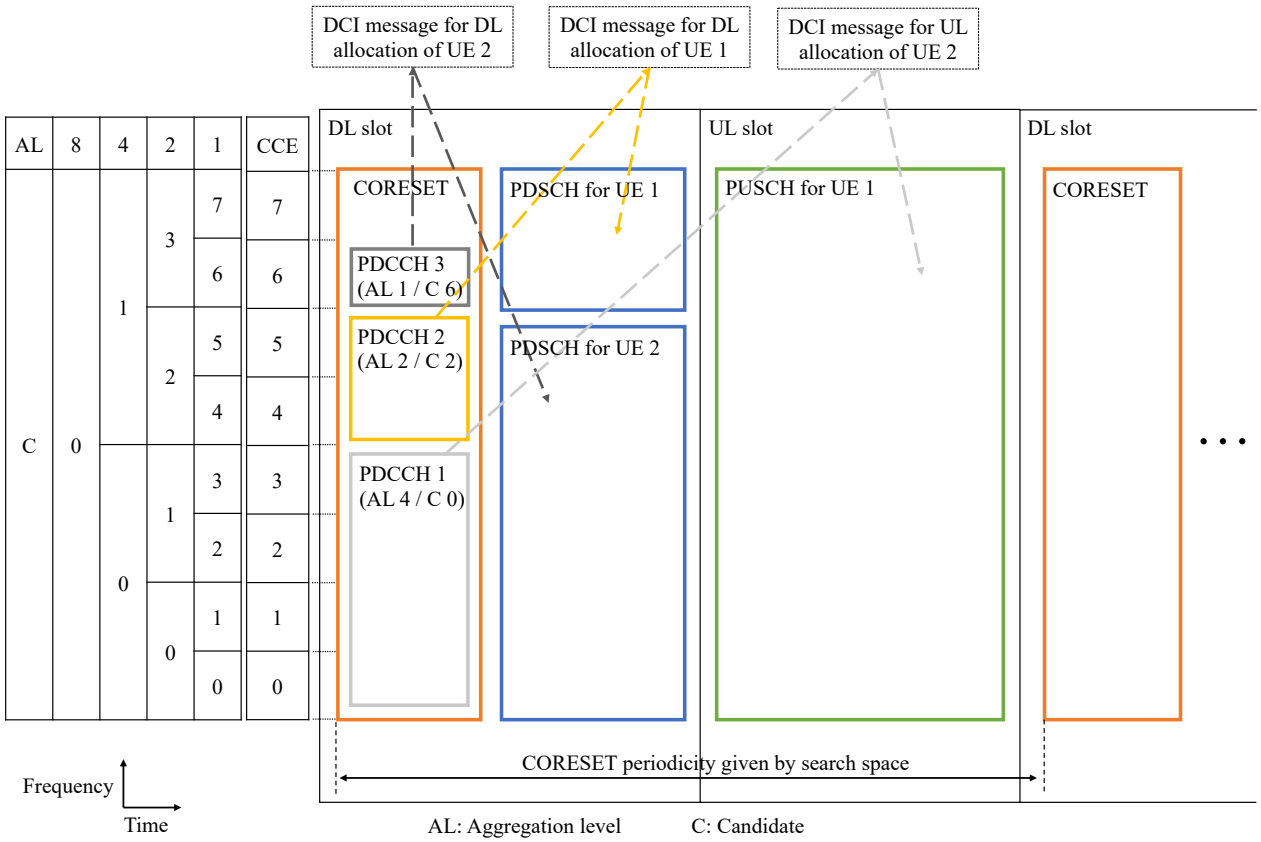


Figure 3.1: Data transmission representation for a CORESET with 8 CCEs and no CCE-to-REG mapping

and the direction (downlink or uplink). Figure 3.2 represents the DCI and PDCCH encoding steps for a generic payload. Given that the Cyclic Redundancy Check (CRC) attachment process is specific for PDCCH, it is detailed below and the other processing steps are detailed in appendix A.3.1.

3.2.2.1 CRC masking and attachment *(Transport channel processing - TS38.212 section 7.3.2)*

CRC computation and validation follows the basic CRC process and is made using g_{CRC24C} polynom, as detailed in section 4.2. The specificity to PDCCH relies on Radio Network Temporary Identifier (RNTI) masking. Indeed, after CRC computation, the CRC is scrambled with the RNTI. The RNTI is the UE's identifier within the cell. RNTI masking enables to identify to which UE the allocation is intended. Indeed, as the PDCCH might be transmitted in shared resources, UEs need to be able to distinguish what are the PDCCH/DCI elements dedicated to them. With RNTI masking, CRC validation is successful only for the UE to which the element is intended. The 24 CRC bits are added to the DCI payload.

3.2.2.2 PDCCH candidates position in CORESET

Before understanding how to place PDCCH IQ samples into the OFDM grid, it is critical to understand how the possible PDCCH elements can be placed in the CORESET.

3.2.2.2.1 CORESET object: The CORESET object contains the following information:

- CORESET ID.
- Frequency domain resources: RBs allocated for the CORESET.
- Duration (in OFDM symbols).

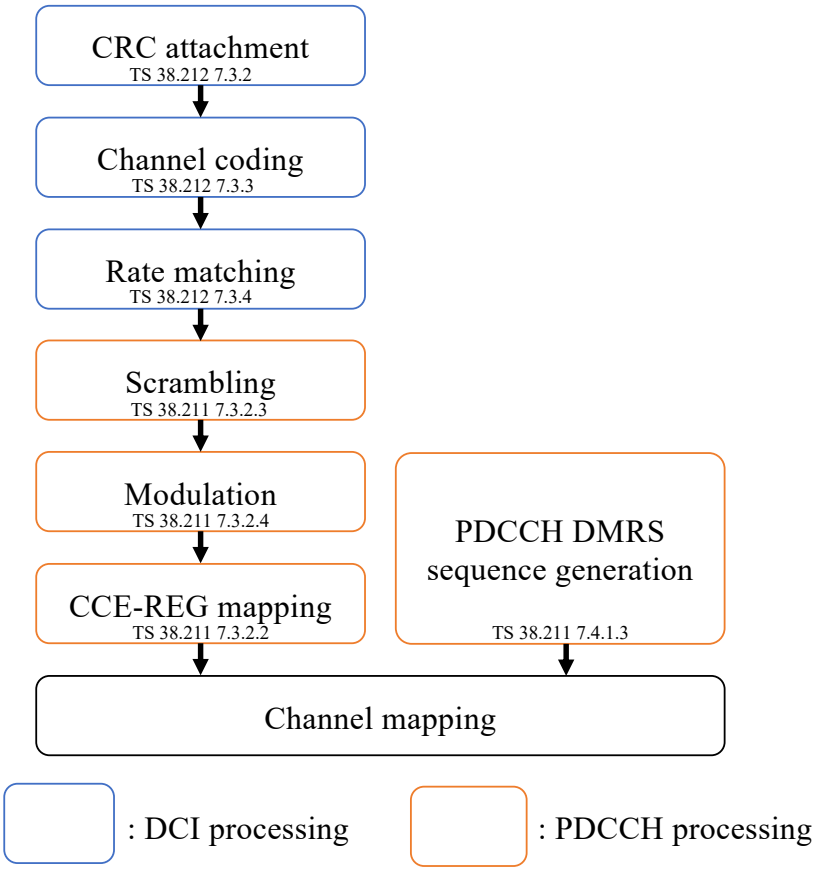


Figure 3.2: DCI payload processing steps

- CCE-to-REG mapping: Whether or not PDCCH IQ samples are placed contiguously in the grid.

Frequency domain resources information gives both the number of RBs in the CORESET ($N_{RB}^{CORESET}$) and the position of those RBs within the active BWP. The duration parameter, expressed in symbols, is noted $N_{symb}^{CORESET}$. The CORESET can be considered as a REs sub-grid of $N_{RB}^{CORESET}$ RBs by $N_{symb}^{CORESET}$ symbols.

3.2.2.2.2 Splitting the CORESET in groups of resources: The CORESET is first split into Resource Element Group (REG). Unlike the RB (which is a frequency only information), the REG is a time and frequency information: 1 REG is 1 RB over 1 OFDM symbol. For example, if $N_{RB}^{CORESET} = 48$ and $N_{symb}^{CORESET} = 1$ or $N_{RB}^{CORESET} = 24$ and $N_{symb}^{CORESET} = 2$, then the number of REGs in the CORESET is 48. Within the CORESET, REGs are numbered in time direction first and frequency direction then. Figure 3.3 represents the different REGs for two different CORESET configurations (48 RBs by 1 symbol CORESET on the left and 24 RBs by 2 symbols CORESET on the right). The REG represents the unit of resource which can be used to place PDCCH elements.

Then, the CORESET is split into physical and logical resources:

- First, it is split in groups of physical resources that are called REG bundles. A REG bundle is a group of 6 REGs. In some specific configurations, the number of REGs in a REG bundle can vary.

- Furthermore, the CORESET is split in Control Channel Element (CCE). One CCE is also a set of 6 REGs. However, CCEs are only representing an amount of logical resources whereas REG bundles represent a physical resource and position in the CORESET.

For example, if $N_{RB}^{CORESET} = 48$ and $N_{symb}^{CORESET} = 1$ or $N_{RB}^{CORESET} = 24$ and $N_{symb}^{CORESET} = 2$, then the CORESET contains 8 CCEs and REG bundles.

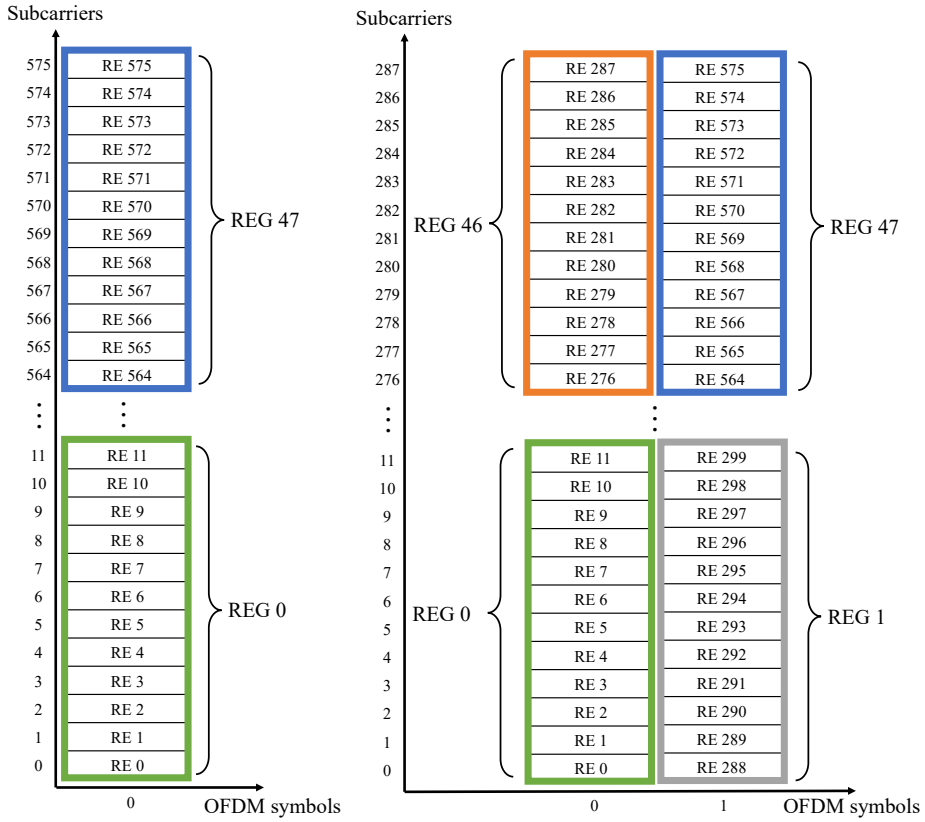


Figure 3.3: Mapping between REs of the CORESET and REGs for a 48 RBs by 1 symbol CORESET on the left and 24 RBs by 2 symbols CORESET on the right

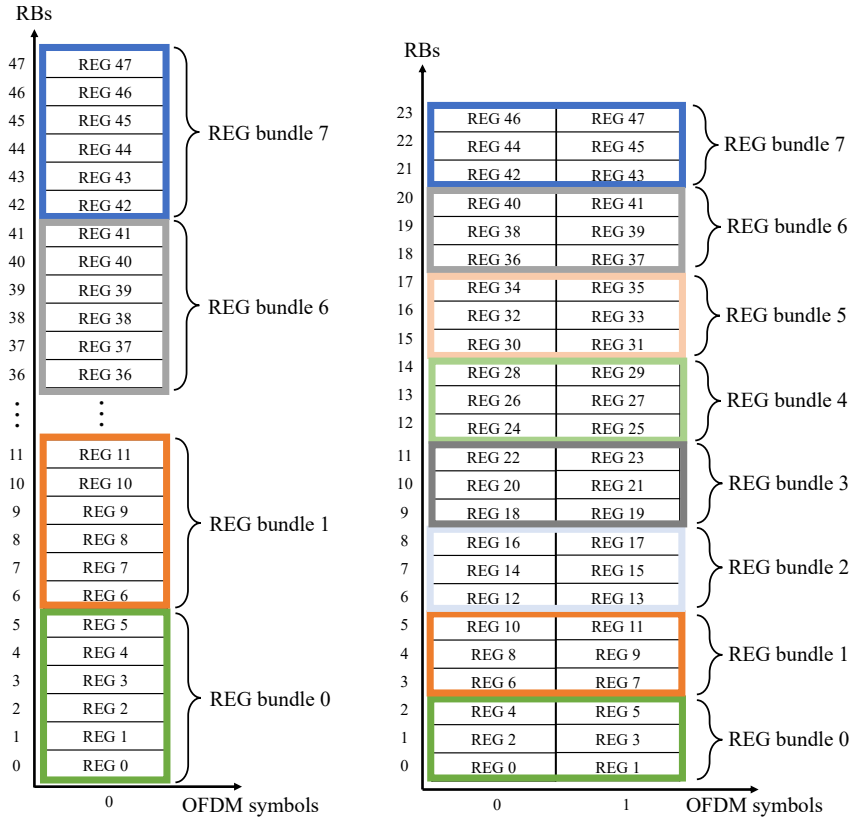


Figure 3.4: Mapping between REGs and REG bundles for a 48 RBs by 1 symbol CORESET on the left and 24 RBs by 2 symbols CORESET on the right

Agreggation level	Candidate	CCEs
8	0	$\{0, 1, 2, 3, 4, 5, 6, 7\}$
4	0	$\{0, 1, 2, 3\}$
	1	$\{4, 5, 6, 7\}$
2	0	$\{0, 1\}$
	1	$\{2, 3\}$
	2	$\{4, 5\}$
	3	$\{6, 7\}$
1	0	$\{0\}$
	1	$\{1\}$
	2	$\{2\}$
	3	$\{3\}$
	4	$\{4\}$
	5	$\{5\}$
	6	$\{6\}$
	7	$\{7\}$

Table 3.2: Candidates CORESET with 8 CCEs

First, the PDCCH elements are associated to logical resources (i.e., CCEs) which are then mapped to physical resources (i.e., REG bundles). The mapping, which is defined in TS38.211 section 7.3.2.2 [21], is made so that CCE x is mapped onto REG bundle $f(x)$. When CCE-to-REG mapping field from CORESET object is set to false, the mapping is deactivated and $f(x) = x$.

Using logical and physical resources enables the base station to increase the frequency diversity of the CORESET. Indeed, the number of elements in each CORESET occurrence depends on the type of PDCCH elements carried by the CORESET. When the CORESET is likely not to be full, CCE-to-REG mapping is activated so that the resources used in the CORESET are not grouped but distributed over the whole CORESET to increase the frequency diversity. On the other side, when the CORESET is likely to be full, it is not necessary to activate the mapping as frequency diversity is already made by occupying all the available resources.

The number of CCEs that are used by a PDCCH element is given by the aggregation level. It can be 1, 2, 4, 8 or 16 but it is limited to the number of CCEs in the CORESET. Using the same example, with a CORESET containing 8 CCEs, the possible aggregation levels are 1, 2, 4 and 8.

As a result, several sets of CCEs can be selected for transmission of a PDCCH element with a given aggregation level. The possible sets of CCEs are called candidates. Table 3.2 gives all the aggregation levels and candidates for a case where there is 8 CCEs in the CORESET. For example, if the aggregation level is 4 and the selected candidate is the second one (with index 1), then the CCEs used for PDCCH transmission are $\{4, 5, 6, 7\}$.

3.2.2.2.3 Placing PDCCH elements in the CORESET: Therefore, the different PDCCH elements are placed in the CORESET by selecting an aggregation level and candidate for each. That information gives the CCEs that can be used for transmission. Finally, the CCEs are mapped to REG bundles depending on the CCE-to-REG mapping field. Once the REG bundles to be used for transmission are determined, the IQ samples of the PDCCH element can be placed in the REGs belonging to the identified REG bundles.

For example, in Figure 3.1, there is three PDCCH elements to be placed in the CORESET. The MAC layer decides to transmit element 1 with aggregation level 4, element 2 with aggregation level 2 and element 3 with aggregation level 1. This decision is based on the transmission importance and UE's channel quality. Indeed, the higher the aggregation level, the higher the number of IQ samples that can be transmitted and, therefore, the higher the number of code bits and channel robustness.

When the aggregation level is chosen, the candidate must be selected. In Figure 3.1, candidate 0 is chosen for the element 1, candidate 2 for element 2 and candidate 6 for element 3. The associated CCEs are therefore $\{0, 1, 2, 3\}$, $\{4, 5\}$ and $\{6\}$ respectively. Other candidates could have been selected such as candidate 1 for the first element, candidate 0 for the second one and candidate 2 for the third one. The associated CCEs would have been $\{4, 5, 6, 7\}$, $\{0, 1\}$ and $\{2\}$ respectively. Given that the CCE-to-REG mapping is deactivated in Figure 3.1, the REG bundles to be used for each PDCCH element correspond to the selected CCEs.

3.2.2.3 Computing positions *(Physical channel processing - TS38.211 section 7.3.2.2)*

The placement of the PDCCH IQ samples into the OFDM grid is made in three steps. First, the search space to be used has to be selected, then the IQ samples are placed into the CORESET, and finally, the CORESET is placed itself into the OFDM grid using search space information.

3.2.2.3.1 Search Space: Search space is an object which defines the PDCCH configuration used by the gNodeB. It is transmitted to the UE using RRC messages. Search space can be UE specific or common to all the UEs.

As defined in TS38.331, search space contains that information:

- Search space ID.
- Monitoring slot periodicity and offset: PDCCH periodicity in slots and offset from the beginning of a frame.
- Monitoring symbols within slot: symbols in the slot where the CORESET is located.
- Number of candidates: table representing the number of PDCCH candidates per aggregation level.
- Search space type: whether or not the search space is common or UE specific and which format of DCI payload can be placed (and found by the UE) in it.
- CORESET ID: Pointer to the CORESET that must be used.

Search space selection depends on DCI format that must transmitted. The data transmission of PDCCH and PDSCH / PUSCH must be scheduled by the MAC layer according to the selected search space configuration. The search space to be used for the transmission is selected and notified by the MAC layer.

3.2.2.3.2 Placing the IQ samples into the CORESET: As defined in section 3.2.2.2, the PDCCH IQ samples are placed in the CORESET by selecting an aggregation level and candidate. Once the CCEs and corresponding REG bundles are identified, the IQ samples can be placed into the CORESET. Within one REG (1 RB over 1 OFDM symbol), REs $\{1, 5, 9\}$ are reserved for DMRS transmission and the remaining ones ($\{0, 2, 3, 4, 6, 7, 8, 10, 11\}$) carry PDCCH payload. Given that there is 9 PDCCH payload IQ samples per REG, the size of the PDCCH payload must be adjusted at the rate matching level to be equal to $9 \cdot 6 \cdot AL$ (where AL stands for aggregation level).

3.2.2.3.3 Placing the CORESET in the OFDM grid: The PDCCH IQ samples' positions within the OFDM grid can be computed when their positions within the CORESET are known. The frequency-domain resources parameter of the CORESET defines its position within the grid. The search space gives the time-domain position. It specifies the slots in the frame and symbols in slots where PDCCH can be located. One usual configuration is to position the CORESET at the beginning of each downlink slot (i.e. at symbols 0 and 1 of each slot for a 2 symbols CORESET). For example, if the CORESET starts at the RB 2 within the BWP and is placed at the beginning of the slot, then the positions within the slot OFDM grid are the same as the positions within the CORESET but

with a subcarrier index shifted by 2 RBs (i.e. 24 REs). The shift offset in the time-domain is equal to 0.

Once the PDCCH IQ samples are placed in the grid, the PDCCH DMRS IQ samples are placed in positions $\{1, 5, 9\}$ of every RB where there are PDCCH IQ samples. DMRS generation is detailed in appendix A.3.1.5.

3.2.2.3.4 gNodeB: On the gNodeB's side, the MAC layer gives the aggregation level, candidate and DCI payload. Encoding and IQ samples placement is made according to the above sections.

3.2.2.3.5 UE: UEs continuously loop over all the monitoring occasions (defined as being one CORESET occurrence) to search PDCCH elements. Both aggregation level and selected candidate are unknown, and the UEs have to loop over all the possible aggregation levels and candidates to search for PDCCH elements. This process is called PDCCH blind search. In some cases, RRC signaling can reduce the number of possible aggregation levels and candidates to make the blind search easier.

For each aggregation level and candidate, the associated CCEs and corresponding REG bundles are identified. The UE extracts the IQ samples from the OFDM grid based on the determined REG bundles and performs PDCCH and DCI decoding. If CRC validation is successful, a DCI payload has been found.

3.2.3 DL-SCH and PDSCH encoding

Once DCI is encoded and ready to be placed in the OFDM grid, DL-SCH must be encoded. Figure 3.5 represents the different DL-SCH/PDSCH processing steps.

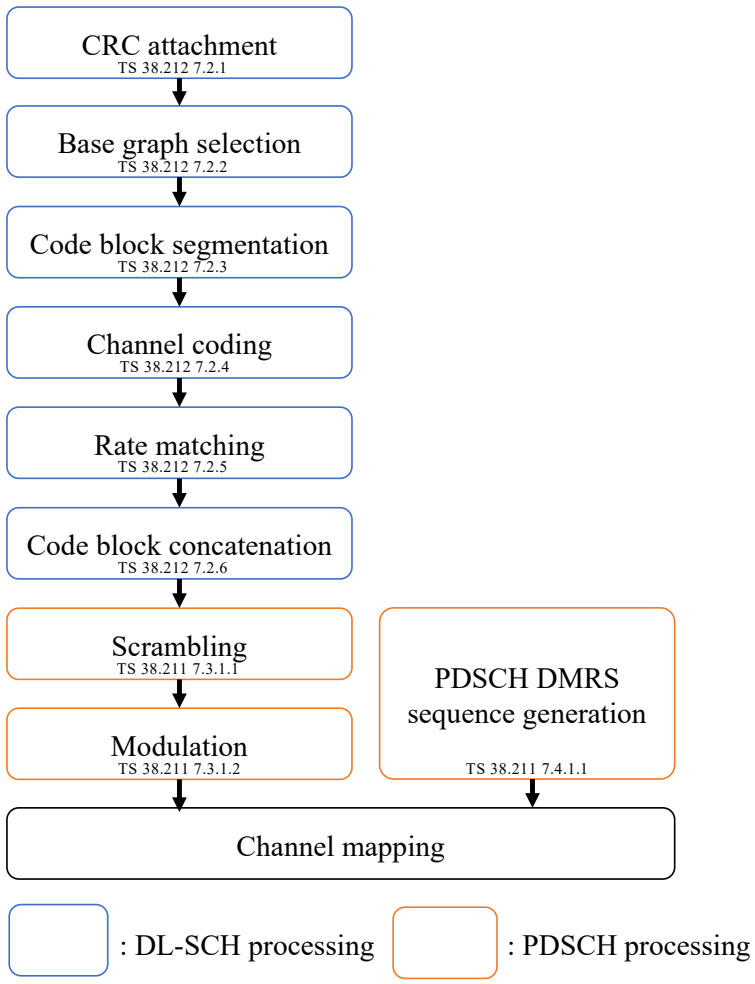


Figure 3.5: DL-SCH processing steps

Those functions are detailed in appendix A.4.1. All the PDSCH functions dedicated to MIMO and beamforming are omitted here as they were not investigated yet, and those technologies are not supported in the current version of *free5GRAN*. Those functions are layer mapping (for MIMO), antenna port mapping (for beamforming), and VRB-to-PRB mapping (for beamforming).

3.2.3.1 Computing PDSCH and PDSCH DMRS positions *(TS38.211 section 7.3.1.5 and 7.4.1.1.2)*

Computing channel positions depends on the RRC configuration. The DCI payload gives the set of REs where PDSCH can be placed. PDSCH DMRS IQ samples are first placed within those REs. PDSCH IQ samples are then placed, either by occupying all the remaining resources or by only occupying all the REs of the symbols where there is no DMRS.

The first step is to compute PDSCH DMRS positions, which is detailed in TS38.211 section 7.4.1.1.2. Unlike PBCH and PDCCH, PDSCH DMRS is not placed at every symbol of the resource allocation. The RRC configuration gives the symbols used for DMRS. There is at least one symbol with PDSCH DMRS. The DMRS placement process relies on four parameters, given by the RRC configuration:

- The first one is the PDSCH mapping type. It can be either type A or type B (defined in TS38.214 section 5.1.2.1 [24]) and defines how the PDSCH and DMRS samples are placed in the OFDM grid. In a default configuration, mapping type A is used.
- The second one is the duration of the allocation and noted l_d. For mapping type A, it is defined as being the number of symbols between the first symbol of the slot and the last symbol allocated for PDSCH.
- The third one is the index of the first symbol where DMRS is transmitted. It is noted l_0.
- The last one is $dmrs{-}AdditionalPosition$. By default, there is always one symbol with DMRS per PDSCH transmission, and this parameter gives the number of additional symbols where DMRS must be transmitted.

For example, for PDSCH mapping type A and if the configuration specifies that PDCCH occupies the first symbol of each slot (i.e. symbol 0) and PDSCH all the other symbols (i.e. symbols $[1, ..., 13]$), then $l_d = 14$. If, moreover, it is considered that $dmrs{-}AdditionalPosition$ is set to 2, then, according to Table 7.4.1.1.2-3 of TS38.211, the symbols where DMRS samples are placed are $\{l_0, 7, 11\}$. In this case, l_0 equals 2. DMRS IQ samples are positioned in every even subcarrier of the allocated RBs on symbols $\{2, 7, 11\}$. The number of DMRS IQ samples to be placed is $3 \cdot 6 = 18$ per allocated RB (there are 3 symbols with DMRS IQ samples placed every even subcarrier in each RB).

The first step to place the PDSCH IQ samples is to determine whether or not MIMO is activated. Indeed, when MIMO is activated, the resources allocated for DMRS of one layer might not be used on other layers to ensure DMRS transmission is the best possible. This configuration is given by the number of CDM (Code Division Multiplexing) groups without data which gives the number of DMRS resource groups reserved for DMRS transmission across all the layers.

Its default value is 2, which means that 2 DMRS resource groups are reserved on all the layers. The first one is the one described above (i.e., all the even subcarriers of all the RBs on symbols with DMRS), and the second one is its complement (i.e., all the odd subcarriers of all the RBs on symbols with DMRS). Therefore, when the number of CDM groups without data is set to 2, all the REs of symbols where there is DMRS are reserved, which means that PDSCH cannot be transmitted on those symbols.

On the other side, when the number of CDM groups without data is 1, there is only one DMRS resources group to be reserved (the first one, i.e., the even subcarriers of each RBs on symbols with DMRS). It means that PDSCH samples can be positioned in every odd subcarrier of each RB on symbols with DMRS. Furthermore, PDSCH can be placed in all the REs of all the symbols without DMRS.

The number of CDM groups without data is given in Tables 7.3.1.2.2-1 to 7.3.1.2.2-4A and depends on the antenna port field of the DCI payload. More information about data and DMRS positioning can be found in TS38.214 section 5.1.6.2.

3.2.4 UL-SCH and PUSCH encoding

In this chapter, it is considered that UL-SCH and PUSCH encoding follow the DL-SCH and PDSCH encoding steps (defined in the above section 3.2.3). The process is defined in TS38.212 section 6.2 and TS38.211 section 6.3.1. This is not true for all the configurations, but it is the case for simple PHY configurations. The main differences between DL-SCH / PDSCH and UL-SCH / PUSCH are:

- For UL-SCH, data and control can be multiplexed after rate matching, which avoids the transmission of a supplementary Physical Uplink Control Channel (PUCCH). It is implemented in *free5GRAN* but not detailed in this chapter. The core concept is to select a set of places in the PUSCH channel used for Uplink Control Information (UCI) placement. UCI can be either punctured onto UL-SCH if the number of UCI bits is small or multiplexed for a higher number of UCI bits.
- In uplink, DFT-OFDM can be used instead of the usual downlink OFDM (described in section 2.9.2 of [25]). This process involves the transform precoding (DFT) of the PUSCH channel.
- Precoding can be applied to the PUSCH for beamforming purposes.
- DMRS location might be different but the process to determine the positions in the OFDM grid is the same. It is detailed in TS38.211 section 6.4.1.1.

3.2.5 Implementation

3.2.5.1 gNodeB

On the gNodeB's side, the first step when uplink or downlink data must be transmitted between the UE and the gNodeB is to build the DCI payload according to the corresponding DCI format. The gNodeB uses the information from the MAC layer like the resources grant and transmission configuration (like modulation, code rate or mapping type) to build the DCI payload. The PDCCH implementation follows the procedure detailed in section 3.2.2. The gNodeB's L2 defines the search space and the CORESET for each UE, the aggregation level and the candidate for a given PDCCH and PDSCH / PUSCH transmission. That information is required before encoding the DCI as the output size depends on the available resources. For example, if the aggregation level is 4, there is $4 \cdot 6 \cdot 9 = 216$ (where 9 is the number of REs available for PDCCH payload in each RB and the 3 others are used for DMRS) REs available for PDCCH samples, which means that the DCI encoding output size is 432 (as PDCCH is QPSK modulated). The inputs of DCI encoding are: the RNTI for CRC scrambling (UE's identifier), search space and CORESET, the aggregation level, the corresponding candidate and the DCI payload.

The PDSCH encoding and PUSCH decoding implementation follow the procedure detailed in the previous section. Before starting encoding or decoding, the gNodeB needs to determine some information:

- The PDSCH/PUSCH and DMRS configuration used to determine the allocation's size and the channel positions.
- The allocation's size (number of REs allocated for PDSCH and DMRS). The allocation's size depends on the RBs and symbols allocated (which can be found in DCI payload) and on the PDSCH and DMRS configuration (section 3.2.3.1 explains how those configurations impact the allocation's size).
- The Modulation and Coding Scheme (MCS) to be used (located in the DCI payload), which enables to determine the code rate and the modulation order.

- The UE's RNTI.

All those information will be used for DL-SCH and PDSCH encoding and channel positions computation. In the uplink direction, after DCI/PDCCH transmission, the gNodeB waits for the UE to transmit the data and can then use the information from the DCI and the above configuration to extract the IQ samples, perform PUSCH equalization and decoding and finally decode UL-SCH.

The global procedure for data transmission on the gNodeB side is thus:

- Receive information/command from MAC.
- Build DCI payload.
- Compute PDSCH/PUSCH information.
- For downlink communication, use the above information for DL-SCH encoding, PDSCH encoding and PDSCH placement in the OFDM grid.
- For uplink communication, use it for computing the position of the PUSCH, wait for the UE to actually transmit PUSCH and then extract and decode it.

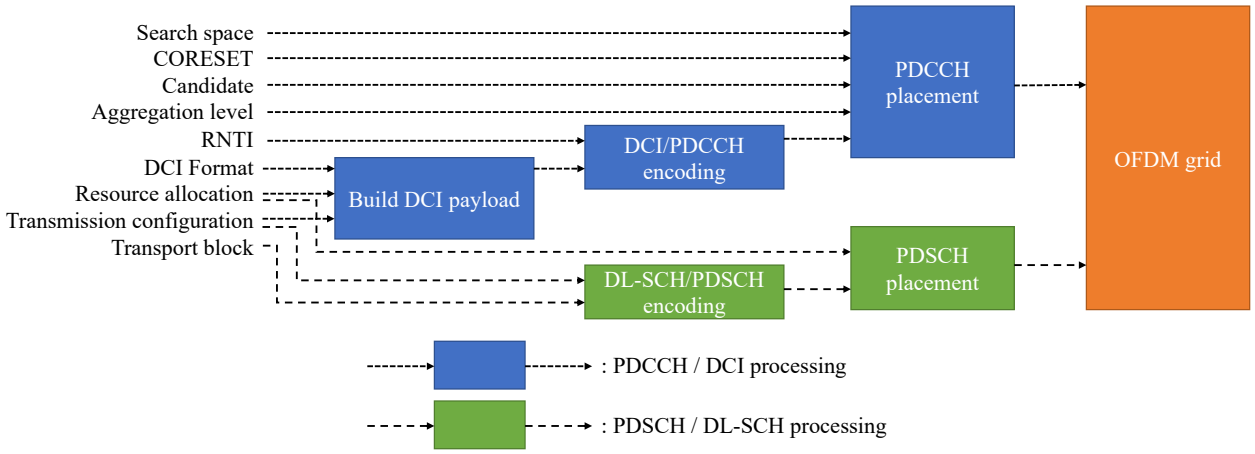


Figure 3.6: Downlink data transmission overview (gNodeB's perspective)

Figure 3.6 represents the downlink data transmission procedure on the gNodeB's side. The transmission configuration field contains information like the MCS or the PDSCH configuration.

3.2.5.2 UE

On the UE's side, the implementation of PDCCH decoding is based on blind search. The global process is given by TS38.213 section 10.1. At each procedure step, the UE knows which search space might use the gNodeB to transmit PDCCH. First, it determines the monitoring occasions using the search space and the CORESET. Then, it performs a blind search to find possible PDCCH transmissions: for each candidate of each aggregation level, it computes the corresponding IQ samples' positions, extracts them, and tries to decode PDCCH and DCI. If CRC validation is successful (after RNTI de-masking), a PDCCH transmission has been found. UE loops over all the aggregation levels and candidates as there might be multiple PDCCH elements within a single monitoring occasion.

For PDSCH decoding or PUSCH encoding, the first step on the UE's side is to extract and determine the parameters listed in the gNodeB's section 3.2.5.1. Then, for downlink, it can perform channel extraction and equalization, PDSCH decoding and DL-SCH decoding. The UE's reception of PDSCH is described in TS38.214 section 5.1. For uplink, it can perform UL-SCH and PUSCH encoding, perform channel mapping, OFDM modulation and finally transmit the signal. The UE's transmission of PUSCH is detailed in TS38.213 section 6.1.

Algorithm 2 details the DCI blind search for one monitoring occasion and must be repeated for all the occasions. For usual UL/DL communications, i.e. after RRC connection, UE's blind search must

be done continuously. Algorithms 3 and 4 detail the processing steps for a UE to decode DL-SCH transport blocks or to transmit UL-SCH. In algorithm 4, function *push_to_channel_mapper* performs OFDM modulation and signal transmission.

The global procedure for data communication on the UE's side is thus:

- Blind search and decode DCI.
- For downlink allocation, extract, decode DL-SCH/PDSCH and push the transport block to the MAC layer.
- For uplink allocation, notify the MAC layer and wait for the transport block. Then, perform UL-SCH/PUSCH encoding and transmit the signal.

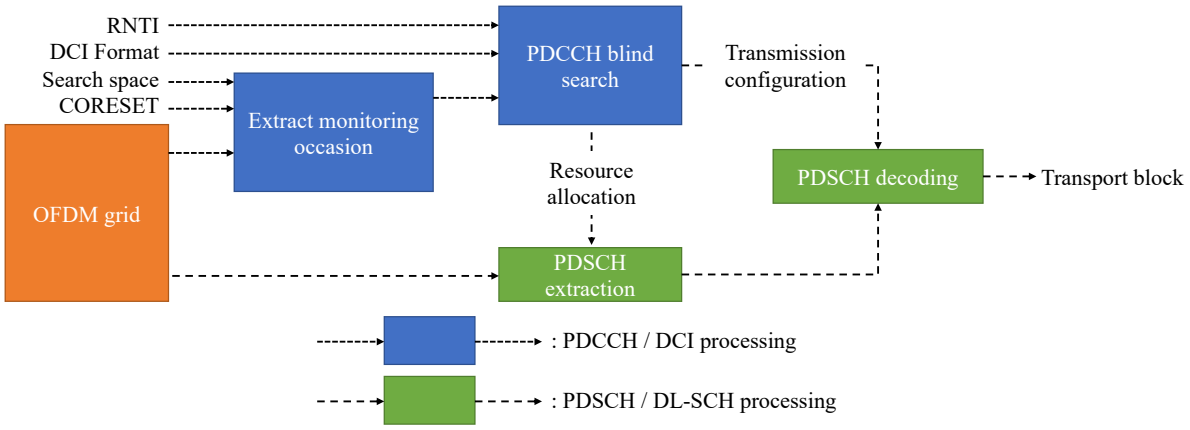


Figure 3.7: Downlink data transmission overview (UE's perspective)

Figure 3.7 represents the downlink data transmission procedure on the UE's side.

3.3 System Information Block 1

Once the UE is fully synchronized with the cell and has determined the network's time, a new transmission from the gNodeB to the UE is required to provide the UE with the default cell's RRC configuration. This default configuration is called SIB1 and enables the UE first to choose whether or not it is willing to attach the cell and to know how to connect to it. The SIB1 is transmitted periodically by the gNodeB, and UEs decode it upon successful cell synchronization and MIB reception. SIB1 transmission requires the implementation of PDCCH and PDSCH transmission (introduced in section 3.2) and reception at the gNodeB's and UE's side. Having a strong understanding of those channels is therefore required.

3.3.1 SIB1 payload

SIB1 is a RRC payload defined in TS38.331 [19]. It is passed through the RLC and MAC layers in transparent mode, which means that the RRC bits are directly transmitted to the PHY layer as being the input transport block to be encoded. It contains:

- $cellSelectionInfo$: Minimum thresholds for cell access like minimum reception power and quality.
- $cellAccessRelatedInfo$: Network and cell identity used by the UE to determine whether or not it has to connect this cell.
- $connEstFailureControl$: How to handle connection establishment failures.
- $servingCellConfigCommon$: Default cell's configuration. This includes:

Algorithm 2 UE DCI blind search

▷ Determine which search space and CORESET to use
$search\_space = determine\_search\_space()$
$coreset = determine\_coreset(search\_space)$
$received\_allocations = []$
▷ Determine monitoring occasion where DCI could be located
$mon\_occ = determine\_mon\_occ(received\_frame, search\_space, coreset)$
▷ Compute the possible DCI sizes of the different DCI formats
$dci\_sizes = compute\_possible\_sizes()$
▷ **Loop over all the possible aggregation levels**
for $agg\_level \in \{1, 2, 4, 8, 16\}$ **do**
    $num\_cand = compute\_candidates(agg\_level, coreset)$
    ▷ **Loop over all the candidates** of the current aggregation level
    for $cand \in [0, num\_candidates - 1]$ **do**
        ▷ **Identify** CCEs used by this candidate
        $cces = compute\_cces(agg\_level, cand)$
        ▷ **Identify** associated REG bundles
        $reg\_bundles = cce\_to\_reg\_mapping(cces)$
        ▷ **Compute** PDCCH and DMRS positions according to selected REG bundles
        $[pdcch\_pos, dmrs\_pos] = compute\_cand\_pos(mon\_occ, reg\_bundles)$
        ▷ Perform channel de-mapping
        $[pdcch\_samp, dmrs\_samp] = channel\_demap(received\_frame, pdcch\_pos, dmrs\_pos)$
        ▷ Compute expected DMRS sequence
        $dmrs\_sequence = compute\_dmrs\_sequence(dmrs\_positions, N_{ID}^{cell})$
        ▷ Perform channel estimation and equalization
        $coefficients = channel\_estimation(dmrs\_pos, dmrs\_samp, dmrs\_sequence)$
        $pdcch\_samp = channel\_equalization(pdcch\_samp, pdcch\_pos, coefficients)$
        ▷ **Decode** PDCCH
        $pdcch\_bits = pdcch\_decoding(pdcch\_samp)$
        ▷ **Loop** over all possible DCI sizes
        for $size \in sizes$ **do**
            ▷ **Decode** DCI
            $[validated, dci\_payload] = dci\_decoding(pdcch\_bits, size)$
            ▷ **If CRC validated**, one DCI payload has been found
            if *validated* **then**
                ▷ Decode information from DCI payload
                $dci\_object = extract\_fields\_from\_dci(dci\_payload, dci\_format)$
                $received\_allocations.push(dci\_object)$
            end if
        end for
    end for
end for

Algorithm 3 UE DL-SCH reception

▷ Loop over all the resources allocations
for $dci\_object \in received\_allocations$ **do**
    ▷ If downlink allocation
    if $dci\_object.direction == DL$ **then**
        ▷ **Compute** PDSCH and DMRS positions
        $[pdsch\_pos, dmrs\_pos] = compute\_pdsch\_pos(dci\_object)$
        ▷ Perform channel de-mapping
        $[pdsch\_samp, dmrs\_samp] = channel\_demap(received\_frame, pdsch\_pos, dmrs\_pos)$
        ▷ Compute expected DMRS sequence
        $dmrs\_sequence = compute\_dmrs\_sequence(dmrs\_positions, N_{ID}^{cell})$
        ▷ Perform channel estimation and equalization
        $coefficients = channel\_estimation(dmrs\_pos, dmrs\_samp, dmrs\_sequence)$
        $pdsch\_samp = channel\_equalization(pdsch\_samp, pdsch\_pos, coefficients)$
        ▷ **Decode** PDSCH
        $pdsch\_soft\_bits = pdsch\_decoding(pdsch\_samp)$
        ▷ **Decode** DL-SCH
        $[validated, dl\_sch\_payload] = dlsch\_decoding(pdsch\_soft\_bits)$
        ▷ **If CRC validated**, downlink communication is successful
        if *validated* **then**
            $push\_dl\_sch\_to\_mac(dl\_sch\_payload)$
        end if
    end if
end for

Algorithm 4 UE UL-SCH transmission

▷ Loop over all the resources allocations
for $dci\_object \in received\_allocations$ **do**
    ▷ If uplink allocation
    if $dci\_object.direction == UL$ **then**
        ▷ **Notify** upper layer that uplink resources have been granted and wait it to provide the transport block
        $ul\_sch\_payload = get\_ul\_sch\_from\_mac(dci\_object)$
        ▷ **Encode** the transport block
        $pusch\_bits = ulsch\_encoding(ul\_sch\_payload)$
        $[pusch\_samp, dmrs\_samp] = pusch\_encoding(pusch\_bits)$
        ▷ **Compute** PUSCH and DMRS positions
        $[pusch\_pos, dmrs\_pos] = compute\_pusch\_pos(dci\_object)$
        ▷ **Generate** signal and transmit PUSCH
        $push\_to\_channel\_mapper(pusch\_samp, dmrs\_samp, pusch\_pos, dmrs\_pos)$
    end if
end for

  - *downlinkConfigCommon* which contains the downlink configuration (initial downlink BWP, PDCCH and PDSCH configuration etc).
  - *uplinkConfigCommon* which contains uplink configuration (uplink BWP, RACH and PUSCH configuration etc).

- *ueTimersAndConstants*: Set of timers and constants to be used for the initial procedures.

3.3.2 Implementation

The SIB1 payload is carried by the DL-SCH transport channel and PDSCH physical channels. The DL-SCH transport block and corresponding resource allocation are received from the MAC layer. Following section 3.2, each PDSCH payload is notified to the UE by a PDCCH/DCI payload, and the first step is, therefore, to build the DCI payload and place it into the OFDM grid.

3.3.2.1 PDCCH placement and DCI payload

Some information must be determined before building the DCI payload, like the appropriate DCI format and size, search space, CORESET and RNTI. First, SIB1 transmission uses CORESET 0 and search space 0, which are the default cell's CORESET and search space. The configuration of the CORESET 0 and search space 0 are given by the parameter $PDDCHconfigcommon$ from the MIB. The procedure to determine the position of the PDCCH for SIB1 in the OFDM grid is detailed in appendix A.5.

On the UE's side, the MIB is used to determine the configuration of CORESET 0 and search space 0. The process is reversed on the gNodeB's side: the gNodeB first chooses CORESET 0 and search space 0 configurations and builds the MIB payload accordingly.

The DCI format used for SIB1 is DCI format 1_0 with SI-RNTI. SI-RNTI is the RNTI used to communicate System Information. As it is static and known by all the UEs, they can decode DCI elements which CRC is masked with SI-RNTI. As described in TS38.212 section 7.3.1.2.1 [11], the DCI payload size depends on $N_{RB}^{CORESET}$. The size of the other fields is fixed.

For example, if $N_{RB}^{CORESET} = 48$, the size of the frequency domain assignment field is 11 bits, and so the total DCI payload is 41 bits, including the reserved bits. The gNodeB uses this information to build the DCI payload and the UE also requires it for DCI blind search.

The main fields of the DCI format 1_0 with SI-RNTI are the frequency and time domain resources allocated for PDSCH:

- Frequency domain allocation noted Resource Indicator Value (RIV). On the gNodeB's side, this information is provided by the upper layers. The allocation is given by the number of RBs to be allocated (noted L_{RB}) and the first RB (noted RB_{start}). The RIV is computed following TS38.214 section 5.1.2.2.2 [24]:

$$RIV = N_{RB}^{CORESET} \cdot (L_{RB} - 1) + RB_{start}$$

  In the same way, at the UE's side, L_{RB} and RB_{start} can be recovered from RIV with:

$$L_{RB} = \left\lfloor \frac{RIV}{N_{RB}^{CORESET}} \right\rfloor + 1$$

  And

$$RB_{start} = RIV - \left((L_{RB} - 1) \cdot N_{RB}^{CORESET}\right)$$

- The time domain resource assignment is a 4 bits field which gives an index in Table 5.1.2.1.1-2 of TS38.214. K_0 is the slot offset between the PDCCH and the PDSCH, S is the first symbol allocated for PDSCH and L is the number of allocated symbols. For example, if PDSCH is transmitted in the same slot as the PDCCH ($K_0 = 0$), and if the allocation starts at symbol 2 and has a duration of 12 symbols, then this field should be 0.

The way frequency resource allocations are transmitted to the UE in the DCI is the same between the different DCI formats and for both directions (uplink and downlink): a RIV field is given to the UE which computes the corresponding L_{RB} and RB_{start}. This process will be used for future communications. However, unlike frequency resources assignment, time-domain resource allocation is not the same between DCI formats and procedures. Indeed, for future procedures, and both uplink and downlink assignments, the list of the possible time-domain assignments will be transmitted in RRC messages, and DCI will give a pointer to an element of this list. Details about the different DCI formats can be found in TS38.212 section 7.3.1 [11].

3.3.2.2 gNodeB

When the information regarding CORESET 0 and search space 0 are determined, and when DCI payload is built, DCI and DL-SCH payloads can be encoded. After encoding, those channels can be placed into the OFDM grid according to section 3.2 and transmitted.

3.3.2.3 UE

On the UE's side, the first step is to determine information about CORESET 0, search space 0 and the DCI's size to perform the blind search. The monitoring of PDCCH with search space 0 is detailed in TS38.213 section 13. Once DCI has been found and CRC is validated, DCI fields can be recovered from DCI payload, and those fields can be used to extract and decode PDSCH. Once PDSCH and DL-SCH have been decoded, SIB1 RRC message can be recovered.

3.4 Random Access

Term or notation	Definition
PRACH occasion	PRACH time and frequency position in the OFDM grid
PRACH format	Number of repetitions of the PRACH signal in one transmission
Preamble	Parameter used for PRACH sequence generation
L_{RA}	PRACH sequence size
N_{CS}	Cyclic shift size

Table 3.3: Terms and notations used in section 3.4

Once UEs are synchronized with the cell and have received the initial RRC configuration, they can determine whether or not to access the cell. This decision is taken with regard to different information like the cell identity and configuration.

The uplink radio connection must be initiated if the UE decides to access the cell. It is done through a procedure called RA which is detailed in TS38.213 section 8 [5] and in TS38.321 section 5.1 [26]. It is the first procedure that involves uplink transmissions. The RA procedure can be either Contention Based Random Access (CBRA) or Contention Free Random Access (CFRA). During RA, contention may happen when multiple UEs use the same resources and the same configuration, and CBRA must be used to resolve the contention. Contention resolution is the process by which the gNodeB selects one of the UEs for which contention happened and informs UEs of which one is selected. The selected one considers RA is successful, whereas the others must restart the procedure. On the opposite, when the gNodeB can notify the UE of which resources and configuration to use for RA, contention does not happen, and CFRA can be used. When a UE initializes a first RRC connection, it is not already known and configured by the gNodeB, so CBRA must be used.

As represented in Figure 3.8, CBRA is made of four steps:

- Physical Random Access Channel (PRACH) transmission (uplink - noted msg1): this is the initial uplink channel transmitted from UEs to the gNodeB. It enables the gNodeB to detect the presence of the UE in the cell and compute its Timing Advance (TA).

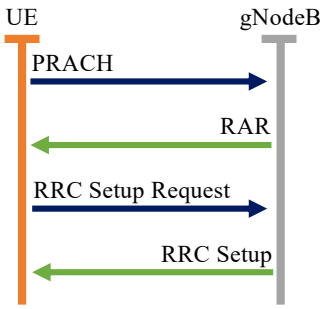


Figure 3.8: Random access procedure

- Random Access Response (RAR) (downlink - noted msg2): this is a MAC PDU that notifies the UE that PRACH has been received and contains two more information: the propagation delay, called the TA, and a resources allocation for subsequent uplink transmission.
- RRC Setup Request (uplink - noted msg3): this is an RRC message transmitted using the resources allocated in the RAR: the UE asks to establish an RRC connection with the gNodeB.
- RRC Setup (downlink - noted msg4): RRC response from the gNodeB, which contains all the initial configuration of the radio connection between the UE and the gNodeB.

3.4.1 Physical Random Access Channel *(TS38.211 section 6.3.3)*

The PRACH is the channel used by UEs to initiate a uplink radio connection with the gNodeB. Like PSS and SSS (and unlike MIB and SIB1), PRACH is a signal that does not contain data, making it a highly robust channel. The signal is generated depending on the preamble parameter, which is randomly selected. The PRACH configuration is received in SIB1 and enables the UE to determine PRACH occasions. A PRACH occasion is a time and frequency location where UEs can transmit PRACH. The configuration contained within the SIB1 enables the UE to determine the set of PRACH occasions and possible preambles. Based on that information, the UE can generate the PRACH sequence by randomly selecting one preamble and positioning it into the OFDM grid by selecting a time and frequency PRACH occasion. The OFDM modulation of PRACH is different from the usual one used in 5G (which is described in section 4.9). A dedicated function for PRACH time-domain signal generation must be implemented.

Beyond initiating the uplink connection, the purpose of the PRACH is to perform uplink synchronization. Indeed, until now, UEs are synchronized to the cell in the downlink direction, which means that UEs know the cell's time. However, uplink synchronization is also required and crucial as the different UEs are not located at the same distance of the gNodeB and do not have the same processing speed. This means that when a UE transmits a signal at a given time, it will arrive at the gNodeB with a delay which reflects the propagation and processing times. As those two delays are not the same from one UE to another, UEs transmissions do not arrive simultaneously at the gNodeB and might interfere with each other. This is why uplink synchronization is required. The gNodeB evaluates the processing and propagation delays, and the UE transmits uplink communications in advance. The TA is the parameter reflecting the two delays and is computed by the gNodeB upon reception of the PRACH and communicated to the UE in the RAR. UEs use TA to compute the required advance for uplink transmissions.

The PRACH channel is designed with regard to one primary constraint: the gNodeB must be able to decode multiple non-synchronized RAs located on the same time and frequency resources (i.e., PRACH occasion). Indeed, before RA, the gNodeB did not know the UE and could not allocate dedicated uplink resources for RA. Therefore, several UEs may be using the same PRACH occasion for RA transmission. In this case, the distinction between the different RAs at the gNodeB's level is based on the different preambles used by UEs for signal generation: PRACH signal is generated so that the gNodeB can distinguish simultaneous transmissions on the same occasion with different preambles. Zadoff-Chu sequences are used for PRACH generation as the correlation between two

shifted sequences generated from a single root sequence equals zero. Moreover, the correlation of two sequences generated with different root sequences is not equal to zero but still very low. The different PRACH signals are generated based on shifted Zadoff-Chu root sequences so that transmissions on the same PRACH occasion do not interfere with each other. If multiple UEs use the same preamble for PRACH transmission on a single PRACH occasion, the gNodeB cannot distinguish them (i.e. there is a PRACH contention), and future RA procedure will be used to resolve the contention.

3.4.1.1 Parameters

In this section, we present the main parameters that are used for PRACH generation and where they can be found. This is not an exhaustive list. On a UE's perspective, the entry point for PRACH configuration is $rachConfigCommon$ from SIB1:

- $rachConfigGeneric$: Object containing those information:
  - $prachConfigurationIndex$: Index in Tables 6.3.3.2-2 to 6.3.3.2-4 from TS38.211. Table 6.3.3.2-2 is used for Frequency Division Duplex (FDD) cells in FR1 bands, Table 6.3.3-2.3 is used for TDD cells in FR1 bands and 6.3.3.2-4 is used for TDD cells in FR2 bands. This table provides those information:
    - Preamble format: The format of the PRACH signal (cyclic prefix length, number of PRACH repetitions and post signal guard period. This parameter is used to determine L_{RA} in Tables 6.3.3.1-1 and 6.3.3.1-2 from TS38.211. L_{RA} is a fundamental parameter which gives the size of the PRACH sequence.
    - Frames number where PRACH can be transmitted, given by x and y such as PRACH can be transmitted in frames with $SFN \pmod x = y$. For example, if $x = 2$ and $y = 0$, PRACH can be transmitted every even frames.
    - Subframe number: subframe where PRACH can be transmitted.
    - Starting symbol in slot.
    - Number of slots containing PRACH in a subframe.
    - Number of PRACH occasions within a PRACH slot.
    - PRACH duration (in symbols).
  - $msg1FDM$ (Frequency Division Multiplexing): Number of PRACH occasions in frequency domain.
  - $msg1FrequencyStart$: Offset in RBs between the lowest RB allocated to the first frequency domain PRACH occasion and the Physical Resource Block (PRB)0 of the uplink BWP. This offset is given in number of PUSCH RBs.
  - $zeroCorrelationZoneConfig$: Index in Tables 6.3.3.1-5 to 6.3.3.1-7 from TS 38.211.
- $ssbperRACHOccasionAndCBPreamblesPerSSB$: Number of SSB per PRACH occasion and number of preambles per SSB.
- $msg1SubcarrierSpacing$: SCS used for PRACH transmission. It can be 1.25, 5, 15, 30, 60 or 120 kHz.
- $restrictedSetConfig$: Whether or not the number of cyclic shifts per root sequence is restricted. Together with $zeroCorrelationZoneConfig$, it enables to recover N_{CS} parameter (which will be explained below).
- $prachRootSequenceIndex$: Root sequence index used to generate the PRACH sequence.

3.4.1.2 Parameters selection

Before starting the cell, the gNodeB must determine the PRACH configuration. The two main parameters that must be determined are L_{RA}, which is the PRACH sequence length, and N_{CS}, which is the cyclic shift size. Multiple PRACH sequences can be generated by shifting a root sequence N_{CS} times. Given that the sequence's size is L_{RA}, $n_{shift}^{seq} = \lfloor L_{RA}/N_{CS} \rfloor$ sequences can be generated with a single root sequence. Those two parameters are selected to provide a good trade-off between those two constraints:

- The first one is that multiple cyclic shifted Zadoff-Chu sequences generated using the same root sequence are orthogonal to each other for all the possible shift values. On the other hand, sequences generated with different root sequences are not orthogonal. Given that the receiver detects PRACH by correlating the possible sequences with the received signal, the lower the number of available root sequences, the lower the reception complexity. The optimal case would be to have a unique root sequence with multiple shifts as all the possible sequences would be orthogonal to each other. It would make the reception correlation more selective on the gNodeB side. With regard to this constraint, the smallest N_{CS}, the highest the number of PRACH sequences generated with a single root sequence (which equals n_{shift}^{seq}), and therefore the lowest the complexity of the PRACH detection.
- On the other side, one main purpose of PRACH is to estimate TA, which corresponds to the UE's processing and propagation delays. It means that PRACH is always received with an offset which can therefore be of two kinds:
  - The first one is the offset due to the transmission and propagation delay.
  - The second one is the offset due to the cyclic shift of a given Zadoff-Chu root sequence.

  The gNodeB must be able to distinguish those two received offsets. For example, if $L_{RA} = 139$ and $N_{CS} = 69$, then two sequences can be generated with one root sequence: the first one ($v = 0$, where v denotes the cyclic shift index) with no cyclic shift and the second one ($v = 1$) with a 69 cyclic shift. The offset, proportionally to one PRACH symbol, from the second one to the first one is $N_{CS}/L_{RA} = 0.49 = 49\%$. For example, if the gNodeB receives a PRACH transmission with an offset of 70%, it must be able to determine whether or not the 70% are fully due to the TA (and therefore UE generated PRACH with $v = 0$) or if the sequence was generated with $v = 1$ which implies an offset of 49% and a TA of 21%. If the receiver cannot distinguish those two cases, then there might be confusion between the preamble used by the UE and the TA, which would certainly lead to a connection error. Thus, the receiver must distinguish those two kinds of offsets.

  This is where the PRACH configuration is critical. As the first offset is due to the TA, it is related to the distance between the UE and the gNodeB. The PRACH configuration must be chosen with regard to the cell's size so that the maximum processing and propagation delay is never bigger than the cyclic shift offset. With the same example, the cell coverage should be limited to the area where UEs' TA is smaller than 49% of a PRACH symbol. In this case, it becomes possible for the gNodeB to detect what is the offset due to the delay and what is the one due to the cyclic shift. The distinction is made as follows:

  - first, all the possible cyclic shifts are substracted to the received offset. With the same example, the two possible shifts are 0 and 49%, and the received offset is 70%. The subtraction gives $70 - 0 = 70\%$ and $70 - 49 = 21\%$.
  - Then, the candidate whose subtraction is higher than 0 and lower than the cyclic shift offset is selected as the correct one. In this case, the correct offset must be between 0 and 49%, which is the case of the second one. The receiver will therefore state that it received a PRACH transmission with a preamble generated with $v = 1$ and a TA of 21% of a PRACH symbol. The zero correlation zone is the distance, within a PRACH symbol, between the symbol's start and the cyclic shift offset (it equals 49% in this example).

Therefore, the PRACH configuration should be selected such as it uses the highest possible number of cyclic shifts per root sequence (for complexity reasons) and respects the coverage constraints. For example, as detailed earlier, with $L_{RA} = 139$ and $N_{CS} = 69$, the number of preambles that can be generated per root sequence is 2 and the maximum offset due to the UE's delay is 49% of an OFDM symbol but if $N_{CS} = 34$, the number of preambles that can be generated per root sequence is 4 and the maximum offset due to the UE's propagation and transmission delay is $34/139 = 0.24 = 24\%$. The second case decreases the receiver complexity as it requires the use of half the number of different root sequences for the same number of preambles. However, it also decreases the potential size of the cell as the maximum TA that the gNodeB could support is half the one of the first case.

The second key parameter for PRACH configuration is the preamble format, which gives the number of repetitions of the PRACH sequence. Indeed, the receiver can leverage the different repetitions to refine the PRACH detection. Therefore, the highest the number of repetitions, the finer the PRACH detection. On the other side, resources that can be used for PRACH must be reserved and cannot be used for uplink or downlink communications. The highest the number of repetitions, the highest the resource consumption of the PRACH. Therefore, the PRACH format must be chosen to provide a trade-off between resource consumption and PRACH detection quality.

3.4.1.3 UE

As done for BCH, DCI and DL-SCH encoding, we first introduce how PRACH is encoded and then how the receiver can handle it. We start here by explaining how a UE generates the PRACH signal and will in the next section explain how the gNodeB detects it. The PRACH signal generation depends on many parameters and can therefore be a very tricky process. In this section, we introduce PRACH generation using a simple configuration. Further information can be found in TS38.211 section 6.3.3 [21] and in section 7.2 and 13.1 of [25].

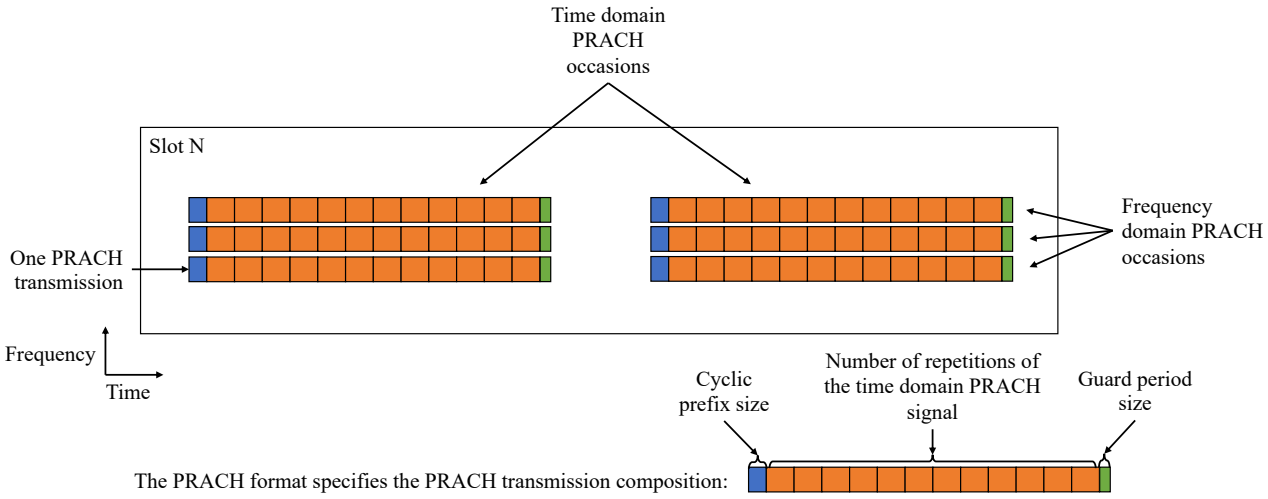


Figure 3.9: PRACH occasions and format

The details can be found in appendix A.6. Nevertheless, the main steps are:

1. Preamble selection. The preamble is required for PRACH signal generation. It is used to distinguish multiple PRACH transmissions performed on the same occasion. The preamble is a value between 0 and 63 but can be restricted for two reasons.

   - First, the gNodeB can configure the cell (through SIB1) so that only a sub-set of preambles are available for PRACH initial PRACH transmission. Other preambles can be used for CFRA or other purposes.
   - Moreover, the selection depends on the radio configuration, especially beamforming. Indeed, the set of available preambles is split between the different beams. When the UE searches for SSB, it selects a beam based on the reception quality. Selecting a preamble

that corresponds to the selected beam enables the UE to inform the gNodeB of the selected beam.

2. PRACH occasion selection. Depending on the cell's configuration, PRACH transmissions can be done at different times and frequencies:
   - In the frequency domain, multiple PRACH transmissions can happen simultaneously and are therefore stacked in the frequency domain. Selecting a frequency domain PRACH occasion comes to selecting an index in this stack.
   - In the time domain, the PRACH configuration specifies the frames, slots and symbols where PRACH can be transmitted. A time-domain PRACH occasion is a set of symbols where PRACH can be transmitted.
3. Frequency domain sequence generation. After selecting a preamble, the time domain Zadoff-Chu sequence can be computed and transformed into the frequency domain by performing a DFT.
4. Time-domain signal generation. Once the frequency domain samples have been computed, they can be placed in the band based on the selected frequency domain candidate. iFFT is therefore performed in order to generate the time-domain PRACH signal.
5. Final PRACH signal generation. The signal generated until now is only one repetition of the PRACH signal. The final PRACH signal contains a set of repetitions of the time domain signal and a cyclic prefix. The PRACH format gives the number of repetitions and the size of the cyclic prefix.
6. PRACH transmission. The final PRACH signal can be transmitted at the selected time domain PRACH occasion.

Figure 3.9 represents the PRACH occasions and format.

3.4.1.4 gNodeB

3.4.1.4.1 PRACH detection: PRACH detection is the process by which a gNodeB continuously searches for PRACH attempts and computes TAs. Some different methods and variations can be used, and the one described here is inspired by the method described in [27]. For the sake of simplicity, the algorithm is split into two parts that are the detection of a PRACH transmission (detected by a high correlation peak) and the determination of the corresponding shift and TA offset.

The first part, detailed in Algorithm 5 (for one time and frequency domain PRACH occasion noted $time\_prach\_occ$ and $freq\_prach\_occ$), consists in correlating in the frequency domain the received signal with all the possible PRACH signals generated from the possible root sequences and by selecting the correlation peaks that are above an input threshold.

The second part, detailed in Algorithm 6 determines the UE's cyclic shift by looping over all the possible shifts for each peak. For each one, the offset from the beginning of the PRACH symbol to the beginning of the received signal is computed by subtracting the corresponding cyclic shift from the candidate's received peak index. Negative offsets are then invalid as that would mean that UE is in advance compared to the gNodeB, and offsets above the zero correlation zone are also invalid as the cell must have been configured so that it is impossible with regard to the cell's size. The shift used by the UE is then the one that is inside the zero correlation zone.

3.4.2 Random Access Response

Once the PRACH has been transmitted from the UE to the gNodeB, the UE waits for the gNodeB's response, the RAR (TS38.213 section 8.2). RAR is a MAC message containing different information that enables the UE to send an RRC Setup Request message to initiate the RRC connection.

Algorithm 5 Detection of PRACH peaks

function DETECTPRACHTRANSMISSION(time_prach_occ, freq_prach_occ, max_p)
    ▷ v_max is n_{shift}^{seq}
    $v\_max = \lfloor L_{RA}/N_{CS} \rfloor$
    $prach\_preambles = []$
    $prach\_offsets = []$
    ▷ Compute zero correlation zone size
    $zero\_corr\_size = compute\_zero\_correlation\_zone\_size(L_{RA}, N_{CS})$
    $u\_p\_max = \lceil max\_p/v\_max \rceil$
    ▷ **Loop** over all the possible root sequences u'
    for $u\_p \in [0, u\_p\_max - 1]$ **do**
        ▷ **Compute** the corresponding preamble value
        $p = u\_p \cdot v\_max$
        $avg\_td\_correlation = []$
        ▷ **Generate** PRACH signal for preamble p
        $ref\_signal = generate\_prach\_signal(freq\_prach\_occ, p)$
        ▷ **Loop** over all the PRACH repetitions
        for $r \in [0, repetitions - 1]$ **do**
            ▷ Get corresponding received signal
            $input\_signal = get\_signal(time\_prach\_occ, r)$
            ▷ Get frequency domain signals and **correlate** them
            $fd\_input\_signal = fft(input\_signal)$
            $fd\_ref\_signal = fft(ref\_signal)$
            $fd\_correlation = correlate(fd\_input\_signal, fd\_ref\_signal)$
            ▷ **Transform** to time domain correlations and average over all the repetitions
            $td\_correlation = ifft(fd\_correlation)$
            $avg\_td\_correlation += td\_correlation/repetitions$
        end for
        for $[value, index] \in avg\_td\_correlation$ **do**
            ▷ **If** correlation is greater than an input threshold, a PRACH transmission has been detected
            if $value > threshold$ **then**
                ▷ **Determine** the cyclic shift and TA offset corresponding to this peak
                $[preamble, ta\_offset] = deter\_shift\_and\_offset(u\_p, avg\_td\_correlation, index)$
                $prach\_preambles.push(preamble)$
                $prach\_offsets.push(ta\_offset)$
            end if
        end for
    end for
    return $[prach\_preambles, prach\_offsets]$
end function

Algorithm 6 PRACH shift and offset determination

```
function DETER_SHIFT_AND_OFFSET(u_p, avg_td_correlation, index)
    ▷ Loop over all possible shifts
    for v ∈ [0, v_max − 1] do
        ▷ Compute cyclic shift size, proportionally to the PRACH symbol duration
        shift_size = compute_shift_size(N_CS, L_RA, v)
        ▷ Compute offset between the correlation peak and the shift size
        offset = index − shift_size
        ▷ If offset is correct
        if offset ∈ [0, zero_corr_size − 1] then
            ▷ Offset is within the zero correlation zone
            ▷ Return the associated preamble and offset
            return [u_p · v_max + v, offset]
        end if
    end for
end function
```

3.4.2.1 MAC payload *(TS38.321 section 6.1.5)*

This section gives a quick introduction to the RAR MAC payload. Like any other MAC PDU, RAR PDU comprises different sub-PDUs and subheaders. There can be three types of sub-PDUs and subheaders for the RAR:

- RAR sub-PDU (TS38.321 section 6.2.3): This is the RAR, which contains that information:
  - TA: as explained in the previous section, the PRACH transmission enables the gNodeB to estimate the transmission and propagation delay of the different UEs. The TA command contains a value that corresponds to an estimated delay. The UE can compensate it by sending all the future transmissions in advance, using the TA value. It enables future uplink transmissions to be synchronized with the cell.
  - Uplink grant: This is a uplink resource allocation that the UE uses to transmit the RRC Setup Request (msg3). It mainly contains:
    - Frequency resource allocation: RBs allocated to UE for msg3.
    - Time resources allocation: symbols and slot where msg3 can be transmitted.
    - MCS: modulation scheme and code rate to be applied for transmission.
  - Temporary C-RNTI: TC-RNTI is the RNTI used for communications with the UE until the RA procedure is finished. After the end of this procedure (i.e. after contention resolution), TC-RNTI becomes C-RNTI and is then used for all the future uplink and downlink communications.
- RAPID subheader (TS38.321 section 6.2.2): This sub-PDU contains the Random Access Preamble Identity (RAPID) corresponding to the current RAR sub-PDU. It is a 6 bits value (between 0 and 63) which contains the preamble used for PRACH transmission to which the RAR is responding. It enables UEs to differentiate multiple RARs when multiple preambles have been used on a single PRACH occasion.
- Back-off Indicator (BI) subheader (TS38.321 section 6.2.2): This sub-PDU is transmitted to the UE when the gNodeB could not handle the transmitted PRACH for overload reasons. In this case, BI is used to notify the UE when to send another PRACH transmission. This field is an index in Table 7.2-1 from TS38.321 [26] and gives a value in milliseconds.

3.4.2.2 Transmission

The RAR is transmitted to the UE using a classic downlink communication scheme, i.e. using PDCCH and PDSCH (see section 3.2). As for SIB1 transmission (section 3.3), CORESET, search space and

RNTI to be used must be determined before encoding or extracting RAR:

- search space: the search space for RAR is identified in SIB1 by the parameter *raSearchSpace*. It gives the ID of a search space that must be defined in the SIB1 payload.
- CORESET: the CORESET used for RAR is identified in the search space. It can be a CORESET defined in SIB1 or the CORESET 0 used for SIB1 transmission.
- RNTI: The RNTI to be used for DCI CRC masking (detailed in section 3.2.2.1) is RA-RNTI. The RA-RNTI is not dedicated to one UE but depends on the time and frequency occasion used for PRACH transmission. Both the UE and the gNodeB can compute it. It enables to identify a PRACH occasion. Two UEs using the same preamble on two different occasions will not receive the DCI for RAR with the same RA-RNTI, and two UEs using the same occasion and different preambles will not receive the same RAR message (RAPID subheader will differ). The RA-RNTI is computed according to the following Equation (from TS38.321 section 5.1.3):

$$RA - RNTI = 1 + s\_id + 14 \cdot t\_id + 14 \cdot 80 \cdot f\_id + 14 \cdot 80 \cdot 8 \cdot ul\_carrier\_id$$

  Where s_id and t_id are the indexes of the first symbol and the first slot of the PRACH occasion, f_id is the frequency PRACH occasion index and $ul\_carrier\_id$ is 0 for normal uplink carrier and 1 when PRACH is transmitted using supplementary uplink carrier.

The DCI payload is DCI Format 1_0 using RA-RNTI. This format is defined in TS38.211 section 7.3.1.2.1 and is pretty much the same as for DCI Format 1_0 using SI-RNTI (see section 3.3.2.1), except that redundancy version and system information indicator are not present, but another field Transport Block scaling is present. This field is used to help the UE to determine the transport block size.

The MAC layer of the UE must receive the RAR before *raResponseWindow* timer expires. This timer is started when UE transmits PRACH and should not expire before the reception of the RAR. If it expires, the RA procedure is considered to be failed. *raResponseWindow* is provided by SIB1 and is expressed in slots.

3.4.2.2.1 gNodeB: On the gNodeB's side, the RAR transmission is made by generating a DCI payload Format 1_0 using RA-RNTI and by encoding it as well as encoding the RAR transport block using DL-SCH and PDSCH encoding procedure. The positions of the PDSCH IQ samples in the OFDM grid is given by the MAC layer and the positions of the PDCCH IQ samples depend on the aggregation level and candidate used for PDCCH transmission. This process is detailed in section 3.2.5.1.

3.4.2.2.2 UE: On the UE's side, the reception of the RAR is done by first searching for PDCCH transmission with RA-RNTI and by then decoding corresponding PDSCH and DL-SCH payload. As for SIB1 blind search, RAR search requires the UE to determine the search space, CORESET and RNTI used for PDCCH/DCI transmission. Then, it performs blind search by looping over all the possible aggregation levels and candidates and trying to decode associated DCI payload. When DCI CRC is validated, DCI payload can be extracted and PDSCH position in the grid can be determined. PDSCH can then be extracted and decoded to recover the MAC RAR transport block. This process is detailed in section 3.2.5.2.

3.4.3 RRC Setup Request

Once RAR has been received by the UE and to finalize the RRC connection, UE transmits RRC Setup Request to the gNodeB in order to get the RRC configuration and to establish the RRC connection. This process is described in TS38.213 section 8.3. RRC Setup Request is an RRC message carried by the Common Control Channel (CCCH) MAC logical channel. It is passed through RLC in transparent mode. It contains the RRC connection establishment cause and a long random bits sequence used

for contention resolution. The RRC payload is encapsulated in a MAC PDU which finally gives a PHY transport block. This transport block is carried by the UL-SCH transport channel and the PUSCH physical channel defined in section 3.2.4. The UE generates the RRC payload, encapsulates it in a MAC PDU and can then perform UL-SCH and PUSCH encoding. The generated PUSCH and DMRS IQ samples are placed in positions allocated in RAR *UL grant* field and transmitted. On the gNodeB's side, the RRC Setup Request can be decoded by extracting the samples allocated in RAR *UL grant* and by performing PUSCH and UL-SCH decoding.

3.4.4 RRC Setup

Upon reception of the RRC Setup Request message, the gNodeB generates the RRC Setup message containing the UE's RRC configuration and encapsulates it in a MAC PDU together with a MAC Control Element (CE) called UE Contention Resolution Identity (UECRI).

UECRI is used for contention resolution and is the bit's representation of the RRC Setup Request message. As the RRC Setup Request message contains a long random bits sequence, it is used to differentiate the different UEs that would have experienced contention by selecting the same preamble on the same PRACH occasion. When, the UE receives the msg4 transmission containing the RRC Setup and the UECRI, it is able to compare UECRI with the transmitted RRC Setup Request. UE considers RA is successful and that contention is solved if the received UECRI matches the transmitted RRC Setup Request. Otherwise, it considers that RA failed and will try a new RA procedure. If RA is successful, the TC-RNTI used for RA procedure becomes the C-RNTI and will be used for all the future communications with the UE.

Once the PHY transport block is ready to be transmitted from the gNodeB to the UE, the DCI payload can be built. For msg4, the DCI format used is DCI Format 1_0 with TC-RNTI. This format is described in TS38.212 section 7.3.1.2.1. The TC-RNTI value to be used is the one contained in the RAR message and the search space and CORESET are the same as for RAR (i.e. *raSearchSpace* from SIB1). After building the DCI payload, DCI and PDCCH encoding can be performed and the transport block can be DL-SCH and PDSCH encoded. The IQ samples are placed in the OFDM grid and transmitted.

On the UE's side, the first step is to perform blind search on the RA search space and CORESET, and to decode the DCI payload. Once DCI is decoded, the corresponding PDSCH IQ samples can be extracted and decoded to recover the MAC and RRC messages.

After RRC Setup message transmission, the RRC connection is established. The RRC configuration can be applied. The PHY layer is fully operational for future transmissions like upper layers procedures and data communications.

3.5 Data communication after RRC connection setup

Once the RRC connection has been established, the PHY layer is fully configured and initialized. Then, it can be used by upper layers for uplink and downlink data communications. This section details the process for uplink and downlink communications after RRC connection.

3.5.1 DCI payloads

Once the RRC connection has been set up, the formats used for DCI transmissions are 1_1 for downlink and 0_1 for uplink. Those formats are defined in TS38.212 sections 7.3.1.2.2 and 7.3.1.1.2. The content can vary depending on the configuration, but the main fields are the time and frequency allocation, the modulation scheme and code rate, the redundancy version, and finally, the HARQ process:

- Frequency domain resources allocation (TS38.214 section 5.1.2.2 [24]): this field is given by an indicator called RIV. The number of RBs (noted L_{RB}) and the first allocated RB (noted RB_{start}) can be recovered from the RIV following the method described in section 3.3.2.1.

- Time domain resources allocation (TS38.214 section 5.1.2.1 [24]): this field is given by an indicator named time domain resources index. It gives an index in a list of possible time domain allocations. This list is initially provided in the SIB1 message ($pdsch-TimeDomainAllocationList$ for downlink and $pusch-TimeDomainAllocationList$ for uplink) and can be updated in later RRC configurations. One element of this list gives the mapping type (configuration for PDSCH and PUSCH) and the Start and Length Indicator Value (SLIV). The index of the first allocated symbol S and number of symbols L can be recovered from the SLIV. The possible combinations of S and L is given in Table 5.1.2.1-1 from TS38.214 [24] and the SLIV is computed using those two equations:

  $$SLIV = 14 \cdot (L-1) + S$$

  If $(L-1) \leq 7$ and

  $$SLIV = 14 \cdot (14 - L + 1) + (14 - 1 - S)$$

  Otherwise.

- The MCS: this field gives an index in Tables 5.1.3.1-1, 5.1.3.1-2 and 5.1.3.1-3 in TS38.214 [24]. The table used depends on the configuration and is usually Table 5.1.3.1-1. The field Modulation Order noted Q_m gives the modulation scheme to be used (2 means BPSK, 4 means QPSK, 6 means 16-QAM, 8 means 64-QAM, and 10 means 256-QAM). The field code rate noted R gives the proportion between the number of transport block bits and the number of code bits added by the channel coding function.

- HARQ process number and redundancy version: information about MAC layer HARQ process. The MAC HARQ process is a procedure defined in TS38.321 sections 5.3.2 and 5.4.2 [26] which enables the correction of transmissions errors. It is done in multiple parallel processes. The HARQ process number is the index of the process to which the transmission belongs, and the redundancy version is used to determine which part of the transport block is being transmitted or retransmitted.

3.5.2 gNodeB

On the gNodeB's side, the input for data transmission is the reception of a downlink transport block or uplink resource grant from the MAC layer. The command contains the data (transport block to be transmitted in the downlink, empty in uplink) and the metadata, which contains the RNTI, the resource grant and transmission configuration (modulation, code rate, HARQ process etc).

For downlink communication, the procedure is:

1. Reception of a command from the MAC layer.
2. - The DCI payload (DCI Format 1_1) is generated based on the metadata (resource allocation, RNTI, search space and CORESET to be used, transmission configuration etc) and DCI encoded.
   - The transport block is DL-SCH encoded.
3. - DCI bits are PDCCH encoded, and DMRS sequence and IQ samples positions are computed.
   - DL-SCH bits are PDSCH encoded, and DMRS sequence and IQ samples positions are computed.
4. OFDM modulation is performed and signal is transmitted.

For uplink communication, the procedure is:

1. Uplink grant is received from MAC layer.

2. The DCI payload (DCI Format 0_1) is generated based on the metadata (resource allocation, RNTI, search space and CORESET to be used, transmission configuration etc) and DCI encoded.

3. DCI bits are PDCCH encoded, and DMRS sequence and IQ samples positions are computed.

4. OFDM modulation is performed and signal is transmitted.

5. The gNodeB waits until grant time has come, extracts the IQ samples, performs equalization, and decodes PUSCH and UL-SCH.

3.5.3 UE

On the UE's side, the downlink process is:

1. Blind search for DCI transmissions.

2. DMRS sequence and IQ samples positions computation, PDSCH extraction and decoding.

3. Received transport block is pushed to MAC layer.

The global uplink process is:

1. Blind search for DCI transmissions.

2. PHY layer gets transport block from MAC layer.

3. UL-SCH and PUSCH encoding, DMRS sequence and IQ samples positions computation and OFDM modulation and transmission.

3.6 Conclusion

In this chapter, we explained how the gNodeB notifies UEs of allocated resources and how UEs use that information to decode or transmit data. We explained how this process is applied to SIB1 transmission, which contains the default cell's RRC configuration. We detailed the RA procedure used by UEs to access the cell, perform uplink synchronization and establish the RRC connection. Finally, we explained how the gNodeB and UEs can communicate after RRC connection establishment. The different procedures and mechanisms implemented in this chapter and chapter 2 enable to build a minimal physical layer. It can now be used for upper layers purposes like network attachment and user traffic transmission. Other PHY layer procedures exist for RRC configuration update, MIMO configuration, beamforming, mobility etc, but they are not mandatory.

Chapter 4

Physical layer algorithms

Contribution

The main contribution of this chapter is to expose feasible algorithms that can be implemented for the non-standardized functions. Those functions are the object of a lot of research, but the proposed algorithms are not always implementable or understandable. This chapter gives a concrete overview of simple algorithms that work in good radio conditions and are implemented in *free5GRAN*. It provides the reader with a complete understanding of the physical layer, including both the procedures (introduced in chapters 2 and 3) and the associated algorithms.

4.1 Introduction

All the procedures implemented in chapters 2 and 3 are precisely defined in the standard and enable the gNodeB and UEs to synchronize with each other and to exchange data. However, those procedures rely on functions not precisely defined in the standard. The functions' input, output, and global behavior are defined, but the algorithms implemented within the functions to achieve the standardized behavior are not defined. Indeed, the purpose of the standard is to enable components from different manufacturers to work together on the same network by defining the procedures, mechanisms, and functions chaining. Nevertheless, the way functions are implemented is manufacturer-specific as it is where resides the core value of the implementation: manufacturers with better algorithms provide better products.

Unlike chapters 2 and 3, the purpose of this chapter is not to give a global understanding of the standard and procedures but to introduce the main functions used across procedures. For each function, we delimit what is defined by the standard and what is manufacturer-specific. Furthermore, we propose simple algorithms for non-standardized functions. The algorithms introduced in this chapter are the ones used in *free5GRAN*. They are selected to work well in our environment, a Faraday cage, where the radio conditions are good. Furthermore, they provide a good balance between optimization and readability.

In this chapter, the different functions that are not standardized are reviewed, and possible algorithms are proposed. First, the functions used for transport channel encoding and decoding are detailed. Then, the functions used for physical channel encoding and decoding are detailed. Finally, the functions used for signal processing are detailed.

4.2 Cyclic Redundancy Check *(Transport channel processing - TS38.212 section 5.1)*

In 5G, 6 CRC polynoms are defined, as detailed in TS 38.212 section 5.1 [11] and in Table 4.1. The different polynoms are used for different use-cases, depending on the sequence's length and reliability requirement. The sequence on which the CRC is computed can be either the full transport block or one code block when there are multiple code blocks per transport block.

Name	Channel	Designed for
g_{CRC24A}	UL/DL-SCH transport block	Large to ultra large sequences
g_{CRC24B}	UL/DL-SCH code block	Medium sized sequences
g_{CRC24C}	BCH and DCI transport block	Small sequences with high reliability requirement
g_{CRC16}	UL/DL-SCH transport block	Small sequences
g_{CRC11}	UCI transport block	Very small sequences
g_{CRC6}	UCI transport block	Very small sequences

Table 4.1: 5G CRC polynoms

First, g_{CRC6} and g_{CRC11} are defined for small UCI transport blocks. Then, g_{CRC24C} is dedicated to small transport blocks with high-reliability requirements, which is the case of the BCH and DCI transport channels. Furthermore, for UL/DL-SCH sequences with normal reliability requirements, the polynom depends on the sequence's size. For small transport blocks whose size is smaller than 3824, g_{CRC16} is used whereas, for large transport blocks (whose maximum size is of the order of a million bits), g_{CRC24A} is used. Finally, when UL/DL-SCH transport block size is larger than 8448 or 3840 (depending on the LDPC configuration), it is split into code blocks. Each code block is applied a CRC generated with g_{CRC24B}.

4.2.1 Computing CRC

The CRC is the remainder of the division of the input sequence by the CRC polynom.

4.2.2 Validating CRC

CRC validation is made by dividing the received sequence (bits sequence + CRC) by the polynom. The CRC is validated if the remainder equals 0. The code snippet for CRC validation in *free5GRAN* is given in appendix A.7.

4.3 Channel coding *(Transport channel processing - TS38.212 section 5.3)*

Two methods are defined for channel coding in 5G: polar coding and LDPC. Polar coding is used for channels with high robustness requirements like BCH and DCI. LDPC is used for channels with high throughput requirements (DL-SCH and UL-SCH). Those two coding techniques are described in chapter 1.

4.3.1 Polar coding *(TS38.212 section 5.3.1)*

As introduced in section 1.3.2.3.1, polar coding consists in polarizing the channel by applying polar transform. Before applying polar transform, a few steps are required. The details of this procedure are given for the encoding and decoding direction in appendix A.8.

The overall process is:

1. Interleaving: the input sequence of K bits is mixed so that the CRC bits are mixed with the data bits. In case of channel errors, there is no contiguous data or CRC loss.
2. Most reliable positions selection: once the bits are interleaved, a set of K most reliable positions are determined from the table provided in the standard (Table 5.3.1.2-1 from TS38.212 [11]). The most reliable positions are placed in a set called $\bar{Q}_I^N$.
3. An intermediate sequence with size N called u is generated. It contains the input bits placed at the most reliable positions of the channel and the frozen bits (set to 0) in the other positions.
4. Polar transform: this is the actual polar encoding done by multiplying the intermediate sequence u with the matrix G_N.

4.3.1.1 Successive cancellation decoding

Different algorithms can be used to decode polar codes, and successive cancellation is the principal. An introduction to this algorithm and its application to polar codes is given in [3]. In this section, the principle of the successive cancellation algorithm is explained. The binary tree representation of polar codes, defined in section 1.3.2.3.1 must be used to understand this algorithm. The decoding is made sub-tree by sub-tree, starting at the root node. The inputs of polar decoding are the received soft bits where bits values are not a hard 0 or 1 value but an Log-Likelihood Ratio (LLR) belief which enables to compute the probability for the bit to be equal to 0 or 1. More details about soft bits are given in section 4.6. Soft bits with negative value are likely to be 1, and soft bits with positive value are likely to be 0.

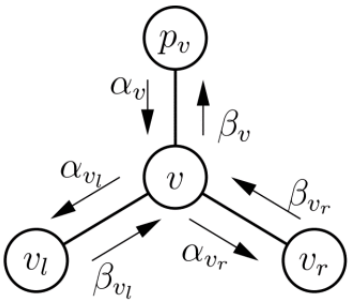


Figure 4.1: Successive cancellation for one sub-tree, from [3]

4.3.1.1.1 Non-leaf node: For one sub-tree v which is not a leaf, the successive cancellation process is represented in Figure 4.1:

1. Node v receives a set of beliefs α_v.
2. It computes a set of beliefs α_{v_l} that will be transmitted to its left child node using Equation:

$$\forall i \in [0, 2^{n-d_v-1} - 1], \alpha_{v_l}(i) = \alpha_v(i) \boxplus \alpha_v(i + 2^{n-d_v-1})$$

   Where d_v is the depth of node v and $\boxplus$ is called the activation function. It is defined by:

$$x \boxplus y = 2 \cdot atanh\left(tanh\left(\frac{x}{2}\right) \cdot tanh\left(\frac{y}{2}\right)\right)$$

3. Left child v_l returns its set of hard decisions which is noted β_{v_l}.
4. Node v computes a set of beliefs α_{v_r} to be transmitted to its right child using Equation:

$$\forall i \in [0, 2^{n-d_v-1} - 1], \alpha_{v_r}(i) = \alpha_v(i)(1 - 2\beta_{v_l}(i)) + \alpha_v(i + 2^{n-d_v-1})$$

5. Right child v_r returns its set of hard decisions which is noted β_{v_r}.
6. Node v returns its hard decision which is $\beta_v = \{\beta_{v_l} + \beta_{v_r}, \beta_{v_r}\}$

For root node r, the input beliefs α_r are the received soft bits. The final corrected codeword is given by the set of hard-decisions made by the leaf nodes.

4.3.1.1.2 Leaf node: If node v is a leaf, the input belief α_v is not a set but only contains one element. The returned hard decision is $\beta_v = 1$ if $\alpha_v < 0$ and $\beta_v = 0$ otherwise.

4.3.1.1.3 Successive cancellation list decoding: Given that successive cancellation might not decode all the errors and might not work perfectly in real implementations, it has been derived to build another algorithm called successive cancellation list decoding. The core concept is the same as for successive cancellation. The main difference is that, at each step, it does not return a hard decision β_v but a set of possible hard decisions together with a metric called the penalty metric. The highest the penalty metric, the lowest the probability for the hard decision to be the good one. At each step, a set of possible hard decisions is kept by selecting the candidates with the lowest penalty metric. Therefore, the output of this algorithm is not a single hard decision codeword but a set of possible codewords. The correct one can be identified at the next step by validating the CRC. A detailed explanation of this algorithm is given in [28].

4.3.2 Low Density Parity Check *(TS38.212 section 5.3.2)*

Term or notation	Definition
K	Input size
N	Output size
Z_c	Lifting size
i_{LS}	Set index, depends on Z_c
BG	Base graph used for parity check matrix generation
k	Number of groups of Z_c bits in the input ($k = K/Z_c$)
n	Number of groups of Z_c bits in the output ($n = N/Z_c$)
m	Number of groups of Z_c check nodes used for LDPC
c	Input sequence
c'	Input sequence, by groups of Z_c bits
w	Parity bits sequence, by groups of Z_c bits
H	Parity check matrix
H_{BG}	Parity check intermediate matrix
A, B, F_1, F_2	Sub matrices of H_{BG}
d'	Received soft bits (for LDPC decoding)
b_j	Hard decision corresponding to soft bit d'_j

Table 4.2: Terms and notations used in section 4.3.2

LDPC is a parity check coding method which concept is detailed in section 1.3.2.3.2. 5G LDPC process is made of 4 steps as described in TS38.212 5.3.2 [11]. Steps 1, 2 and 4 are not detailed here. The actual LDPC encoding function is located in step 3.

Before encoding or decoding, some parameters must be determined. First, the lifting size Z_c has to be computed following the procedure detailed in TS38.212 section 5.2.2. The lifting size is the number by which the bits are processed in 5G LDPC: bits are not processed individually but by groups of Z_c bits.

Once Z_c is computed, the set index (noted i_{LS}) can be found in Table 5.3.2-1 of TS38.212. The set index is used to build the parity check matrix and it is associated with a lifting size. Then, the output size (noted N) can be computed: $N = 66 \cdot Z_c$ for BG 1 and $N = 50 \cdot Z_c$ for BG 2.

The parity check matrix must also be computed based on the above parameters. As described in section 1.3.2.3.2, in 5G, the parity check matrix H is given by an intermediate matrix H_{BG}.

4.3.2.1 Encoding

Then, LDPC encoding can be performed. The objective of this step is to compute the parity bits sequence w. In this section, we use a method described and proved in [29], and we recapitulate the main processing steps here. This method processes input and parity bits by groups of Z_c and therefore uses parity check matrix H_{BG}. We note k the number of groups of Z_c bits in the input ($k = K/Z_c$). The encoding method is based on two observations about the parity check matrix. The first statement

is that H_{BG} is always built in such a way that it can be split in 6 sub-matrices (A, B, F_1, F_2, a zeros sub-matrix and an identity sub-matrix I) as shown in Figure 4.2.

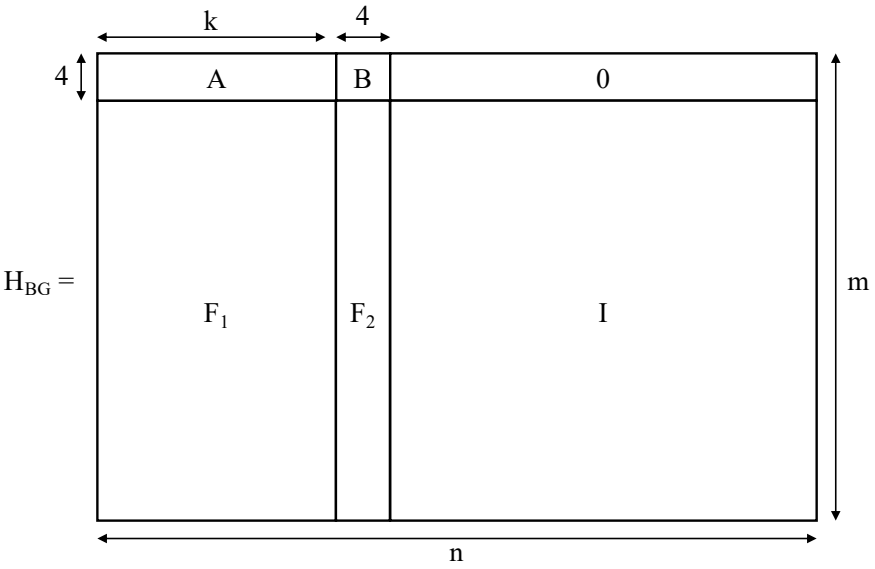


Figure 4.2: H_{BG} matrix can be split into 6 sub-matrices

Based on this observation, parity bits w generation can be split into two parts $w = [w_1, w_2]$. We note $w = w_{1,0}, ..., w_{1,3}, w_{2,0}, ..., w_{2,m-5}$ and $c = c'_0, ..., c'_{k-1}$ (each $w_{j,i}$ and c'_i is a Z_c long sub-sequence of w and c). w must be generated such as $H_{BG} \cdot [c'_0, ..., c'_{k-1}, w_{1,0}, ..., w_{1,3}, w_{2,0}, ..., w_{2,m-5}]^T = 0$ which gives the following Equation:

$$\begin{pmatrix} A & B & 0 \\ F_1 & F_2 & I \end{pmatrix} \cdot \begin{pmatrix} c^T \\ w_1^T \\ w_2^T \end{pmatrix} = 0$$

Which leads to:

$$\begin{cases} A \cdot c^T + B \cdot w_1^T = 0 \\ F_1 \cdot c^T + F_2 \cdot w_1^T + w_2^T = 0 \end{cases}$$

Furthermore, sub-matrix B can only take 4 different values depending on BG and i_{LS} (B_1 is used for BG 1 and $i_{LS} \in \{0, 1, 2, 3, 4, 5, 7\}$, B_2 is used for BG 1 and $i_{LS} = 6$, B_3 is used for BG 2 and $i_{LS} \in \{0, 1, 2, 4, 5, 6\}$ and B_4 is used for BG 2 and $i_{LS} \in \{3, 7\}$):

$$B_1 = \begin{pmatrix} 1 & 0 & -1 & -1 \\ 0 & 0 & 0 & -1 \\ -1 & -1 & 0 & 0 \\ 1 & -1 & -1 & 0 \end{pmatrix} \quad , \quad B_2 = \begin{pmatrix} 0 & 0 & -1 & -1 \\ 105 & 0 & 0 & -1 \\ -1 & -1 & 0 & 0 \\ 0 & -1 & -1 & 0 \end{pmatrix}$$

$$B_3 = \begin{pmatrix} 0 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 \\ 1 & -1 & 0 & 0 \\ 0 & -1 & -1 & 0 \end{pmatrix} \quad , \quad B_4 = \begin{pmatrix} 1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 \\ 0 & -1 & 0 & 0 \\ 1 & -1 & -1 & 0 \end{pmatrix}$$

We note $a^{(l)}$ the sequence a left shifted l times (and thus $a^{(-l)}$ the sequence a right shifted l times). The first step of the parity bits generation computes w_1 parity bits and the second one computes w_2:

- First, compute $w_1 = (w_{1,0}, w_{1,1}, w_{1,2}, w_{1,3})$, using the equation system corresponding to the BG

and i_{LS}:

$$B_1 : \begin{cases} w_{1,0} = \sum_{i=0}^{3}\sum_{j=0}^{k-1} c'^{(a_{i,j})}_j \\ w_{1,1} = \sum_{j=0}^{k-1} c'^{(a_{0,j})}_j + w^{(1)}_{1,0} \\ w_{1,3} = \sum_{j=0}^{k-1} c'^{(a_{3,j})}_j + w^{(1)}_{1,0} \\ w_{1,2} = \sum_{j=0}^{k-1} c'^{(a_{2,j})}_j + w_{1,3} \end{cases} \quad B_2 : \begin{cases} w_{1,0} = \left(\sum_{i=0}^{3}\sum_{j=0}^{k-1} c'^{(a_{i,j})}_j\right)^{(-105 \pmod{Z_c})} \\ w_{1,1} = \sum_{j=0}^{k-1} c'^{(a_{0,j})}_j + w_{1,0} \\ w_{1,3} = \sum_{j=0}^{k-1} c'^{(a_{3,j})}_j + w_{1,0} \\ w_{1,2} = \sum_{j=0}^{k-1} c'^{(a_{2,j})}_j + w_{1,3} \end{cases}$$

$$B_3 : \begin{cases} w_{1,0} = \left(\sum_{i=0}^{3}\sum_{j=0}^{k-1} c'^{(a_{i,j})}_j\right)^{(-1)} \\ w_{1,1} = \sum_{j=0}^{k-1} c'^{(a_{0,j})}_j + w_{1,0} \\ w_{1,2} = \sum_{j=0}^{k-1} c'^{(a_{1,j})}_j + w_{1,1} \\ w_{1,3} = \sum_{j=0}^{k-1} c'^{(a_{3,j})}_j + w_{1,0} \end{cases} \quad B_4 : \begin{cases} w_{1,0} = \sum_{i=0}^{3}\sum_{j=0}^{k-1} c'^{(a_{i,j})}_j \\ w_{1,1} = \sum_{j=0}^{k-1} c'^{(a_{0,j})}_j + w^{(1)}_{1,0} \\ w_{1,2} = \sum_{j=0}^{k-1} c'^{(a_{1,j})}_j + w_{1,1} \\ w_{1,3} = \sum_{j=0}^{k-1} c'^{(a_{3,j})}_j + w^{(1)}_{1,0} \end{cases}$$

Where $a_{i,j}$ is the element in line i and column j of sub-matrix A.

- Then, $w_2 = w_{2,0}, ..., w_{2,m-5}$ can be computed using the following Equation:

$$w_{2,i} = \sum_{j=0}^{k-1} c'^{(f_{i,j})}_j + \sum_{j=0}^{3} w^{(f_{i,k+j})}_{1,j}$$

Where $f_{i,j}$ is the element in line i and column j of sub-matrix F (with $F = (F_1 F_2)$).

4.3.2.2 Decoding

Different methods exist for LDPC decoding and the one used in *free5GRAN* is Belief Propagation. As for polar decoding, this algorithm uses soft bits. Unlike encoding, which is based on intermediate parity check matrix H_{BG}, Belief Propagation decoding is based on the full parity check matrix H.

This section gives a global overview of the belief propagation algorithm. Deeper details about this method can be found in the literature like in [30].

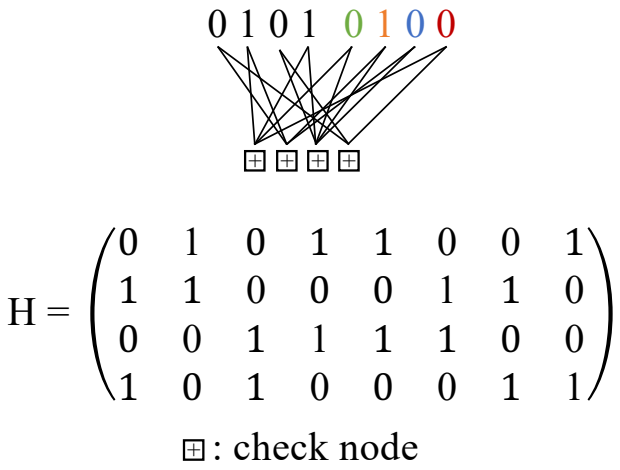


Figure 4.3: Tanner graph and associated parity check matrix

In order to make things understandable, the decoding process is explained using a toy example. We consider an example with 4 input bits and 4 parity bits with the parity check matrix H proposed in Figure 4.3. The input bits and parity bits are $c = \{0, 1, 0, 1\}$ and $w = \{0, 1, 0, 0\}$. In this example, we consider that we receive $d' = \{0, 0, 0.9, -1.0, 0.7, -0.9, 1.1, 0.8\}$. The belief propagation method describes how a soft bit value can be updated depending on other bits values. It must be done for all the bits j in the sequence. Here is the process for updating (correcting) d'_0 ($j = 0$):

1. First, the probability for bit j (noted b_j, which is the hard bit value of d'_j) to be equal to 1 is computed for all the bits in d'_j:

$$P(b_j = 1) = \frac{1}{1 + e^{2 \cdot d'_j}}$$

With our example, we obtain $\{0.5, 0.5, 0.14, 0.88, 0.19, 0.86, 0.1, 0.17\}$.

2. Then, the set of all the rows in the column j where $H_{i,j} = 1$ is computed and noted $C_j = \{i : H_{i,j} = 1\}$. It contains the check nodes to which bit j is connected. With our example ($j = 0$), $C_0 = \{i : H_{i,0} = 1\} = \{1, 3\}$.

3. Then, for each element of C_j, the set $R_i = \{j : H_{i,j} = 1\}$ is computed. R_i gives the set of the bits j involved in check node i. $R_{i\backslash j}$ is also determined as being the set of the other bits than bit j to be involved in check node i. Here, $R_{1\backslash 0} = \{1, 5, 6\}$ and $R_{3\backslash 0} = \{2, 6, 7\}$.

4. Then, the expected probability for bit j to be equal to 1 or 0 in order to satisfy check node i is computed. It is noted $r_{i,j}(1)$ (and $r_{i,j}(0)$) and is computed as follows:

$$\forall i \in C_j, r_{i,j}(1) = \frac{1}{2} - \frac{1}{2} \cdot \prod_{j \in R_{i\backslash j}} (1 - 2 \cdot P(b_j = 1))$$

and

$$\forall i \in C_j, r_{i,j}(0) = 1 - r_{i,j}(1)$$

In our example, we have:

$$r_{1,0}(1) = \frac{1}{2} - \frac{1}{2} \cdot (1 - 2 \cdot P(b_1 = 1)) \cdot (1 - 2 \cdot P(b_5 = 1)) \cdot (1 - 2 \cdot P(b_6 = 1)) = 0.5$$

and

$$r_{3,0}(1) = \frac{1}{2} - \frac{1}{2} \cdot (1 - 2 \cdot p(d'_2 < 0)) \cdot (1 - 2 \cdot p(d'_6 < 0)) \cdot (1 - 2 \cdot p(d'_7 < 0)) = 0.31$$

with

$$r_{1,0}(0) = 0.5$$

and

$$r_{3,0}(0) = 0.69$$

This means that to satisfy check node 1, bit 0 should be as likely to be 0 as 1 and that to satisfy check node 3, bit 0 should have a probability of being equal to 0 of 0.69.

5. Finally, all the information is put together in one metric, which reflects the expected probability of bit j to be 1 or 0 with regard to all the check nodes in which it is involved as well as confronting it with the actual received value.

$$q_j(1) = P(b_j = 1) \cdot \prod_{i \in C_j} r_{i,j}(1)$$

and

$$q_j(0) = (1 - P(b_j = 1)) \cdot \prod_{i \in C_j} r_{i,j}(0)$$

With our example, it gives for $j = 0$:

$$q_0(1) = P(b_0 = 1) \cdot r_{1,0}(1) \cdot r_{3,0}(1) = 0.5 \cdot 0.5 \cdot 0.31 = 0.08$$

and

$$q_0(0) = (1 - P(b_0 = 1)) \cdot r_{1,0}(0) \cdot r_{3,0}(0) = 0.5 \cdot 0.5 \cdot 0.69 = 0.17$$

6. This metric can finally be normalized to update probability of bit j to be 1 or 0:

$$P(b_j = 1) = \frac{q_j(1)}{q_j(0) + q_j(1)}$$

Which gives, for example:

$$P(b_j = 1) = \frac{0.08}{0.17 + 0.08} = 0.32$$

7. Finally, the new LLR value of bit j can be recovered:

$$d'_j = \frac{1}{2} \cdot \ln\left(\frac{1}{P(b_j = 1)} - 1\right)$$

Here:

$$d'_0 = 0.37$$

As $d'_0 > 0$, bit 0 is likely to be equal to 0, which corresponds to the initial input of the example (input bits were $\{0, 1, 0, 1\}$). Using the same procedure, soft bit 1 (d'_1) can be recovered, and the new value is $d'_1 = -0.31$: bit 1 is likely to be 1, which also corresponds to the input sequence.

The above process can be repeated for each bit j of d' until condition $H \cdot {d'}^T = 0$ is met. Finally, the input sequence c can be recovered by taking the first K elements of d'. After channel decoding, the LLR bits can be moved to hard bits using hard decision: $c_j = 1$ if $d'_j < 0$ and $c_j = 0$ otherwise.

Algorithm 7 LDPC decoding

```
1: ▷ First, recover d' (d_p) from d (d_p is then a N + 2 · Z_c long sequence)
2: d_p = push_zeros(d, 2 · Z_c)
3: while H · d_p^T ! = 0 do
4:     ▷ Compute p(d'_j < 0) for each j and put it in prob_array
5:     prob_array = compute_probabilities(d_p)
6:     for j ∈ [0, N + 2 · Z_c − 1] do
7:         ▷ Update value of bit j with Belief Propagation algorithm
8:         d_p[j] = update_value(H, prob_array, j)
9:     end for
10: end while
11: ▷ Recover c from d'
12: for j ∈ [0, K − 1] do
13:     if d_p[j] > 0 then
14:         c[k] = 0
15:     else
16:         c[k] = 1
17:     end if
18: end for
```

Algorithm 7 represents the overall procedure for LDPC correction using Belief Propagation. *update_value* is the function that actually performs the Belief Propagation process described above. In Algorithm 7, the decoding loop (line 3) is infinite, which means that if the received signal contains too many errors, the decoder will try to decode indefinitely and never succeed.

In *free5GRAN* and other implementations, a maximum number of iterations has to be defined so that decoder does not try indefinitely but stops after trying a given number of times, after which it can be considered that decoding was not successful. This maximum number of iterations is a trade-off between the decoding capacity (the highest the number of iterations, the highest decoding capacity) and the algorithm complexity. A value between 5 and 10 can be typically used in usual implementations.

4.4 Rate matching *(Transport channel processing - TS38.212 section 5.4)*

Rate matching (or rate recovering in decoding direction) is a process that enables to adapt the size of the channel coding output (noted N) to the size of the allocated channel (noted E). If $N > E$, there are too many bits to transmit compared to the allocation size, and some bits have to be punctured or shortened. Otherwise, if $E > N$, there is too much space in the allocation compared to the number of bits to be transmitted. In that case, the bits are repeated. The rate matching strategy depends on the algorithm used for channel coding. The rate matching algorithm described in the standard

can be directly implemented. Nevertheless, it is essential to understand the algorithm's purpose and concept.

4.4.1 Rate matching for polar codes *(TS38.212 section 5.4.1)*

The rate matching process for polar codes follows the standard (TS38.212 section 5.4.1 [11]). It is detailed in appendix A.9.1.

4.4.2 Rate matching for LDPC *(TS38.212 section 5.4.2)*

The rate matching for LDPC follows the same purpose as for Polar Codes (i.e., adapting the transport block size to the available size on the grid) but is done in a completely different way. The main difference is that rate matching for LDPC will remove a significant part of the transport block payload, enabling the transfer of the fewer possible data. Selecting the bits to transfer is based on a parameter called the Redundancy Version (RV_{ID}).

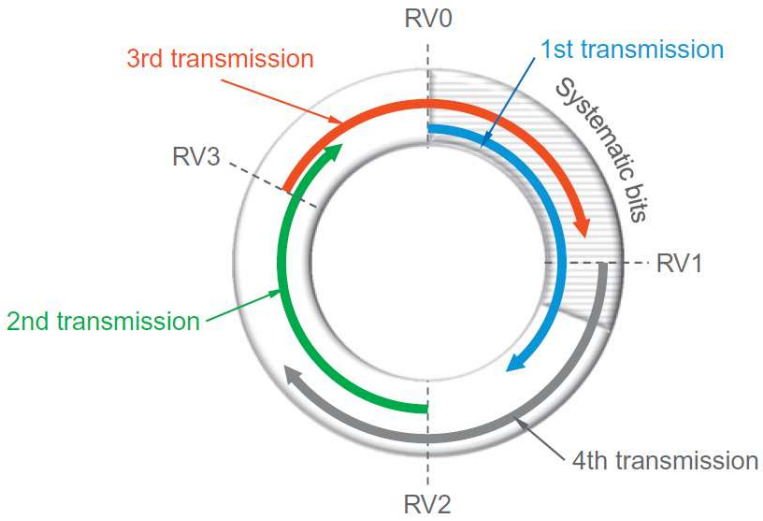


Figure 4.4: HARQ buffer transmission, from [4], Figure 13.3

This selection is part of the upper layer HARQ process: the transport block is first transmitted with $RV_{ID} = 0$. If the receiver cannot recover the entire message at the LDPC level, then it notifies the transmitter to send the same transport block with $RV_{ID} = 1$ (which selects other bits to be transmitted at the rate matching level). Then, the receiver can concatenate the data received with $RV_{ID} = 0$ and $RV_{ID} = 1$ and try again to recover the missing data using LDPC decoding. This process can be done 4 times until $RV_{ID} = 3$, and if the receiver cannot even decode after the 4 transmissions, then transport block transmission is reported as failed to the upper layers. Figure 4.4 represents the part of the buffer that is transmitted for each RV_{ID}. The systematic bits are the LDPC input bits, and the other ones are the parity check bits. This process enables the reduction of the transmission size. The role of rate matching for LDPC is to select which bits must be transmitted depending on the HARQ state. This process is split into bits selection and interleaving, performed for each code block, i.e., before code blocks concatenation. Details about bits selection and interleaving can be found in appendix A.9.2.

4.5 Scrambling *(Physical channel processing - TS38.211 6.3 and 7.3)*

Scrambling consists in mixing an input sequence with a pseudo-random sequence. Its purpose is to avoid confusion at the different levels of the PHY layer. The outputs of the scrambling of a single input sequence with two different pseudo-random sequences will be highly different. In 5G, the pseudo-random sequences are Gold sequences defined in TS38.211 section 5.2.1 [21] and depend on an input parameter noted c_{init}.

One primary use case is inter-cell confusion. The physical channels are scrambled with a pseudo-random sequence generated with the cell's PCI. Thus, the UEs can only decode the transmissions sent by the cell to which they are attached. UEs located at the cell's edges could receive transmissions from other cells but might not be able to decode them as the PCI is different.

Scrambling a sequence a into a sequence a' is done using the following equation:

$$a'_i = (a_i + s_i) \pmod 2$$

Where $i \in [0, A-1]$, A is a sequence size, and s is the scrambling pseudo-random sequence. De-scrambling a sequence a' into a sequence a is done by reversing a and a' sequences in the equation.

4.6 Modulation *(Physical channel processing - TS38.211 section 5.1)*

The modulation is the process by which a sequence of M_{bit} bits is transformed into a sequence of M_{symb} IQ samples. Different modulation schemes can be used depending on the channel's quality and the robustness requirement: the worst the channel is, or the highest the robustness requirement is, the lowest the modulation order is. The modulation order m (also noted Q_m in the standard) gives the number of bits that are encoded per IQ sample: $m = M_{bit}/M_{symb}$.

Scheme	Order
BPSK	1
$\frac{\pi}{2}$-BPSK	1
QPSK	2
16-QAM	4
64-QAM	6
256-QAM	8

Table 4.3: Modulation schemes

4.6.1 Modulation mapping

The modulation mapping process is defined in TS38.211 section 5.1. It consists in taking sequences of m bits and assigning them an output IQ sample following equations provided by the standard. It is equivalent to assigning a sequence of bits to one point in the complex plane. Figure 4.5 represents the mapping between bits sub-sequences and IQ sample for 16-QAM modulation. The set of all the possible modulation points is called the constellation. For example, the sub-sequence $[0, 1, 1, 0]$ is mapped to complex $0.94 - j \cdot 0.31$.

4.6.2 Modulation de-mapping

The modulation de-mapping enables the receiver to recover a sequence of M_{bit} bits from a sequence of M_{symb} IQ samples. This process is done sample per sample. Given that channel is never ideal, the received sample never corresponds precisely to the sample that the transmitter has sent, i.e., the position of the sample varies in the complex plane. Coming back to example in the last section (4.6.1), if the sent sample is $0.94 - j \cdot 0.31$, then the received sample might be $1.1 - j \cdot 0.45$. The better the channel conditions are, the closer to the sent sample the received one is. In these conditions, modulation de-mapping tries to recover the sent bits based on the analysis of the received IQ sample. Two different methods exist.

4.6.2.1 Hard modulation de-mapping

The first one, hard modulation de-mapping, is the most basic one, which is efficient in terms of complexity but not in terms of accuracy. It approximates the received IQ sample to the closest possible constellation point. Figure 4.6a illustrates hard modulation de-mapping for QPSK modulation. If the

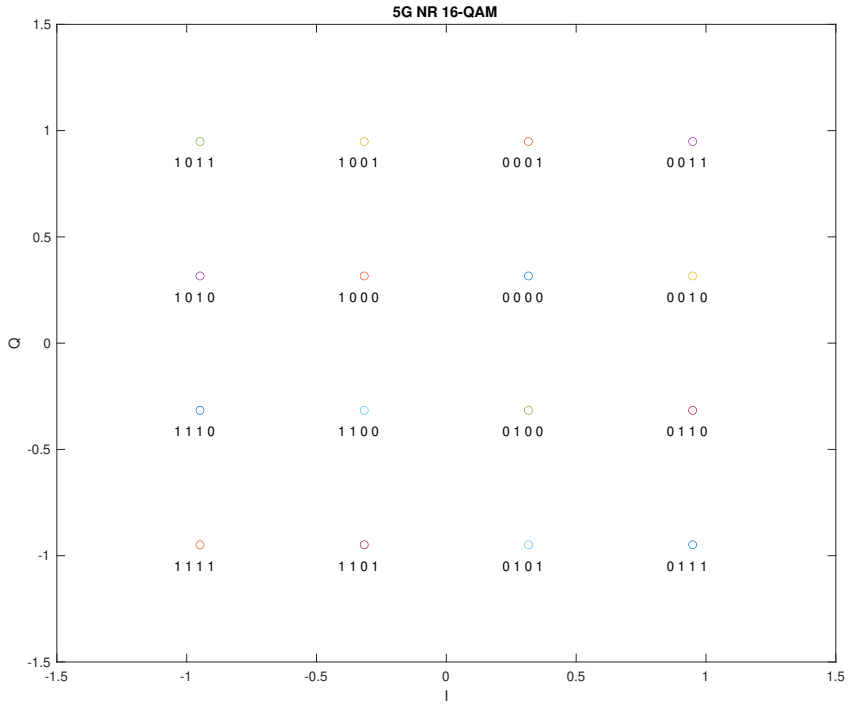
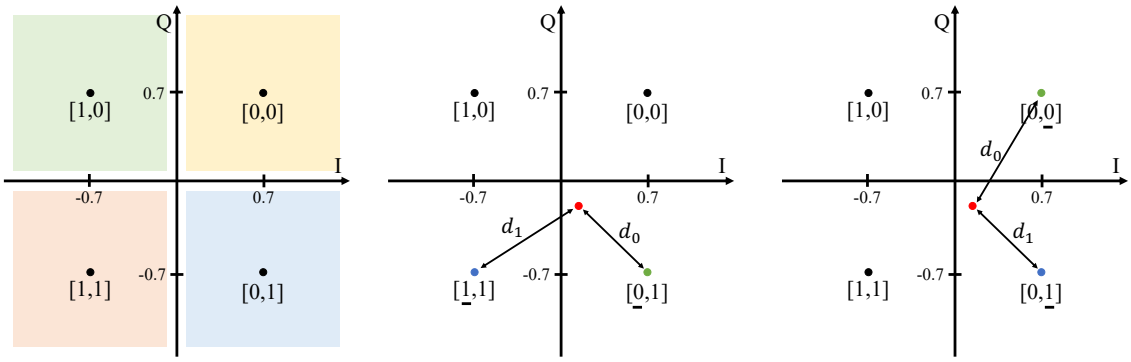


Figure 4.5: 16-QAM complex plane mapping

received sample is in the blue area, it will be approximated to the $0.7 - j \cdot 0.7$ QPSK constellation point. The modulation de-mapping output will thus be $[0, 1]$.

4.6.2.2 Soft modulation de-mapping

Even if hard modulation de-mapping is very efficient, it is not an operational algorithm as channel conditions will highly impact its performances. The main issue is that hard modulation de-mapping does not transfer to the channel decoder how close the received sample was to the constellation point, so the channel decoder cannot estimate how reliable the values of the received bits are. For example, considering Figure 4.6a, the hard modulation de-mapping does not make a difference between a sample received in position $0.72 - j \cdot 0.69$, which is close to the constellation point (and so the values of the bits are reliable and not likely to be false) and a sample received at position $0.1 - j \cdot 0.15$. They will be both assigned the bit sequence $[0, 1]$. To solve this problem, soft modulation de-mapping is introduced as a way not to allocate a fixed value to each bit but a value that represents the probability for the bit



(a) Hard de-mapping (b) Soft de-mapping: bit 0 (c) Soft de-mapping: bit 1

Figure 4.6: QPSK modulation de-mapping

to be equal to 1 or 0. In such conditions, the channel decoder can rely on bits with a high probability to be equal to 1 or 0 to correct errors of bits with the lowest probability. For soft modulation de-mapping, bit values are called LLR, and the modulation de-mapping is not done sample per sample (as for hard-demodulation) but bit per bit. A good overview of the LLR is given in [31]. The LLR value of bit i for received sample r is defined as follows:

$$llr_i = \ln\left(\frac{P(b_i = 0|r)}{P(b_i = 1|r)}\right) \tag{4.1}$$

Where $P(b_i = 0|r)$ and $P(b_i = 1|r)$ denote the probabilities for bit i to be equal to 0 and 1. According to [31], this leads to:

$$llr_i = \ln\left(\frac{\sum_{k/k_i=0} e^{\frac{-\|r-c(k)\|^2}{2\cdot\sigma^2}}}{\sum_{k/k_i=1} e^{\frac{-\|r-c(k)\|^2}{2\cdot\sigma^2}}}\right) \tag{4.2}$$

Where σ is the estimated noise variance. The sum is made across all the modulation points k for which the value of bit i (k_i) is 0 or 1 and $c(k)$ gives the coordinates of point k in the complex plane (and $\|r - c(k)\|^2$ is the distance from received sample r and the point k).

Given that computing the LLR for every bit can be highly complex, some approximations must be made. A commonly used approximation is to consider that the most important element of the sum in Equation 4.2 corresponds to the closest constellation point whose value is 1 or 0. The sum can therefore be approximated by only keeping the element that corresponds to the closest modulation point. The LLR value can therefore be approximated such as:

$$llr_i \approx \ln\left(\frac{e^{\frac{-\|r-c(k_i^0)\|^2}{2\cdot\sigma^2}}}{e^{\frac{-\|r-c(k_i^1)\|^2}{2\cdot\sigma^2}}}\right) \approx \frac{1}{2\cdot\sigma^2}\cdot(\|r - c(k_i^1)\|^2 - \|r - c(k_i^0)\|^2)$$

Where k_i^0 and k_i^1 are the closest constellation points to r whose values for bit i is 0 and 1.

Moreover, a second approximation can be done by not considering the noise ($\sigma = 1$). Finally, the LLR approximation for bit i is:

$$llr_i \approx \frac{1}{2}\cdot(\|r - c(k_1)\|^2 - \|r - c(k_0)\|^2) \tag{4.3}$$

The LLR value for bit i is therefore the difference between the distance from r to the closest constellation point which value for bit i is 1 (noted d_1) and the distance from r to the closest constellation point which value for bit i is 0 (noted d_0).

Let us consider an example where the modulation scheme is QPSK, and the received sample is $0.1 - j\cdot 0.15$ (red point in Figures 4.6). As the modulation is QPSK, there are two bits ($[llr_0, llr_1]$) to recover from each IQ sample:

- llr_0 (Figure 4.6b): The two closest points which have 0 and 1 values for llr_0 are $-0.7 - j\cdot 0.7$ (value 1 for bit 0) and $0.7 - j\cdot 0.7$ (value 0 for bit 0). The distance from those two points are $d_1 = 0.97$ and $d_0 = 0.81$. The LLR value for the first bit will be $llr_0 = d_1 - d_0 = 0.16$.
- llr_1 (Figure 4.6c): The two closest points which have 0 and 1 values for llr_1 are $0.7 - j\cdot 0.7$ (value 1 for bit 1) and $0.7 + j\cdot 0.7$ (value 0 for bit 1). The distance from those two points are $d_1 = 0.81$ and $d_0 = 1.04$. The LLR value for the second bit will be $llr_1 = d_1 - d_0 = -0.23$.

In this example, the output LLRs for IQ sample $0.1 - j\cdot 0.15$ are $[0.16, -0.23]$. It would have been $[1.4, -1.37]$ if the received sample had been $0.72 - j\cdot 0.69$.

Finally, following Equation 4.1 and given that $p(b_i = 0|r) + p(b_i = 1|r) = 1$, the probability for each bit to be equal to 1 or 0 is given by Equation 4.4.

$$p(b_i = 1|r) = \frac{1}{1 + e^{2\cdot b_i}}, \quad p(b_i = 0|r) = 1 - p(b_i = 1|r) \tag{4.4}$$

Received IQ sample r	**Bit**	**Value**	$p(b_i = 1\|r)$	$p(b_i = 0\|r)$
$0.1 - j \cdot 0.15$	b_0	0.16	0.42	0.58
	b_1	-0.23	0.61	0.39
$0.72 - j \cdot 0.69$	b_0	1.4	0.05	0.95
	b_1	-1.37	0.93	0.07

Table 4.4: LLR bits associated probabilities

In our example, the probability for the different bits to be equal to 1 or 0 is given by Table 4.4.

It can be observed that the probabilities associated with IQ sample $0.1 - j \cdot 0.15$ are much closer to 0.5 than the probabilities associated with IQ sample $0.72 - j \cdot 0.69$. As 0.5 is the non-determined value, where the bit is as likely to be 0 or 1, the channel decoder considers that the bits whose associated probability is the furthest from 0.5 are the most reliable. In this case, it can consider that LLR bits for IQ sample $0.72 - j \cdot 0.69$ are much more reliable than LLR bits for IQ sample $0.1 - j \cdot 0.15$. Using hard modulation de-mapping, those two samples would have been given the same value, and the reliability information would not have been propagated back to the decoder. Soft modulation de-mapping is a much more complex process, but it enables a much finer channel decoding and thus increases channel quality.

The code snippet in appendix A.10 is an example from *free5GRAN* for 16-QAM soft modulation de-mapping. The code is specific for each modulation scheme for performance reasons, but the process is the same.

4.7 Channel mapping / de-mapping *(Physical channel processing - TS38.211 6.3 and 7.3)*

The purpose of channel mapping is to map different signals and channels onto a single OFDM grid. This is used both for uplink and downlink transmissions at the gNodeB and UE side. Channel mapping is the function performed by the transmitter (gNodeB in downlink direction and UE in uplink direction), and channel de-mapping is performed at the receiver side.

4.7.1 Transmitter side

On the transmitter side, the channel mapper receives a list of channels and signals together with their position in the slot OFDM grid. The mapper should place all the IQ samples at the given position into the grid. The mapper upper-layer input is the list of channels IQ samples and positions.

4.7.2 Receiver side

On the receiver side, the channel de-mapper receives a list of channels and signals that must be extracted from the grid. The de-mapper must put all the IQ samples from the OFDM grid in the corresponding signal/channel buffer, based on the list of positions. The de-mapper input is the list of channels' positions, and the output is the channels' IQ samples.

The code snippet in appendix A.12 shows the *free5GRAN* implementation of channel de-mapping.

Figure 4.7 represents the channel mapping and de-mapping process. On the left (channel mapping), the input channel IQ samples and their positions are placed in the OFDM grid. On the right (channel de-mapping), the OFDM grid and the channel positions are used to extract the IQ samples from the grid.

4.8 Channel estimation and equalization *(Signal processing)*

Upon reception of a physical channel (in the receiving direction, i.e., after channel de-mapping), gNodeB and UEs have to estimate and equalize the channel before performing upper layers decoding. This process aims to mitigate the effects of noise and interference over the signal. It relies on the

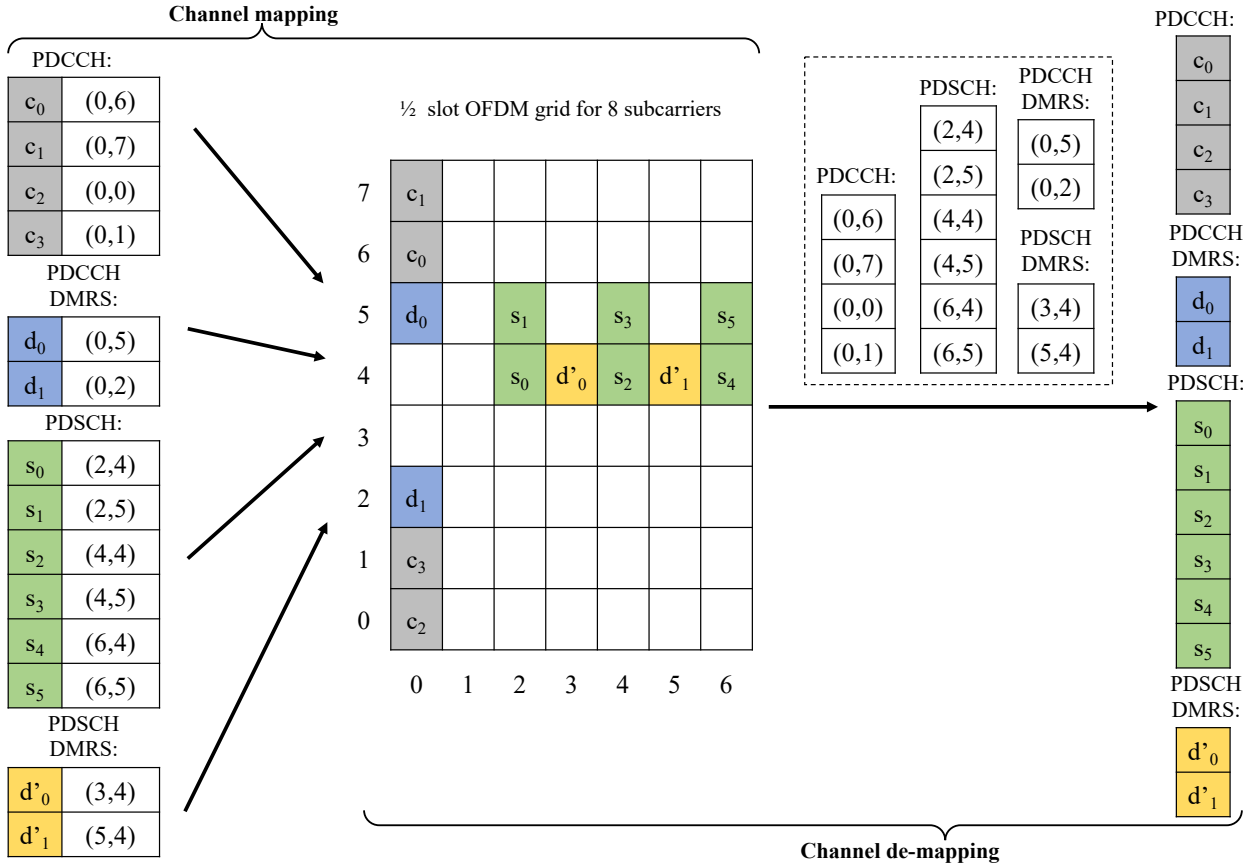
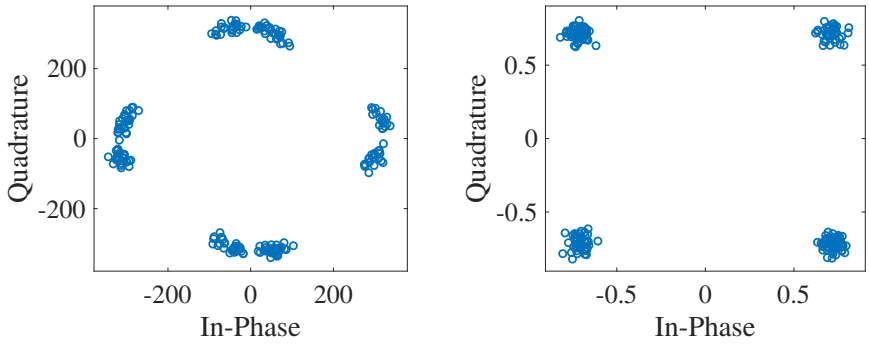


Figure 4.7: Channel mapping and de-mapping

DMRS signal, sent together with the physical channels, which the receiver uses as a reference. Figure 4.8a and 4.8b represent a constellation before and after equalization for a QPSK modulation.

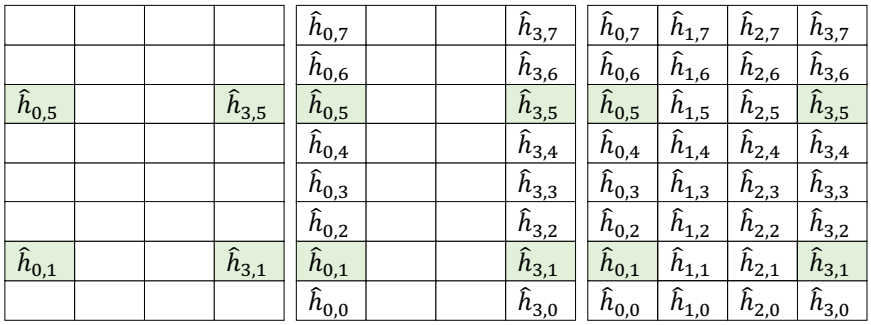
4.8.1 Channel estimation

The first step is channel estimation, which consists of computing a channel coefficient for each position in the OFDM grid. The coefficient reflects the channel perturbations. Different methods exist for channel estimation, and the one used in *free5GRAN* is zero-forcing, which is the most simple and effective. Robert Lucky introduced it in 1965 [32]. It consists in computing the channel coefficients by reversing the channel at the DMRS positions and propagating them over the whole OFDM grid by performing linear interpolation. Figure 4.9 represents the overall procedure for zero-forcing channel estimation.



(a) Before (b) After

Figure 4.8: IQ samples constellation before and after equalization



(a) Compute coefficients for DMRS positions (b) Frequency-domain interpolation (c) Time-domain interpolation

Figure 4.9: Channel estimation process for 4 symbols by 8 subcarriers OFDM grid (DMRS REs are in green)

The main steps are:

1. Compute the DMRS sequence that the transmitter has sent.
2. Compute the channel coefficients for the DMRS positions:

$$\hat{h}_{i,j} = \frac{y_{i,j}}{x_{i,j}}$$

   where $\hat{h}_{i,j}$ is the estimated channel coefficient at OFDM symbol i and subcarrier j, $x_{i,j}$ is the complex representation of the expected DMRS IQ sample previously computed and $y_{i,j}$ is the complex representation of the actual received IQ sample. This computation is made for all the (i, j) positions where DMRS has been sent.
3. Finally, those coefficients can be propagated over the whole OFDM grid (in positions where there are no DMRS) by interpolating the computed coefficients in time and frequency directions.

The output of channel estimation is an OFDM grid that contains, for each RE, the associated channel coefficient $\hat{h}_{i,j}$, which represents the perturbations implied by interference and noise.

The code snippet in appendix A.11 is the implementation of channel estimation in *free5GRAN*.

4.8.2 Channel equalization

Channel equalization consists in reversing interference and noise effects using the estimated channel coefficients. Equalized IQ samples $\hat{x}_{i,j}$ can be computed using the following Equation:

$$\hat{x}_{i,j} = \frac{y_{i,j}}{\hat{h}_{i,j}}$$

This must be done for all the (i, j) positions of the physical channel being decoded. Once channel equalization has been performed, modulation de-mapping and upper layer processing can be done.

4.9 OFDM modulation *(Signal processing - TS38.211 section 5.3.1)*

When the different channels have been placed into a single OFDM grid, the frequency-domain signal must be transformed to generate the corresponding time-domain signal. This process is called OFDM modulation and is a two-step function. First, the frequency domain symbols are passed through an iFFT and then, the cyclic prefix is added to each time domain symbol. All the time-domain symbols can finally be concatenated. The OFDM modulation is defined in TS38.211 section 5.3.1 [21].

Before performing those two steps, the process should be initialized by computing the number of iFFT elements and the cyclic prefix lengths for each symbol of the slot:

- Fast Fourier Transform (FFT) size: the number of FFT elements is given by the SCS and the number of RB in the band. For example, if the number of RB in the band is 50 (i.e. $12 \cdot 50 = 600$ subcarriers) then the FFT size should be the lowest power of two bigger than 600, i.e. 1024. The 600 subcarriers might be centered in the 1024 FFT elements and the other 424 elements should be filled with zeros. Please note that the RF device should have been previously initialized accordingly, by setting the sampling rate to $1024 \times 30 \cdot 10^3 = 30.72$MHz, in case the SCS is 30kHz. That information is computed while initializing the RF device and physical layer.
- Cyclic prefix length: the cyclic prefix length depends on the FFT size, SCS and the symbol position in slot. As specified in TS 38.211 section 5.3.1 [21], it is given by the following equation (we consider the case of normal cyclic prefix length):

$$N_{CP}(l) = \begin{cases} 144 \cdot \kappa \cdot 2^{-\mu} & l \notin \{0, 7 \cdot 2^{\mu}\} \\ 144 \cdot \kappa \cdot 2^{-\mu} + 16 \cdot \kappa & l \in \{0, 7 \cdot 2^{\mu}\} \end{cases}$$

Where $N_{CP}(l)$ is the cyclic prefix length of symbol l, μ is the numerology index (0 for a SCS of 15kHz and 1 for 30kHz for example) and $\kappa = FFT_{size}/1024$.

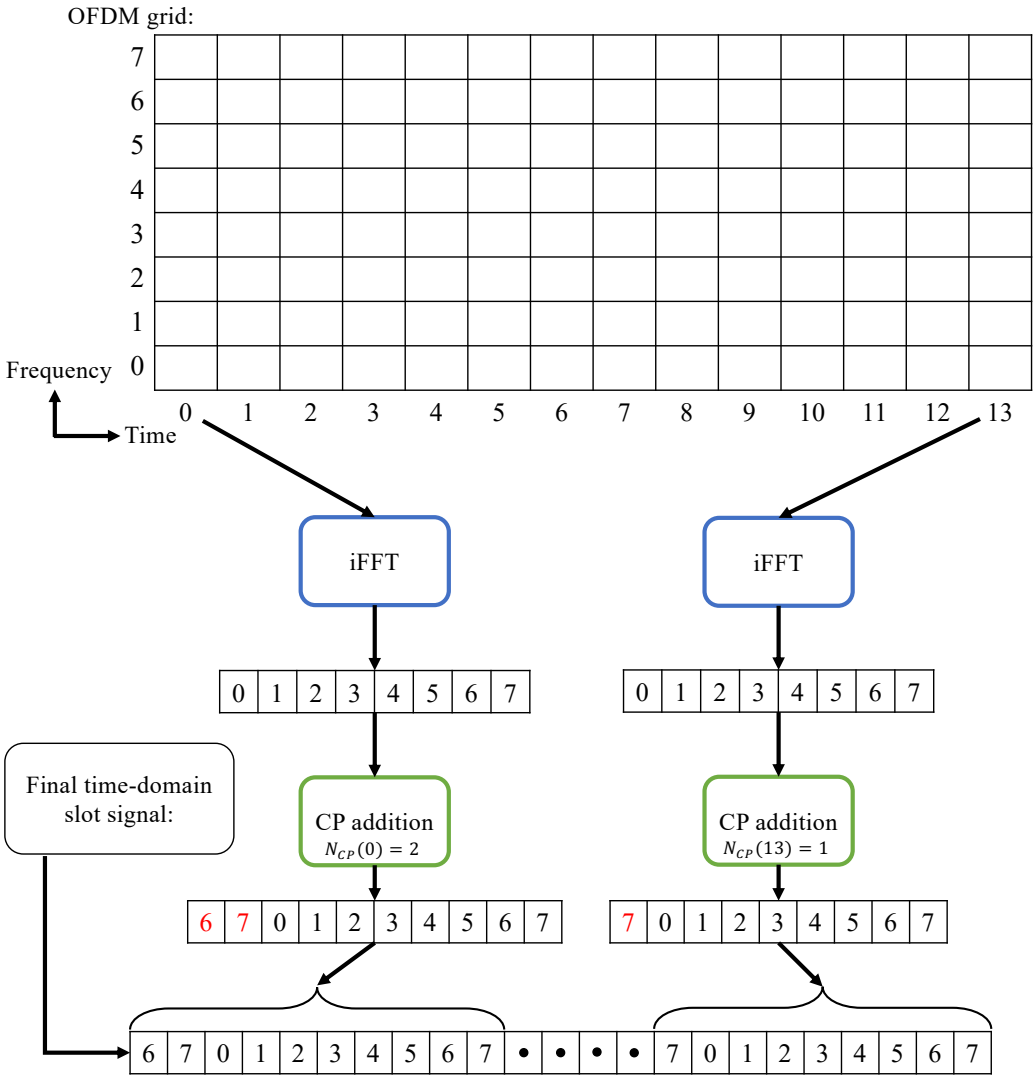


Figure 4.10: OFDM modulation on the transmitter's side

Figure 4.10 represents the OFDM modulation process. For each symbol in slot:

- Perform the iFFT on all the symbol's subcarriers. The output is the time domain symbol, without cyclic prefix. In Figure 4.10, this is represented with blue iFFT blocks for symbols 0 and 13.

- Perform the cyclic prefix addition by copying the $N_{CP}(l)$ last samples and putting them in front of the iFFT output (where l is the symbol index within the slot). For example, in Figure 4.10, the cyclic prefix addition is the green block, and the cyclic prefix size is 2 for symbol 0 and 1 for symbol 13.
- Put the obtained time-domain symbol signal inside the time domain slot signal. For example, in Figure 4.10, the final time-domain signal is represented in the lowest array.

The OFDM demodulation is the reverse procedure:

- Cyclic prefix addition is replaced by cyclic prefix deletion: the $N_{CP}(l)$ first samples of each symbols are deleted.
- iFFT is replaced by FFT.
- The slot OFDM grid can be recovered when all the symbols FFT have been perfomed

An example code snippet from *free5GRAN* for OFDM demodulation can be found in appendix A.13.

4.10 Conclusion

In this chapter, we introduced the different physical layer functions whose input, output, and behavior are defined in the standard but whose implementation is not precisely defined. We explained the purpose of the different functions and proposed concrete algorithms that work in our environment inside a Faraday cage.

The algorithms where reside the highest challenges are channel decoding, modulation de-mapping and channel estimation. Belief propagation and successive cancellation are used for LDPC and polar decoding. Modulation de-mapping is approximated by only considering the two closest constellation points. The zero-forcing method is used for channel estimation.

All the other functions are still crucial, but a smaller number of possible algorithms exist, and performances mostly reside in algorithmic optimization. Those algorithms are implemented in *free5GRAN*. From now on, all the minimal physical layer procedures and associated algorithms have been detailed.

Chapter 5

Software implementation and architecture

Contribution

This chapter exposes the *free5GRAN* project's structure and software architecture. The structure is made so one can easily modify functions and algorithms in the *free5GRAN* project. This chapter also overviews a possible physical layer architecture, which is a critical aspect of the PHY implementation. The software architecture is modular so that it can be used for different services and uses multi-threading so that *free5GRAN* can be easily deployed in dis-aggregated networks.

5.1 Introduction

This chapter introduces the project structure and software architecture that enables the implementation of the *free5GRAN* PHY layer. Indeed, a PHY layer is a system that includes different procedures and algorithms. While building a system, understanding how to read the standards and implement each component is half of the problem, which has received attention in the previous chapters. Understanding how to create an operational code structure that can be incremented components after components is the other half and is the heart of this chapter. Different architectures can be used. We expose the *free5GRAN* project structure and software design to give a concrete example, but especially to show the problems that the design has to answer.

Two main challenges must be solved. The first one is that telecommunication systems evolve very quickly, and the implementation is not a one-step task but rather an iterative process where each component can be updated independently. Therefore, the project must be structured so it can be easily incremented over the standard versions and global evolutions. The second main challenge is that the PHY layer is highly constrained. Indeed, it is a near real-time application with high-performance requirements, but some functions have high algorithmic complexity. The software architecture must deal with those two constraints.

The proposed software design is not monolithic but extensively uses multi-threading for two reasons. The first one is that the 5G standard introduces different base-station architectures, among which dis-aggregated RAN receives a lot of attention. Therefore, even if the current PHY layer implementation does not support functional split, the software architecture has been thought so that it is easy to derive the current architecture to implement RAN splitting option $7.2x$ (which is the option selected by the O-RAN alliance). The second reason is related to the high heterogeneity of the services that can be provided over the 5G system. Indeed, as some services might have strong requirements in terms of reliability and isolation, implementing a modular architecture enables to dedicate resources to one or another service. Therefore, the current architecture can be easily derived to allocate threads to services to strengthen the reliability and control over the PHY layer processing.

The first section of this chapter focuses on the project's structure, whereas the second one investigates the software architecture and design.

5.2 Project structure

free5GRAN architecture is split into two parts. The first part is the library which contains common functions, and the second part is the implementation which contains custom code for the transmitter and the receiver. *free5GRAN* code and documentation are available online (`https://github.com/free5G/free5GRAN`). Figure 5.1 represents *free5GRAN* structure.

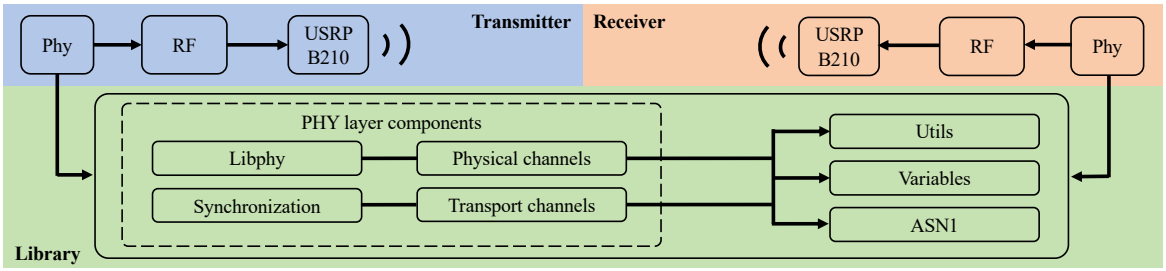


Figure 5.1: free5GRAN architecture overview

5.2.1 Library

The library is a set of static functions and variables commonly used by both the UE and gNodeB. All the functions have been designed to be used independently. It is split into four parts, which are *phy*, *utils*, *variables* and *asn1c*.

5.2.1.1 phy

The *phy* library contains the code for PHY layer processing. It is split into four parts which represent different 5G PHY components. Those parts are *libphy* for signal processing functions, *physical_channel* for physical channels processing, *transport_channel* for transport channels processing and *synchronization* for time and frequency synchronization procedures. The main functions of the different libraries are:

- *libphy*: channel estimation, channel mapping/de-mapping and OFDM modulation.
- *physical_channel*: PBCH, PDCCH, PDSCH and PUSCH encoding and decoding, defined in TS38.211 [21].
- *transport_channel*: BCH, DCI, DL-SCH and UL-SCH encoding and decoding, and all related sub-functions, defined in TS 38.212 [11].
- *synchronization*: PSS, SSS and PRACH detection for time synchronization.

5.2.1.2 utils

It contains functions that are called at different levels of the code. It includes:

- Synchronization sequences generation: generates PSS, SSS and PRACH sequences for receiver time synchronization, as defined in TS 38.211 section 7.4.2 [21].
- DMRS sequences generation: generates pilot sequences for channel estimation, as defined in TS 38.211 [21].
- Pseudo random sequence generation: generates the pseudo random sequence defined in TS 38.211 section 5.2 [21] based on an initialization value c_{init}.

5.2.1.3 variables

It is a set of variables and data structures definitions used by both the library and implementation. The most important variables are:

- Global variables defined in the standard.
- Polar coding matrices: G_n matrices, $n \in [5, 10]$, for polar coding (defined in TS 38.212 section 5.3.1.2 - [11]) and corresponding inverse matrices for polar decoding.
- LDPC base graphs, defined in TS 38.212 section 5.3.2 [11].

5.2.1.4 asn1c

It contains the code and structures for encoding and decoding RRC messages. It uses the ASN1C library [33]. In the current implementation, it is used by the receiver to decode SIB1 data. The library can currently parse and encode all the RRC PDU messages and types defined in TS 38.331 section 6.2 [19].

5.2.2 Implementation

The implementation part provides code dedicated to the receiver. It uses objects that store different processing information and exposes methods for implementing receiver functionalities. It is split into two main classes that are *rf* and *phy*.

5.2.2.1 rf

This class represents the RF device. It stores information about RF device configuration and current state and exposes methods for receiving and transmitting the signal. It implements an interface so that different RF devices can be supported through the same methods calls. In the current version, *free5GRAN* supports USRP B210 and uses USRP UHD library [34] for device exchanges.

5.2.2.2 phy

This class represents the PHY layer. It stores different cell's and PHY layer status information and implements the methods used by the gNodeB and the UE for signal transmission and reception.

5.3 Software architecture and design

This section introduces the software components defined to implement the different functions of the PHY layer.

5.3.1 Data structures

One critical aspect of the implementation of the PHY layer is how signals, channels, and other information are shared from one component to another. This section introduces the central data structures used at both the gNodeB and UE sides.

5.3.1.1 Channel buffer

The first data structure is a structure that carries a channel and its metadata. This structure is called *channelElement* and is described in Table 5.1. It is used for communicating data between the different threads.

The data field is the main one and contains the IQ samples or bits, depending on the case where this structure is used. All the other fields are metadata. It is used for placing the channel in the OFDM grid (fields position, SFN, slot_id and periodicity) and for building the DCI payload at the gNodeB's side. The RNTI field is used to identify a UE and the configuration field contains the transmission configuration to be used for processing like modulation scheme and code rate.

Parameter	Description
data	Vector of bits or complex IQ samples.
position	Where to map the signal / channel into the slot OFDM grid.
sfn	SFN where signal / channel should be mapped
slot_id	slot where signal / channel should be mapped
periodicity	signal / channel periodicity in slots. If non-periodic signal / channel, this is set to 0. For example, if PSS and SSS are sent every frame (10ms) and SCS is 30kHz, then their periodicity is 20 slots.
RNTI	UE's identifier (if applicable)
Configuration	Information about resources allocation (if applicable)

Table 5.1: *channelElement* data structure

Parameter	Description
samples	Vector of slot complex IQ samples.
sfn	Slot SFN
slot_id	Slot ID

Table 5.2: *slotElement* data structure

5.3.1.2 Slot buffer

The second main data structure defined is used to transfer slots between components. The structure is called *slotElement* and is described in Table 5.2.

The first field, called samples, contains the IQ samples of a slot, and the SFN and slot_id fields give the network's time associated with the slot.

5.3.2 gNodeB

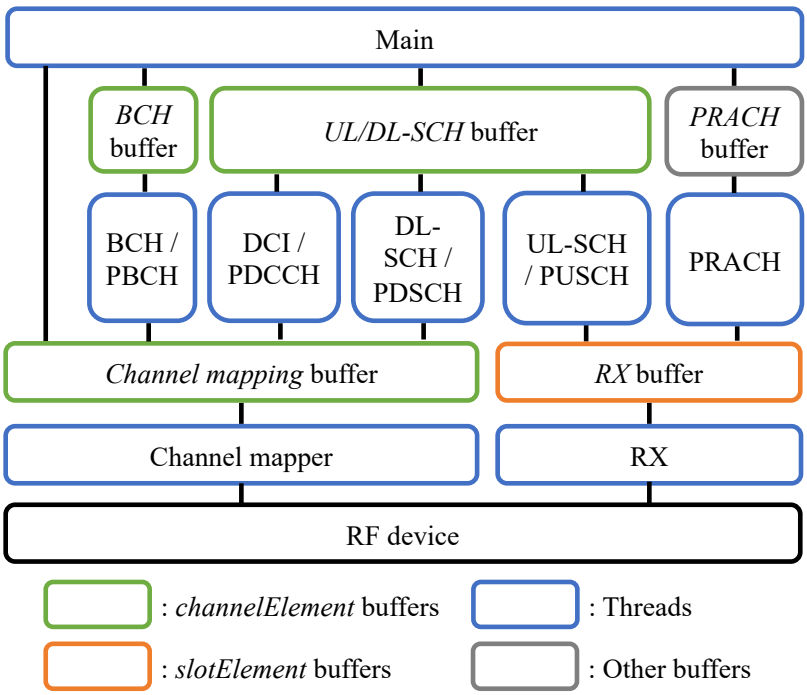


Figure 5.2: gNodeB's structure

Figure 5.2 represents the global software architecture of the gNodeB. The blue components are threads, and the green are variables and buffers used to communicate between the threads.

The different components are:

- Threads:
  - The main thread: it is responsible for initializing all the components and threads. In the future, the main thread will also implement the PHY configuration API and the MAC-to-PHY API.
  - Channel mapper thread: it is a thread that continuously maps the different signals and channels to be sent on the OFDM grid and performs OFDM modulation. This thread is designed to support all the possible signals and channels. It operates slot by slot.
  - One thread for BCH and PBCH encoding.
  - One thread for DCI / PDCCH payload generation and encoding. It gets the resource allocation from the *UL/DL-SCH* buffer and the corresponding RNTI and builds the DCI payload, performs DCI and PDCCH encoding and computes the IQ samples positions in the OFDM grid. The encoded channel and DMRS, together with their position, are pushed to the *Channel mapping* buffer.
  - A DL-SCH / PDSCH thread which performs DL-SCH and PDSCH encoding of the transport block and computes the position in the OFDM grid. The encoded channel and DMRS, together with their position, is pushed to the *Channel mapping* buffer.
  - A RX thread which receives slots from the RF device and pushes them into the *RX* buffer.
  - A PUSCH extraction and decoding thread which extracts the uplink resource allocation given by the *UL/DL-SCH* buffer. It waits grant's time has come and then extracts the corresponding IQ samples, performs PUSCH/UL-SCH decoding and finally pushes the transport block to the *UL/DL-SCH* buffer.
  - A PRACH thread which receives PRACH transmissions. It pushes the information to the main thread using the PRACH buffer.
- Buffers:
  - A *channelElement* buffer for channel mapping (called *Channel mapping* buffer in Figure 5.2) which is used by the different threads to push data to the channel mapper thread.
  - A *channelElement* buffer used by the Main thread to push BCH payloads to be transmitted (called *BCH* buffer in Figure 5.2).
  - A *channelElement* buffer used by the Main thread to push downlink transport blocks to be transmitted and uplink allocations to be notified (called *UL/DL-SCH* buffer in Figure 5.2).
  - A *slotElement* buffer called *RX* buffer.

5.3.2.1 Channel mapper

Algorithm 8 details the channel mapping thread's main processing steps (where the input parameter *mappingChElemsBuff* is the *Channel mapping* buffer).

Line 6, the first transmission time is initialized to 0.5ms in the future, which is the estimated time of the first iteration.

For each iteration, the first step is to wait (line 9) for the next slot transmission time to be less than 0.5ms in the future. This is the first flow control step: channel mapping must remain consistent with the current network slot. Without control, the channel mapping could be done very quickly and take a huge advance compared to the current network slot and SFN, and the channel mapper might become inconsistent. Therefore, it is required to wait to be close to the transmission time of the slot for performing channel mapping and OFDM modulation. In *free5GRAN*, we wait to be less than 0.5ms (which corresponds to one slot when the SCS is 30kHz) before the transmission time of the slot.

Algorithm 8 gNodeB channel mapper

```
▷ mappingChElemsBuff is the bufferElement Channel mapping buffer
function channelMapper(mappingChElemsBuff)
    ▷ Initialize sfn and slot ID
    sfn = 0
    slot_id = 0
    ▷ Initialize next slot starting timestamp
    nextSlotTime = getCurrentTime + 0.5ms
    while true do
        ▷ Wait next slot is less than 0.5ms in the future
        while nextSlotTime − currentTime > 0.5ms do
            sleep(5us)
        end while
        toSend = ∅
        ▷ Put in toSend all the buffer elements with current SFN and slot
        for bufElem ∈ mappingChElemsBuff do
            if bufElem.sfn = sfn and bufElem.slot_id = slot_id then
                toSend.push(bufElem)
                if bufElem.periodicity = 0 then
                    ▷ If non-periodic element, remove it from buffer
                    mappingChElemsBuff.remove(bufElem)
                else
                    ▷ If periodic element, setup next element occurence
                    [bufElem.sfn, bufElem.slot_id] = next_occ(sfn, slot_id, bufElem.periodicity)
                end if
            end if
        end for
        grid = init_empty_grid()
        ▷ Place all the elements in one grid (channel mapping)
        for elem ∈ toSend do
            place_in_grid(elem, grid)
        end for
        ▷ Initialize an empty time domain slot
        tdSlot = []
        ▷ For each symbol in slot
        for symbol ∈ grid do
            ▷ Perform iFFT, add cyclic prefix and push back symbol into time domain slot
            tdSymbol = ifft(symbol)
            tdSymbolCP = add_cp(tdSymbol)
            tdSlot+ = td_symbol_cp
        end for
        ▷ Transmit signal over the air
        tx_send(tdSlot, nextSlotTime)
        ▷ Update next slot time, sfn and slot ID
        next_slot_time+ = slotDuration
        iterate_sfn_slots(sfn, slot_id)
    end while
end function
```

Line 22, for periodic channels, the channel mapper will set the SFN and slot ID of the element to the next occurrence. For example, if PSS is sent every two frames on slot ID 3, then the main thread will push an initial occurrence into the *Channel mapping* buffer with SFN 0 and slot ID 3 and periodicity of 40 (if we consider a case where there are 20 slots per frame). The channel mapper will not delete the PSS from the buffer as its periodicity is not equal to 0, but it will iterate the SFN and slot ID to the next occurrence, that will be, in that case, SFN 2 and slot ID 3 (which corresponds to 40 slots after slot 3 of SFN 0).

5.3.2.2 Downlink data transmission

Figure 5.3 represents what are the different components used for downlink data transmissions and associated processes. For the sake of simplicity, the channel mapping thread is not represented.

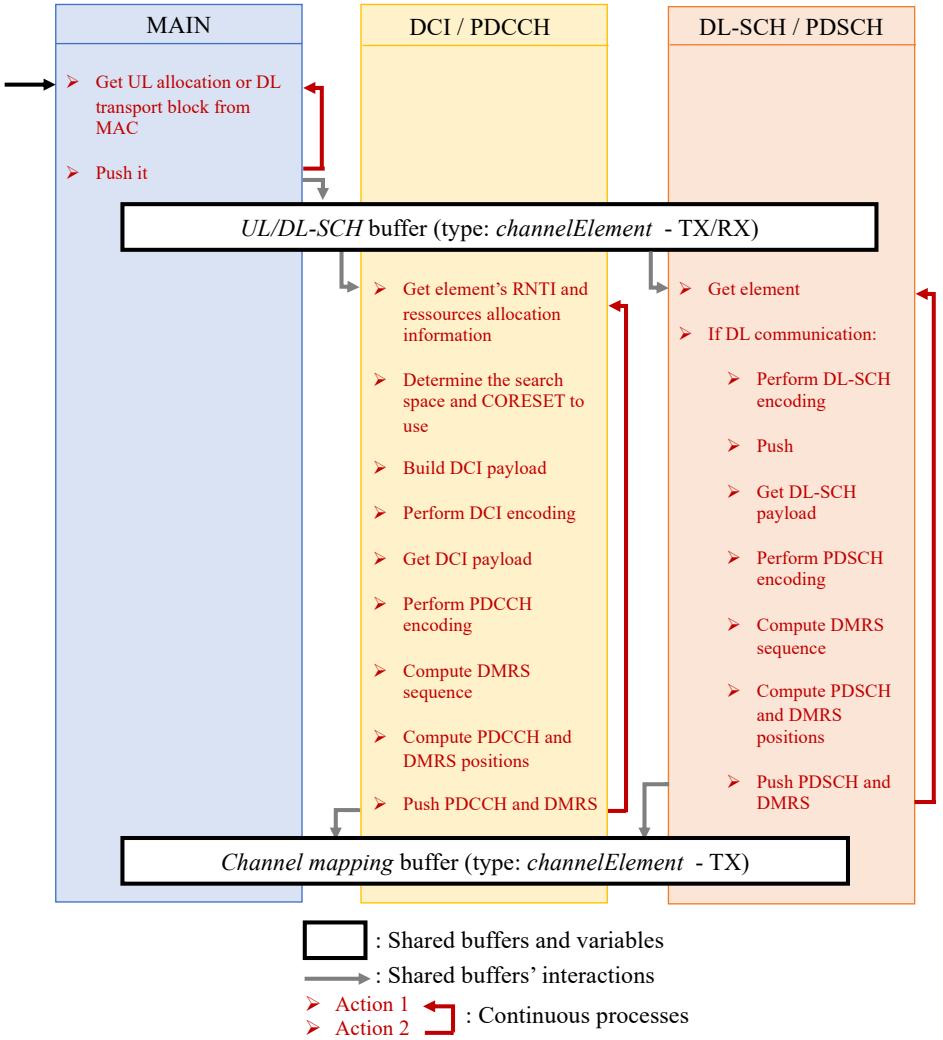


Figure 5.3: Overall gNodeB's procedure for downlink communication

5.3.2.3 Random access

Figure 5.4 represents how the different components interact together for the RA procedure. As the figure is complex, the procedure steps have been numbered, the downlink threads (DCI / PDCCH and DL-SCH / PDSCH threads in Figure 5.3) have been merged in one downlink thread and the channel mapping thread is not represented. The process is:

1. PRACH thread receives PRACH transmission. It pushes the information to the main thread using the PRACH buffer.

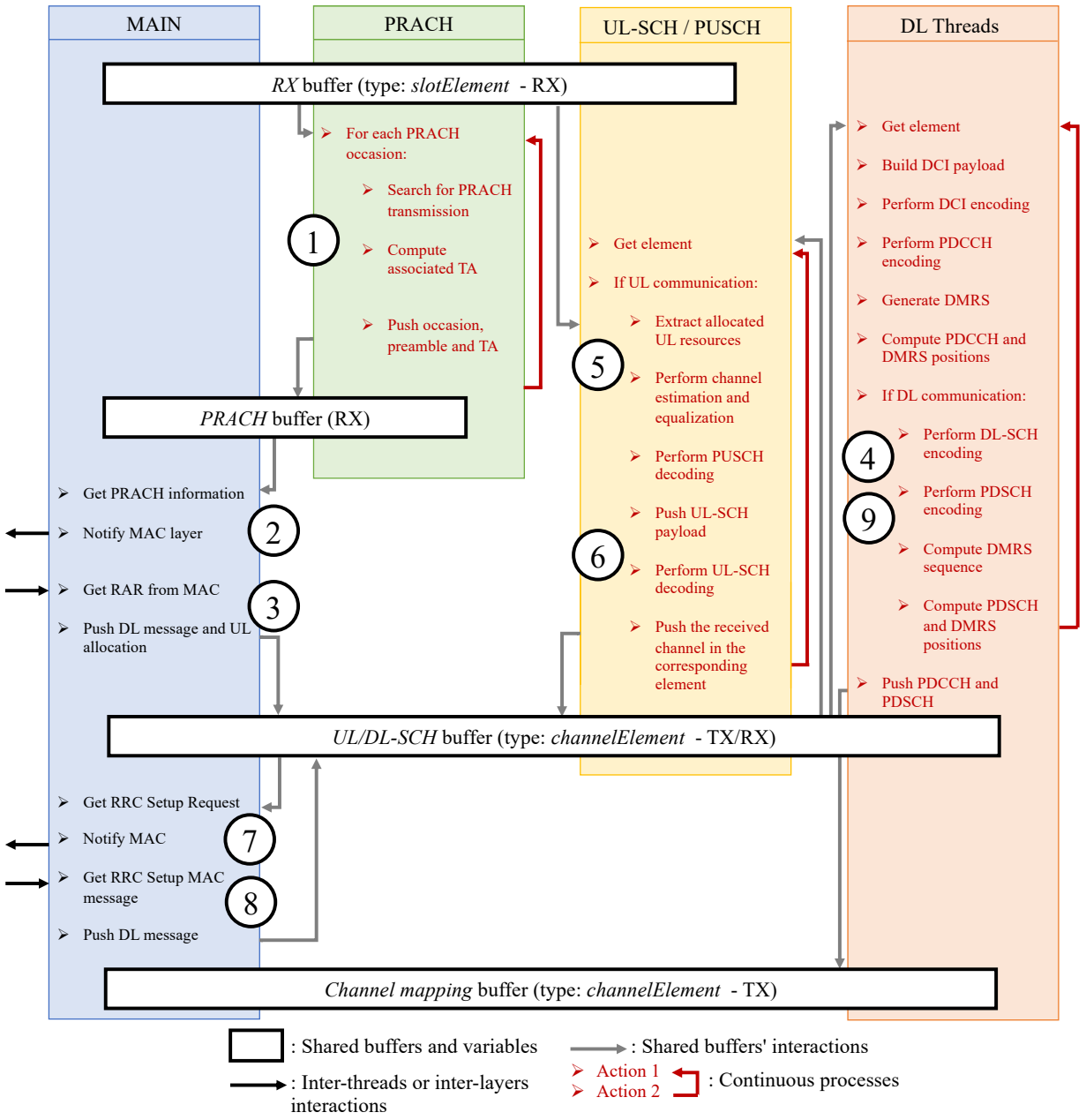


Figure 5.4: Overall gNodeB's random access procedure

2. Main thread forwards RA request to MAC layer and waits RAR.

3. It pushes the RAR into the *UL/DL-SCH* buffer. It includes the downlink RAR response and the uplink grant for RRC Setup Request.

4. DCI/PDCCH and DL-SCH/PDSCH encoding are performed for RAR transmission. The two channels are pushed into the *Channel mapping* buffer for transmission.

5. PUSCH thread extracts the uplink resources allocation pushed in step 3 by the main thread. It waits grant time has come and then extracts the corresponding IQ samples and perform PUSCH decoding.

6. After PUSCH decoding, UL-SCH decoding is performed and the transport block is pushed to the main thread.

7. The main thread forwards the RRC Setup Request to the MAC layer and waits for RRC Setup message.

8. It pushes the message in the *UL/DL-SCH* buffer.

9. Downlink threads handle the message by creating the DCI payload and performing DCI/PDCCH and DL-SCH/PDSCH encoding. The two channels are pushed into the *Channel mapping* buffer for transmission.

5.3.3 UE

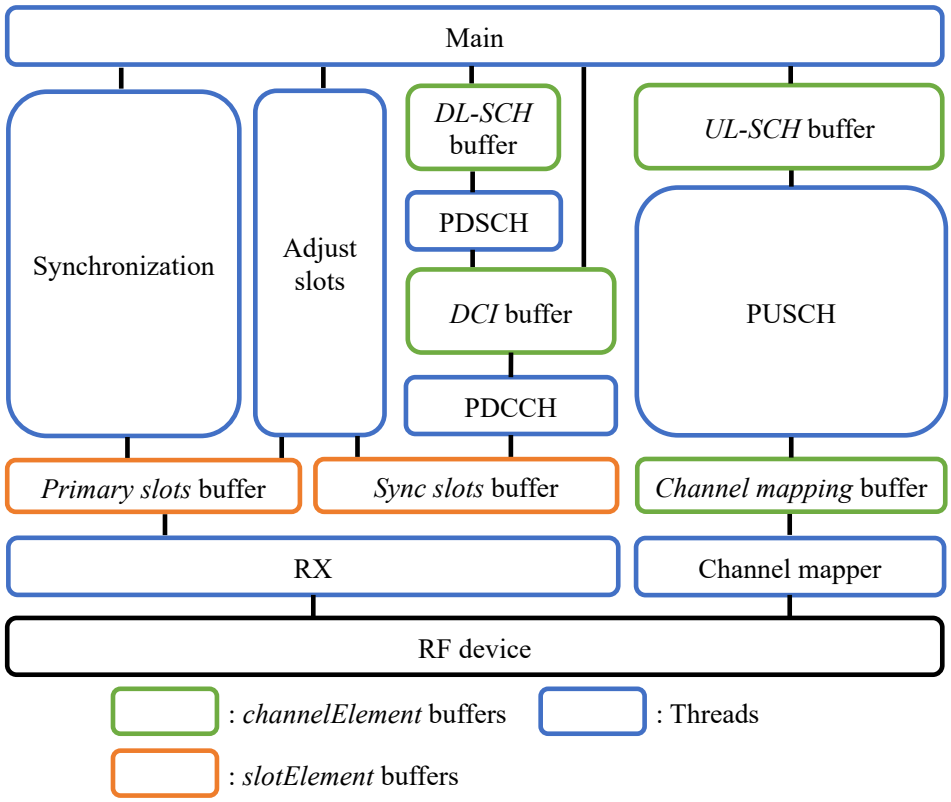


Figure 5.5: UE's overall structure

The UE's architecture, represented in Figure 5.5 differs from the gNodeB's one. The main difference is that the UE's structure must include the synchronization related components. The main elements are:

- The main thread. As for the gNodeB, it initializes the physical layer and will, in the future, implement the MAC and RRC API.
- The second element is the RX thread. This thread is responsible for receiving the signal from the RF device and putting it into a primary shared buffer. Once started, this thread continuously receives buffers of samples from the RF device. After receiving one buffer, it tags it with the current internal SFN and slot ID (initialized at 0 at the beginning) and pushes it into the shared buffer.
- The primary *slotElement* buffer (called *Primary slots* buffer in Figure 5.5), used by the RX thread to share slots with other threads. It is called primary because, at this level, the synchronization is not done. This means that the network's SFN and slot ID are unknown, and the slot is not cell-synchronized, i.e., it cannot be used to recover the slot's OFDM grid.
- The synchronization thread. It will be responsible for initial and continuous synchronization. Once started, this thread first performs PSS and SSS lookup within the signal as well as MIB decoding. This is initial time and frequency synchronization, and it enables to know where a cell's synchronized slot starts within a primary slot. Then, the periodic re-synchronization

process starts: an infinite loop continuously searching for PSS and SSS signals to keep time and frequency synchronization.

- The slots adjustment thread is responsible for building the cell's synchronized slots and tagging them with the accurate SFN and slot ID. This thread uses synchronization information provided by the synchronization thread and the *Primary slots* buffer.
- The synchronized *slotElement* buffer (called *Sync slots* buffer in Figure 5.5). It contains the cell's synchronized slots pushed by the slots adjustment thread.
- The PDCCH thread. It continuously blind searches for DCI transmissions. The search space and CORESET to be used for blind search are given by the RRC Setup message.
- A DCI *channelElement* buffer (called *DCI* buffer in Figure 5.5) used by the PDCCH thread to push decoded DCI payloads. This buffer is connected to the PDSCH thread for downlink DCI and to the main thread for uplink DCI.
- A PDSCH thread: get downlink resource allocation from *DCI* buffer and extract the corresponding IQ samples. Perform PDSCH and DL-SCH decoding.
- The DL-SCH *channelElement* buffer (called *DL-SCH* buffer in Figure 5.5) used by the PDSCH thread to push downlink transport blocks to the main thread.
- The UL-SCH *channelElement* buffer (called *UL-SCH* buffer in Figure 5.5) used by the main thread to push uplink transport blocks to be transmitted.
- PUSCH thread: get uplink transport block from the *UL-SCH* buffer, perform UL-SCH encoding and PUSCH encoding.
- A *channelElement* buffer for channel mapping (called *Channel mapping* buffer in Figure 5.5) and uplink channels transmission.
- Channel mapping thread: as for the gNodeB, a channel mapping thread is required to get all the uplink channels of the UE, place them into the OFDM grid, perform OFDM modulation and transmit them to the gNodeB.

5.3.3.1 Slots adjustment

Figure 5.6 represents the overall operation of the slots adjustment process. For the sake of simplicity, the synchronization thread is not represented but it is used to provide synchronization information for slots cutting.

5.3.3.2 Data communication

Figure 5.7 represents how the different threads work together for uplink and downlink data transmissions. The channel mapper and RX thread are not represented for readability reasons. The downlink process is:

1. The DCI thread blind searches for DCI transmission.
2. The PDSCH thread extracts and decodes the transport block.
3. The main thread pushes the received transport block to the MAC layer.

The uplink process is:

1. The DCI thread blind searches for DCI transmission.
2. The main thread notifies the MAC layer and waits for the transport block to be provided.
3. The PUSCH thread encodes the transport block and pushes it to the channel mapping thread.

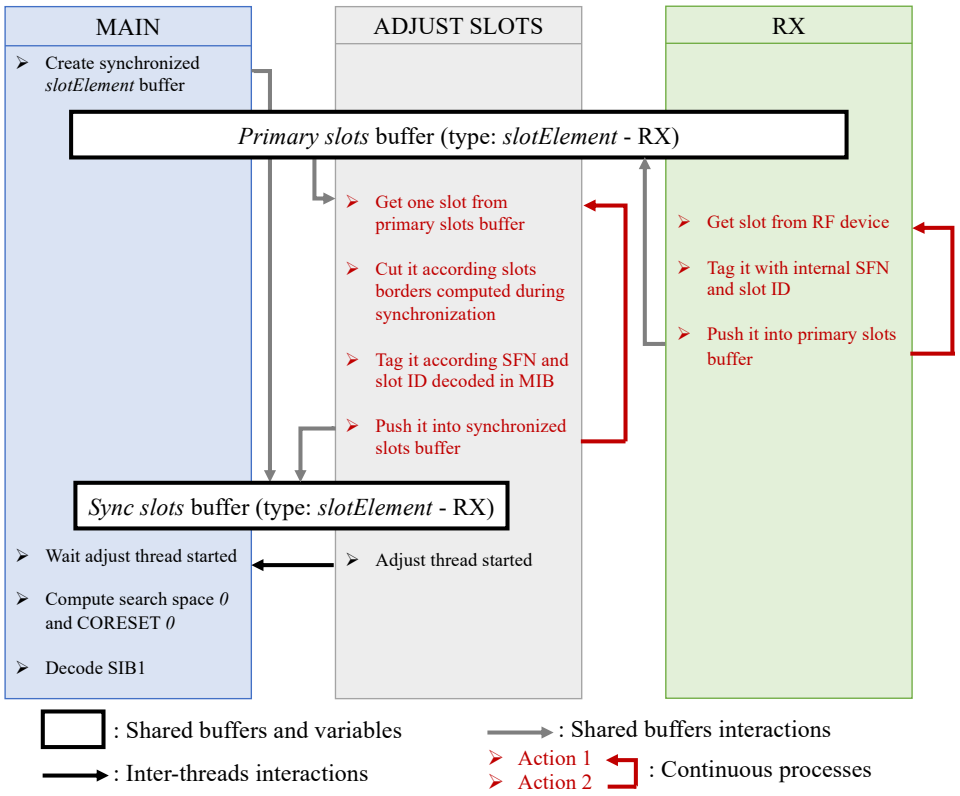


Figure 5.6: Overall UE procedure for slots adjustment

5.4 Conclusion

In this chapter, we introduced the *free5GRAN* project's structure and software architecture. Those two aspects are critical because the physical layer is a highly constrained system that evolves over time. Therefore, the project's structure and software architecture are of the utmost importance to guarantee that the physical layer can be easily updated over time and can address multiple performance constraints.

The project's structure is built so the functions and algorithms can easily be modified. It should enable one to implement and test new algorithms and also to implement future standards evolution.

The software architecture is built with modularity so that the PHY layer can be adapted to the different services and slices. Moreover, it relies on multi-threading, so the current architecture can easily be modified to implement a RAN splitting ready base-station.

This chapter concludes the first part of this thesis. First, the minimal procedures and functions chaining defined in the standard have been reviewed to provide a global understanding of the physical layer operation. Then, the functions whose algorithm is not defined in the standard have been detailed, and a possible implementation has been proposed. Finally, this chapter proposed an architecture that is used in *free5GRAN* to put together the procedures, mechanisms and algorithms in order to build a minimal gNodeB's and UE's 5G physical layer.

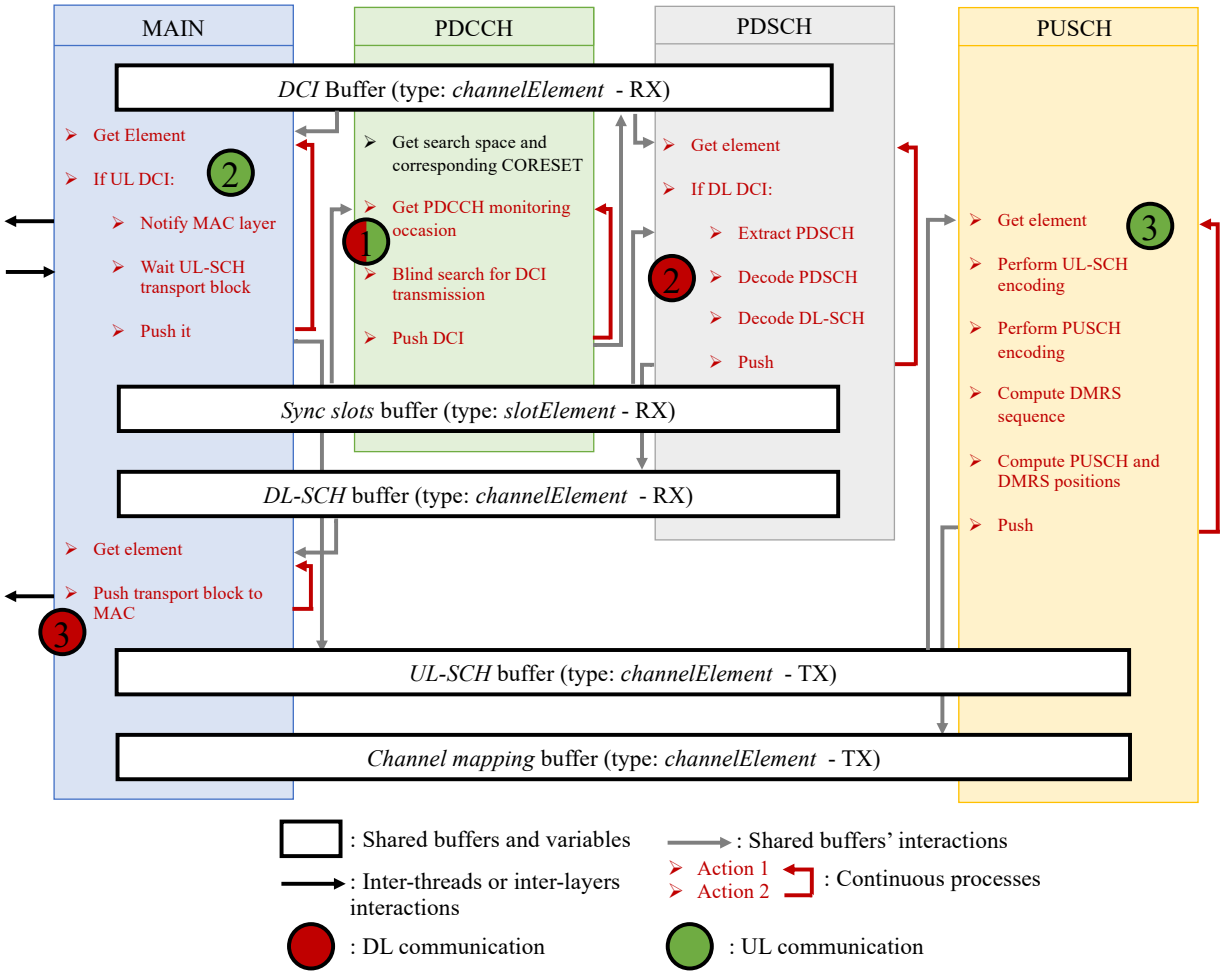


Figure 5.7: Overall UE's procedure for downlink and uplink data communications

Part II

5G network slicing

Introduction

The main evolution between 4G and 5G is the high heterogeneity of the terminals that might connect the network. Those terminals must not connect to the same data network; some connect to the internet while others connect to private data networks. Furthermore, the user traffic associated with the multiple terminals will be highly different, and some might be critical. Therefore, it is necessary to reproduce, on the 5G system, the concept of Virtual Private Network (VPN), which already exists on usual IP networks. Such VPNs have two main aspects. The first one is that it enables to create a virtual network between one or multiple data networks hosting applications, and the terminals, such as the public infrastructure is transparent for UEs, that are therefore working as if they were connected to a private mobile network. The second aspect is that it enables isolating and identifying the different user traffics on top of a single infrastructure.

Implementing such VPNs would not have been possible in 4G. Indeed, in 4G, the network is hardware-centric, which means that the different functions are grouped and implemented in dedicated hardware devices. The architecture is built so that the devices are close enough to the UE to provide better performances and are centralized enough so that the operator maintains a consistent understanding of the network. Therefore, it is impossible to dynamically place and dedicate resources to one logical network. The implementation of VPNs is therefore static and highly expensive, as it must be done by deploying dedicated hardware. Virtualization is the leading technology that makes VPNs possible in 5G. Indeed, in 5G, the different network functions are microservice software applications implemented to be executed on generic hardware. The way the functions are grouped and placed in the architecture is flexible, enabling to split the control and user plane functions. This split is named CUPS.

The virtualization technologies enable much better flow management and can be used to implement VPNs. Indeed, given that the constraints are on the user's traffic, user plane functions can be dedicated and optimized (configuration and position in the network) to one or another virtual network, whereas the control functions can be shared to keep a global view of the network. In 4G, dedicating a function to a virtual network requires the deployment and dedication of a device, whereas in 5G, dedicating a function can be made by deploying a software application on top of an existing generic hardware platform. Furthermore, cloud technologies can be leveraged to ease the provisioning and orchestration of the different functions and virtual networks. The two requirements for deploying VPNs on the network are that generic computing devices must be deployed at the different levels of the infrastructure (edge, middle-edge, and cloud) so that the functions can be placed based on the requirements. The second requirement is that the operator must be able to deploy and manage functions with ease and flexibility.

The 5G standard introduces network slicing as the mobile networks implementation of such VPNs. Indeed, network slicing aims at building virtual networks on top of the 5G physical infrastructure. Furthermore, those networks are private as they can be dedicated to one customer willing to connect a set of terminals to a specific data network. However, in IP networks, VPNs mainly refer to the topology which connects multiple sites to the same network whereas, in 5G, network slicing represents the logical networks connecting terminals to data networks but also the associated service and QoS profiles.

Network slicing is achieved by allocating resources to one or multiple slices. In this part, we first detail the network slicing concept and propose a review of the different resources allocated to network slices. Then, we introduce a model for RAN resource allocation which takes into account network slicing constraints. Finally, we propose a supervision system that controls the resource allocation

quality to strengthen the network's reliability.

Chapter 6

RAN architecture for network slicing

Contribution

In this chapter, we first clarify the concept of network slicing and define which type of network slicing might be achievable in the first deployments. Then, we explain how it can be implemented in the base station. Given that network slicing resides in resource allocation, we review the different resources that the base station can associate with slices. Furthermore, we also detail the technologies that can be used for associating resources to slices. Finally, we use the knowledge acquired during the implementation of the physical layer to review some key configuration aspects of the radio interface and PHY layer that can be customized to support heterogeneous QoS profiles.

6.1 Introduction

Network slicing is one of the central technology introduced in 5G. It enables to build logical networks on top of a single physical infrastructure that connect terminals to one or multiple data networks. Those logical networks can be customized to fit specific service requirements. Operators will use this technology to address new verticals. Given that network slicing is a critical component of the 5G system, it has received much attention. Considerable work has been done to explain how network slicing can be used to address new verticals. However, there is a lack of clear information about how network slicing can be implemented.

In the standard, it is defined that network slice instances are a set of resources that are associated with a slice. Therefore, the implementation of network slicing resides in resource allocation. However, it is not trivial to determine the different kinds of resources that can be used for network slicing implementation. Furthermore, some techniques and technologies must be used to associate resources with slices for each kind of resource.

This chapter first clarifies the concept of network slicing. Then, the different resources used for network slicing are reviewed to clearly explain how network slicing can be implemented. The different technologies and techniques that can be used to share or dedicate the resources to one or another slice are detailed for each type of resource. Finally, some key configuration aspects of the physical layer to support diverse network requirements are reviewed.

This work focuses on the base station's perspective and does not investigate the UE's and CN's levels. It is considered that a UE does not require access to multiple slices simultaneously and that the CN is deployed and provisioned to support the different logical networks.

6.2 Network slicing

A network slice is defined in TS23.501 [6] as a logical network that provides specific network capabilities and characteristics. Similarly, the network slice instance is a set of network function instances and the required resources (e.g., compute, storage, and networking resources) which form a deployed network slice. It can be understood as the mobile networks' implementation of the usual IP networks' VPNs.

Slice Service type	SST value
eMBB	1
uRLLC	2
mMTC	3
V2X	4

Table 6.1: Standardized SST (TS23.501 Table 5.15.2.2-1 [6])

Therefore, network slicing refers to the ability of the 5G network and the underlying infrastructure to operate multiple logical networks. A network slice is a logical network between terminals and one or a couple of data networks, and the network slice instance refers to the different resources associated with a network slice. Two parameters define a network slice. The first one is the service type to which the slice is dedicated. It refers to the addressed business vertical and corresponding kind of UEs that are likely to connect the slice. For example, all the smartphones will be associated with the same service type, and the connected vehicles will be associated with another. Multiple types of user traffic can be supported within one kind of service but must have close network requirements and constraints. The second parameter that defines a slice is the customer or set of customers to which it is associated. The main group of customers is the public end-users, and another group could be prime users. The customer can also be companies like car manufacturers in the case of connected vehicles.

In the standard (TS23.501 [6]), a slice is identified by its Slice Service Type (SST) and a Slice Differentiator (SD). The SST defines the slice associated service type and can be either a standardized value, as detailed in Table 6.1, or an operator-specific value. The SD is an operator-specific value that identifies a customer or group of customers with a given SST. The network manages the different slices based on the identifier (SST + SD). Each network function exposes its list of supported slices, and the network will allow a UE to connect a slice based on the availability of network functions that support the required slice.

A network slice creates a logical network between terminals and one or multiple data networks. When a UE connects the network and is associated with a specific slice, multiple PDU sessions are created from the terminal to the different data networks that the UE can connect. The PDU session can be understood as being a session of the VPN between a UE and a data network. The QoS profiles corresponding to the different user traffic types likely to go through a specific data network are configured for each PDU session. For example, two PDU sessions might be defined for smartphones, one toward the IP Multimedia Subsystem (IMS), where the configured QoS profiles are signaling and conversational voice (5QI 5 and 1) and another one toward the internet, where the configured QoS profiles are video streaming and TCP traffic (5QI 8).

As long as possible, the different slices are deployed using shared resources, granting the best optimization and flexibility. However, some slices require dedicated resources for two main reasons. The first one is that some slices have strong isolation, control, and reliability constraints, which imposes the use of dedicated resources. The second reason is that the user traffics associated with the different slices can be so heterogeneous that the traffics requirements respect cannot be guaranteed using shared resources. Therefore, the two main challenges while implementing network slicing are the association of the different resources (like network or computing resources) with the slices and the configuration of the resources to support a single or set of services.

In this chapter, we first review the different kinds of resources to study which can be shared between the slices and which must be dedicated. Then, some key configuration aspects of the PHY layer are detailed to support the different services.

6.3 Scope of work

Given that the main concern about network slicing is resource allocation, it is essential to precise which kind of network slicing is considered in this work so that resource allocation can be made accordingly. The global objective, which is the object of much research, is dynamic network slicing:

a UE connects the network and requires access to a given slice and service. Then, the network dynamically and autonomously allocates the resources to the UE depending on the required service type. Furthermore, all the slices are available everywhere in the network, and every cell has to be able to serve all the slices simultaneously. This works for non-critical and weakly constrained slices: all the slices are multiplexed by the base-station and access the network one after another. However, for highly constrained slices such as uRLLC, it is a complex problem regarding scheduling and resource allocation. First, this would require the operator to provision low latency capabilities everywhere in the network, which would have a considerable cost. Furthermore, this would be achievable with one uRLLC slice activated among other eMBB slices, as the network can prioritize uRLLC. Nevertheless, this becomes highly challenging if there is a massive amount of uRLLC slices that are activated simultaneously in a single cell, as the network will have to prioritize uRLLC slices among other uRLLC slices, which will break the reliability constraint of such services.

Therefore, it appears that dynamic network slicing, thought as "every slice can be activated everywhere at every time", might be tricky with the current technologies. In this chapter, we assume another kind of network slicing must be considered, at least for the first deployments. This other type can be called semi-static network slicing and is mainly based on provisioning. The operator knows which slice can be activated in which cell and at what time. This means that for a given cell in the network and at a given time, the operator knows the slices that can be activated and provisions the network so that the cell can serve all the potential slices. Furthermore, each slice's services and underlying traffic constraints must also be known in advance.

It is semi-static as the presence of a slice in a given cell is not fully dynamic and requires a call admission function. Such function validates that the slice required by a UE is active and that there are enough resources in the cell to serve the UE. However, it is not fully static as the presence of a slice in a cell can evolve with time as long as the operator has the agility to properly re-provision the network. Furthermore, the slice-dedicated resources are dynamically preempted by the slice when it is activated but can be used by other slices when the slice is not activated. On the other hand, this is not fully dynamic as any slice cannot be activated anytime in every network cell. This is a radically different way of thinking about network slicing. In this chapter, it is assumed that the operator knows in advance the different slices that can be activated in an area and places the resources (like base stations, CPUs, or memory) depending on the slices' requirements.

Furthermore, we assume that each network slice is associated with a single application's type and that there is no heterogeneous traffic multiplexing within one slice. We propose a slicing implementation architecture for one cell where three slices can be activated: eMBB, mMTC and uRLLC. This is a fundamental assumption for our work, all the architecture elements and propositions consider semi-static network slicing.

6.4 Resources association

This section investigates how the different resources used by a 5G base station can be associated with the different slices. In addition, the cases where the resources cannot be shared between slices are discussed. The two main features that the network must provide to the slices are isolation and heterogeneous traffic multiplexing.

At the RAN level, the main conflict between heterogeneous traffics requirements and constraints is between throughput (network's capacity) and reliability and delay. Indeed, the approach is not the same while building the network for high throughput and low latency services. To implement high throughput services, the usual question is how to optimize resource sharing to maximize the throughput while keeping a reasonable latency. On the other side, to implement low delay services with high reliability, it is first considered that the resources are dedicated to ensure network control, and then the question is how much the resources can be shared while still respecting and controlling the delay requirements.

This section investigates how the different resources can be used to implement a multi-slice base station with the objective of explaining how to reconcile the two approaches for deploying low latency and high throughput services on a single system. Table 6.2 gives an overview of the different resources

Level	Associated resources
RF	RF device
Radio interface	BWPs
Network	RAN layers instances
System	Computing resources and OS scheduler
Software	Software design and algorithms
Cross-haul	Segment routing policies and physical nodes

Table 6.2: Resources associated with the slices in the base-station

associated with the slices.

6.4.1 RF device and RF scheduler

With the virtualization of the network functions, the RF device becomes the only hardware component of the base station. It is possible to deploy a dedicated RF device for each slice or service type, ensuring isolation between the slices. However, this does not enable to manage the network and slices dynamically. Therefore, this work proposes sharing the RF device between the slices. The operator deploys the RF devices across the network to guarantee a good coverage, after which it manages the different slices without needing on-site intervention.

Nevertheless, the RF device can be slice-aware. Indeed, most of the time (especially in the context of open-RAN), the RF device is connected to the base station through a transport network that carries the signal to be transmitted and received. A slice-agnostic RF device would have a single data flow with the base station to transfer IQ samples of the signal, whereas a slice-aware RF device has one data flow per slice. Therefore, in case of troubles on the transport network, both the RF device and base-station can select which slice to prioritize. This guarantees the prioritization of the critical slices for the access to the RF device. This mechanism requires the use of a flexible RF scheduler, which manages the access of the different slices to the transport network based on the transport network's health and slices requirements. It is called flexible as it must be able to support the dynamic creation, deletion, and adjusting of the different slices, which might imply an update of the prioritization rules.

6.4.2 Radio interface resources

On the radio interface level, the main resource used by the base station is the bandwidth. There are two ways to share the bandwidth between different UEs:

- First, in usual OFDMA systems, like 4G or 5G, the bandwidth is shared between all the terminals by scheduling the access to the resource at the MAC layer. In this configuration, the radio resource is shared between all the slices and the MAC scheduler ensures the respect of the different services requirements.

- The second way is to use BWPs, introduced in section 1.3.2.2.3. A BWP is a part of the cell's bandwidth with a specific configuration. The purpose of the BWP is three-fold. It can first be used for energy saving as the full cell's band is split, and UEs with power consumption constraints are operating on a portion of the band and not on the full cell's bandwidth which reduces the power consumption. Furthermore, it can be used to have multiple radio configurations within a single cell. Finally, it can isolate the resources between different groups of UEs.

Even if sharing the resources between the slices is the best in terms of optimization and flexibility, it does not ensure the respect of the different slices heterogeneous traffics and isolation requirements. In this chapter, it is proposed to specify BWPs to the different types of services. Two BWPs are defined for the eMBB and mMTC services, where the corresponding radio configuration is optimized for high throughput and low energy consumption. The different slices associated with those two service types share the same BWP given that there is no strong isolation requirement. Finally, each

uRLLC slice will be dedicated a BWP. Beyond optimizing the radio configuration, the objective is to grant isolation to the uRLLC slices. In this chapter, we consider that there are only three slices that are one eMBB slice, one mMTC slice, and one uRLLC slice. Three BWPs are defined, and each slice has its dedicated BWP.

In a more general perspective, BWPs can be configured for each type of service with close underlying traffics requirements. When the associated slices do not require isolation, the BWPs are shared between the slices to increase the flexibility and optimization of the radio resources. However, it is possible to allocate radio resources to one or another slice which requires strong isolation by dedicating a BWP to a single slice.

6.4.3 RAN resources

At the RAN level, the available resources are the protocol stack layers. In usual systems, the protocol stack is monolithic, which means that all the slices share a single protocol stack. However, in 5G, virtualization and functional split can be leveraged to dedicate the network's resources to one or another slice. This section reviews the functional split in the context of network slicing, and a RAN stack composition is proposed.

6.4.3.1 5G functional split

As introduced in section 1.4, 5G functional split gives the ability to split the RAN stack into different groups of functions or layers. The RAN implementation becomes flexible and must not be mandatory deployed with a monolithic architecture. This work considers that the functions are split and grouped by layers (PHY, MAC, RLC etc).

5G functional split enables the implementation of different layers in different software applications that can be deployed on different physical devices. Furthermore, it enables the building of a flexible RAN architecture as there can be multiple replicas of a single layer in a single RAN stack. The unique constraint to integrate multiple replicas of a single layer in a stack is that upper and lower layers must be able to determine which replica of the layer they must communicate with. For example, multiple MAC layers can exist within one RAN stack as long as the RLC layer can distinguish which MAC layer to use for each payload (it can be based on the payload's DRB for example). Moreover, the PHY layer must also be able to send the uplink transport block to the correct MAC layer. This can be done as the uplink allocation is received from the MAC layer. The PHY layer can therefore forward the uplink transport block to the MAC layer that communicated the corresponding uplink allocation.

Multiple versions of a single layer can be used to dedicate resources to one or another slice or type of service. Furthermore, the deployment and software implementation options used for one layer might differ (and be specific to a slice or service) from one version to another. Therefore, in this section, a layer instance is defined as being one version of a protocol layer that can be optimized for one or another slice or type of service. In this context, RAN layers can be considered as being network resources that the gNodeB can associate with slices.

5G functional split is thus leveraged as the 5G RAN stack of a single cell can be composed of common layers instances (shared for multiple slices) or slice-specific layers instances (dedicated to one slice). Dedicating layer instances to one slice provides isolation between slices and better controls the available resources and the processing delay. Hence, it might be used for low delay slices. On the other side, for high throughput slices, it is worth sharing the instances as it enables a better optimization of the resources to provide the highest throughput.

6.4.3.2 RAN stack composition

Given that the different layers can be used with flexibility, depending on the requirements and constraints, a possible RAN architecture composed of common and slice-specific layers instances is proposed in this section. Similar architectures have been proposed for 4G in [35]. The main difference is that the functional split is standardized in 5G, whereas the work in [35] is a non-standardized modification of the 4G RAN stack.

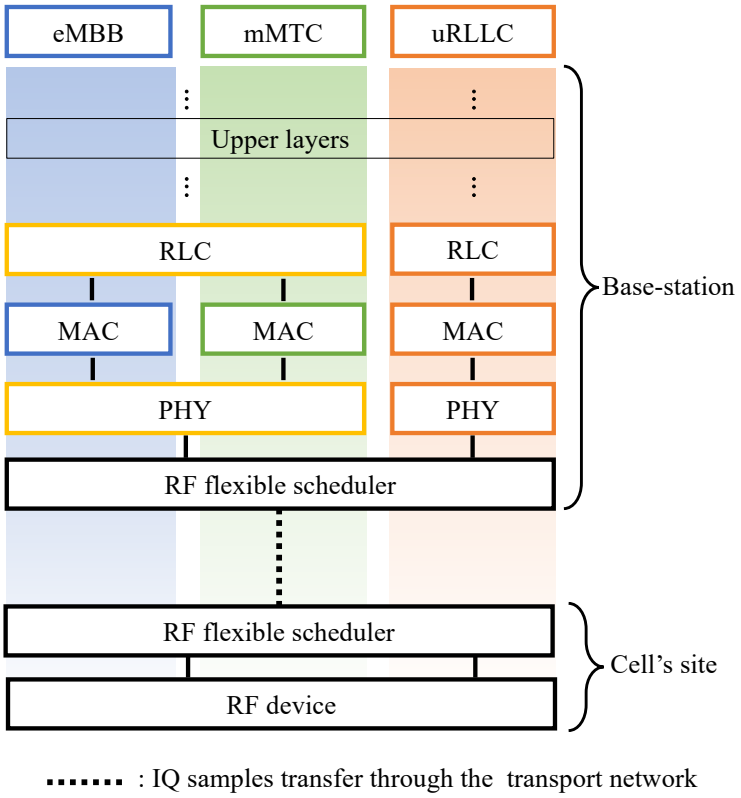


Figure 6.1: Proposed RAN architecture

Figure 6.1 represents the RAN architecture. It is proposed to share the operating band and RF device between the different slices. Depending on the constraints, other resources and layers can be shared or slice-specific. The three lowest components are the RF device and the two RF flexible schedulers introduced in section 6.4.1.

At the PHY layer, it is crucial to split the components between the uRLLC slice and the others as there are many functions whose performances regarding delay and throughput profoundly depend on the implementation. Dedicating a PHY instance to one or another slice enables to implement different algorithms for each function depending on the constraints. Furthermore, the PHY layer is usually implemented in hardware. However, with the rise of virtualization technologies, more and more implementations are either hybrid (software using hardware acceleration) or full software implementations. Each of these three options has pros and cons and better corresponds to one kind of requirement (like reliability, performance, or flexibility). Therefore, dedicating a PHY instance to a slice enables the use of different kinds of implementations (hardware, hybrid, or software) for different slices.

In the architecture presented in Figure 6.1, single MAC layer instances have been deployed for each slice. The main reason for deploying dedicated MAC layer instances is that MAC contains the RAN scheduling algorithm. This function is a key component for achieving network slicing as it is responsible for allocating resources to UEs and respecting slices requirements. A lot of research projects focused on the design of scheduling algorithms that support heterogeneous slices. The scheduler must provide isolation between the slices and serve heterogeneous traffics but it should also provide the ability to compute outage probability for network control and reliability. Therefore, this work proposes deploying one MAC instance for each slice to have a slice-dedicated MAC scheduler. The classic Proportional Fair algorithm can be used for the eMBB slice as this is a good trade-off between throughput maximization and network accessibility. An introduction of this algorithm applied to OFDMA systems is given in [36]. For the uRLLC slice, the Earliest Deadline First algorithm appears to be a good option to respect the delay constraints. An introduction to this algorithm is given in [37]. Finally, for the mMTC slice, the scheduling algorithm might depend on the traffic type: it will not be the same if there is mainly uplink traffic, downlink traffic, or a balance between uplink

and downlink.

From the RLC layer, it is proposed to share all the components between the eMBB and mMTC slices and to deploy dedicated instances for the uRLLC slice. Indeed, the layers above RLC can be considered slice-agnostic as they mainly contain encapsulation, de-capsulation, and flow management functions. Nevertheless, it is still interesting to dedicate layers' instances to the uRLLC slice as it enables first to optimize the component for real-time processing but also to isolate the network resources for the critical slices. In a more general perspective, it can be considered that the layers above RLC are slice agnostic for non-critical slices, but it is still worth dedicating resources to critical services.

6.4.4 System resources

On the system level, there are two kinds of resources that the base station uses. The first type is computing resources like Central Processing Unit (CPU) and memory and the second one is Operating System (OS) scheduler, which manages the access of the instances to the computing resources. Those two kinds of resources are deeply correlated. The resources must be dedicated to the layer instances associated with slices with high control and delay constraints. Indeed, dedicating computing resources ensures that the instances have immediate access to the computing resources so that the network's response is the shortest possible, and it also provides the best possible control of the network as both the set of resources and set of UEs with strong requirements are known. Furthermore, it is worth dedicating the OS scheduler as the computing resources must be accessed with strict timing requirements, which is not optimized for other services like eMBB. The layer's instances associated with the eMBB and mMTC slices can share both the computing and OS resources as it enables better optimization of the resources usage and high flexibility. Indeed, those slices do not have specific requirements in terms of isolation. Furthermore, given that the latency requirements are not strong, the associated instances can afford not to have immediate access to the resources. Depending on the layer and the implementation design, different OS schedulers might be used. A comparison between two main general-purpose schedulers, *FreeBSD ULE* and *Linux CFS*, can be found in [38].

Three options exist today to dedicate computing resources and OS features to slices:

- The first one is to deploy the components on different devices where each device has its dedicated resources and an OS which is configured to satisfy the constraints. The instances with low constraints can be deployed on shared devices, and highly-constrained instances can be deployed on dedicated ones. However, this requires the transport network connecting all the devices to meet the slices requirements. As this is not trivial, this is further investigated in section 6.4.6. Furthermore, this does not meet the flexibility and optimization requirements, as the resources are statically associated with slices, and the allocation cannot be dynamically re-adjusted.

- The second solution is to deploy all the components on a single machine running different Virtual Machine (VM) where each VM instance has its own OS and resources. For example, an hypervisor like *OpenStack* (well introduced in [39]) can be deployed on a host device running a generic OS. Given that usual hypervisors might not be able to serve both real-time and general-purpose VMs, other CPU resource management systems must be integrated into those hypervisors, as it was done in *RT-OpenStack*, introduced in [40] or in *Poris*, introduced in [41]. Therefore, two VMs can be deployed, one for the eMBB and mMTC slices and one for the uRLLC slice. The first one can use a standard OS whereas, the second one can use a real-time OS like *RTLinux* (which is defined in [42]).

- The last solution is to deploy all the components on one device and create pools of resources, where each pool has its scheduler. The components are allocated to a resource group that matches the resources and scheduling requirements of the slice running on it. In modern Linux distributions, this can be done using *cpuset* [43] mechanism. An introduction of this mechanism applied to high-performance computing is given in [44]. Two resource sets can be declared where one will contain the CPUs, memory and scheduler optimized for eMBB and mMTC slices, and the other one will contain resources optimized for uRLLC. This solution is promising

as all the components can be run on the same host, and it does not involve virtual machines. Cloud technologies like containers and *Kubernetes* can be leveraged to ease the management and orchestration of the layers instances and the corresponding dedicated resources. Furthermore, another key asset of this method is that it is much easier to resize and re-optimize the resource allocation compared to the two previous options. The use of Linux CPU scheduler for real-time applications running on containers has been investigated in [45].

A good introduction to the differences between VMs and containers is given in [46], which also introduces *Kubernetes*. Furthermore, performances comparison between VMs and containers can be found in [47] and [48]. Containers might be the best solution as they provide excellent capabilities in terms of resources and OS functions allocation together with the best flexibility and greatest lightness.

6.4.5 Software resources

On the software level, the resources available are the software architecture and algorithms. This section investigates the software designs that might be used to implement the layer's instances associated with the different slices. Indeed, components serving eMBB/mMTC must be designed to support ultra high throughput and a massive number of UEs whereas, for uRLLC, it must be designed with a focus on latency. Differentiating the implementation method based on the slice's constraints is made possible by dedicating layers instances to slices.

On one side, components with low delay and high reliability constraints may be implemented using a real-time processing design. A good introduction to real-time processing of streams is given in [49]. As explained at the beginning of section 6.4, the implementation of functions with a focus on delay starts by stating that the resources are dedicated to one or another UE, and then resources sharing can be envisioned as long as it respects the delay constraints.

Therefore, in a first step, the resources are dedicated to UEs by extensively using multi-threading. Each UE is allocated a thread and, beyond a thread, a CPU. This method provides strong reliability as it guarantees that the computing resources are available when there are data transmissions between the gNodeB and the UE. Immediate access to the computing resources is guaranteed. However, this is not enough as this does not enable control of the delay. Controlling the delay is made by having a precise estimation of the algorithmic complexity and the number of operations of the different functions performed in a layer. Given that a CPU is dedicated to one UE and that the CPU's performances and the number of operations are known, the network infrastructure can compute the processing delay of a layer for one user traffic transmission. Even if this method consumes a lot of CPUs, it provides reliability and control over the delay, which is critical for some slices.

In this context, it is fundamental to estimate precisely the algorithmic complexity and number of operations of all the functions of the different layers. Therefore, for the slices with low delay and high reliability constraints, probabilistic algorithms must not be used as the number of operations required to achieve a function is not precisely known by advance. This substantially impacts the different layers, especially at the PHY layer, as this kind of algorithm is typically used for channel decoding as proposed in [50] for convolutional codes. From a more general perspective, the complexity and number of operations might not be easy to determine, even for non-probabilistic algorithms. For example, for belief propagation, which is one of the possible algorithms for the decoding of LDPC codes in 5G, the number of operations of one iteration can be computed (as described in [51]), but its number of iterations depends on the quality of the received signal. Therefore, the processing delay of this algorithm cannot be estimated precisely, but an upper bound can be given by imposing a maximum number of iterations.

Once an implementation with dedicated resources is made, it is possible to start sharing those resources between multiple UEs. In this case, the processing delay cannot be estimated precisely as access to the computing resources cannot be guaranteed immediately. However, it is possible to compute an upper bound of the processing delay as the number of UEs per thread, and the processing delay of one UE are known. The upper bound of the processing delay corresponds to the worst case where the UE must wait for all the other UEs to perform a task before accessing the resources. The upper bound of the delay equals the number of UEs times the processing delay of one UE's

transmission. Furthermore, when resources are shared between UEs, the context switching overhead from one thread to another must be considered. Tools like *lmbench* [52] on Linux exist to estimate this overhead.

On the other hand, this design does not meet high throughput and UEs density requirements. Indeed, the number of threads increases with the number of UEs (or tasks in general), and a vast amount of CPUs might be required. This would be highly un-optimized as eMBB, and mMTC traffic are uneven, and the dedicated CPUs might be unused a significant part of the time. The different threads must therefore be multiplexed onto CPUs. This is achievable for a small number of threads, but the context switching overhead from one thread to another might explode when the number of threads increases. This is why other software designs are required, among which batch processing seems to be the most interesting.

The different options for implementing functions with low delay requirements and others with high throughput requirements is widely reviewed like in [53] and [54]. In general, it appears that software architecture and design are fundamental in network slicing, especially for ultra-reliable communications components where network control is of the utmost importance. Indeed, the reliability of a component is given by its outage probability and processing time. It is then required to choose an architecture and algorithms which provide the ability to compute those two key information. Beyond algorithm complexity, all the operations have to be considered.

6.4.6 Transport network resources

In this last section, we investigate the possibility of spreading all the instances represented in Figure 6.1 across the network. This is the concept introduced by open-RAN, where the base station is split into three components: the RU, DU, and CU. Then, those three components are spread across the network.

In this section, we consider that the RAN functions are virtualized and deployed in the cloud. Therefore, the main concern is the network connecting the functions, called the cross-haul network. The first naive solution is to dedicate point-to-point links like dark fiber, but this is not flexible nor optimized. Therefore, the cross-haul network has to leverage frame-based (Ethernet) or packet-based (IP, MPLS, Segment Routing) networks. The question is then: what technologies can be used on the cross-haul network to support dis-aggregated RAN deployments? The constraints that are on the cross-haul network depend on the layer and the slices constraints.

First, for the two lower layers, the network constraints are huge. Indeed, the samples sent from the PHY layer to the RF device have to arrive in order, with a very short delay, and with high reliability. Indeed, the loss of a single sample or the inversion of groups of samples imply a loss of data and a loss of cell's synchronization. Such losses can be bearable for non-critical slices but not highly critical ones. Finally, the network must support ultra-high throughput as signal transmission requires a huge capacity. For example, let us consider a cell with a bandwidth of 100MHz with 32x32 Massive-MIMO and that the samples are encoded in 8 bits complexes ($2 \cdot 8$ bits). The required network throughput, just for the data, is around $100 \cdot 10^6 \cdot 8 \cdot 2 \cdot 32 = 51.2$ Gbits/s. In addition, different network headers for traffic routing and samples re-ordering might be added and increase the throughput, but this might be mitigated by the compression mechanism used to transport samples. All those constraints make spreading the two lower components over the cross-haul difficult and expensive.

For the upper layers, the constraints are not as great as the lower ones as the data payloads contain bits sequences and not IQ samples. Two major issues remain. The first one is the synchronization between the components. Indeed, both the MAC and RLC layers use timers for different network procedures. If components are spread across the network, the transmission time between the component and the RF device must be considered while computing the timing expiration of one procedure. Furthermore, for the uRLLC slice, the backhaul network still has to ensure the reliability of the communications and the respect of a specific delay between the components, which is not trivial on a classic IP networks.

On the cross-haul network level, the resources available are the physical nodes (routers and switches), the links deployed to interconnect the RAN functions, and the virtualized network constructs in the servers where these functions are deployed and running.

These resources form the cross-haul transport network, which includes multiple segments (front-haul, mid-haul, and back-haul) with different latency, jitter, packet drop, and bandwidth service level objectives. With open-RAN, the Common Public Radio Interface (CPRI) is evolving towards eCPRI, which decreases the required capacity between the DU and the RU. With that interface enhancement, it is now possible to leverage frame or packet-based transport networks (like Ethernet, IP or Segment Routing), which provide much more flexibility compared to legacy transport methods like dark fiber, Dense Wavelength-Division Multiplexing (DWDM) or Passive Optical Networks (PONs) commonly used in 4G architectures.

In the following, we consider only frame or packet-based transport networks as they allow flexible deployments of 5G open-RAN architecture, and we describe the mechanisms available to meet stringent open-RAN requirements.

6.4.6.1 Physical cross-haul transport network

First, the physical nodes of the cross-haul must be configured to support the different slices. Two main technologies can be leveraged to support slices with heterogeneous constraints.

6.4.6.1.1 Time Sensitive Network: The open-RAN front-haul connectivity requires low-latency and jitter requirements which can be addressed by IEEE 802.1 Time Sensitive Networking [55]. Time Sensitive Network (TSN) is an ethernet-based transport network and consists of multiple standards.

A TSN profile is defined for each TSN use case (like TSN applied to open-RAN front-haul) to narrow the breadth of features and requirements and retain only the required options, protocols, and configuration parameters. TSN profiles have been standardized to support eCPRI front-haul interfaces over ethernet-based transport. The standard seeks to reduce latency for the aggregation and switching portions of the front-haul network. It defines two main profiles:

- Profile A addresses latency in two ways. First, time-sensitive traffic is prioritized over non-time-sensitive traffic so that the non-time-sensitive traffic gets buffered as time-sensitive traffic receives scheduling precedence. Additionally, Profile A limits the maximum frame size to $2,000$ bytes to minimize the time that time-sensitive traffic must wait when it arrives behind non-time-sensitive traffic.
- Profile B goes a step further by adding frame preemption, which allows transmission-in-progress of a non-time-sensitive frame to be interrupted when a time-sensitive frame needs to be transmitted. Once the time-sensitive frame is transmitted, the transmission of the non-critical transmission can resume. Profile B with frame preemption has two implications compared to Profile A. First, frame preemption eliminates the need to limit frame size. Second, it ensures a bounded latency, which is impossible with Profile A. This is a crucial feature with regard to network control requirements.

6.4.6.1.2 Segment routing: Segment routing (introduced in [56] and standardized in IETF RFC 8402 [57]) runs natively on an MPLS or IPv6 data planes. It provides complete control over the forwarding paths by combining simple network instructions with the source routing paradigm. Segment routing allows a node to steer a packet flow along any path. The head-end is the node where the instructions for segment routing are written into the packets and hence becomes the starting node for a specific segment routing path. Intermediate per-path states are eliminated thanks to source routing.

A segment routing policy is an ordered list of segments (i.e., instructions) representing a source-routed policy. The policy takes place between a source and destination pair. The source calculates the path and encodes it in the packet header as a segment. The segment instructs the routers in the provider network to follow the specified path instead of the shortest path calculated by the Interior Gateway Protocol (IGP). The source can also delegate the path computation to a centralized Path Computation Engine (PCE).

Flexible Algorithm (noted flex-Algo) is an algorithm that defines how the best path is computed by IGP. Routers advertise the support for the algorithm as a node capability. Segments are also advertised with an algorithm value and are tightly coupled with the algorithm itself.

In order to meet open-RAN systems transport requirements, segment routing policy and flex-Algo can be leveraged. Path computation constraints can be made based on attributes like latency, jitter, and reliability. Policies are installed in the head-end routers interconnecting with the open-RAN functions and guarantee end-to-end service level objectives related to the path constraints.

6.4.6.2 Virtualized cross-haul transport network

Given that the RAN functions are virtualized and deployed in servers, it is also necessary to optimize the virtualized routing and switching functions that enable the RAN instances to communicate with each other and with the physical network.

6.4.6.2.1 Real Time Performance: The first challenge to be addressed is to provide real-time performances and low latency on virtualization platforms where different workloads with different profiles (like real-time, high throughput, CPU or memory-intensive) share the same resources.

For a TTI down to 0.2ms, the preferred OS response time for L1/L2 must be less than 5 μs, and the determinism must also be preserved when the system is scaled out over many CPU cores. It means that, for the RAN functions, the same software solution might be used on systems of different sizes, ranging from 4 or 8 cores deployments to much larger systems, including central units with pooled resources on massive multi-core devices.

Assuming virtualization platforms are based on the Linux OS, it is possible to leverage Symmetrical Multi-Processing (SMP) Linux feature to consolidate real-time critical L1/L2 functions with more relaxed L2/L3 functions. Further information about SMP can be found in [58].

6.4.6.2.2 High Throughput: The second challenge is related to the high throughput required for open-RAN applications. Assuming applications are deployed on the Linux operating system, optimizing packet processing using the pool mode driver instead of the kernel driver and interacting with the network card or virtual devices in the user space is possible.

DPDK [59] is a commonly used framework for packet processing optimization. It leverages multi-core architectures and can achieve $\approx$ 20 million packets/second on a single Intel Xeon E5-2695v4 core with 64 bytes packets and $\approx$ 60 million packets/second with 4 cores system.

6.4.6.3 Physical and Virtualized Network anchoring

Assuming open-RAN functions are deployed as cloud-native applications on a container orchestration platform like *Kubernetes*, service level objectives must be advertised from the application infrastructure to the cross-haul transport network. Given that RAN functions use IP technologies, that inter-working is only possible with IP based transport networks.

In that case, the Border Gateway Protocol (BGP) (standardized in RFC 4271 [60]), which is responsible to advertise network reachability, is used to propagate QoS objectives information like low-latency or high-reliability. The BGP extended community attribute provides a mechanism for labeling information carried in BGP. It allows the usage of network policies based on the community value associated with an open-RAN application.

It is possible to seamlessly advertise a RAN application with a given QoS objective in the transport network. It is done by mapping the *Kubernetes* service (RAN function in this case) with a BGP extended community attribute. This mapping must be done in the Container Network Interface (CNI) plugin that is responsible for containers interface management.

For example, it can differentiate data flows between the eMBB and uRLLC slices. The uRLLC instances can be advertised with a specific extended community. It allows propagating the information to the segment routing transport infrastructure to handle the related traffic with a policy optimized for low latency. On the other hand, the eMBB instances are not announced with any specific community, and the traffic will be managed within the IGP cost-optimized topology.

6.5 Optimization of the PHY layer configuration

After reviewing the different levels of resources associated with the slices, this section investigates how to optimize the radio interface and PHY layer to support the heterogeneous user traffics. We focus on the radio interface and PHY layer as this topic has been widely investigated in this thesis. For end-to-end network slicing, all the network components must be optimized in such a way. In this section, we state that the three slices are given a dedicated BWP and that, therefore, the radio interface and PHY configuration can be slice-specific by configuring the corresponding BWP.

The first key configuration aspect is numerology. Indeed, in 5G, the SCS is flexible and can be tuned depending on the service requirements. For FR1 bands, the three possible SCS are 15kHz, 30kHz and 60kHz. The three slices can have different SCS:

- First, for uRLLC, a 60kHz SCS might be chosen as this provides the shortest symbol's duration, which enables the base station to be as responsive as possible. With this configuration, the time unit of the radio interface is the smallest possible, which is a starting point for further optimization of the PHY layer to finally provide the shortest possible end-to-end transmission delays.

- Then, for mMTC, a SCS of 15kHz might be chosen as it minimizes the frequency selectivity of the channel and, consequently, maximizes the channel's quality. Given that the channel's quality directly impacts the complexity of the signal processing functions and associated power consumption, it is worth selecting the smallest possible SCS for UEs with power consumption constraints.

- Finally, for eMBB terminals, which does not have strong delay requirements nor strict power consumption constraints, a SCS of 30kHz is chosen. This provides a trade-off between two constraints:

  - On one side, as for mMTC, it is worth selecting the smallest possible SCS for energy-saving purposes, as the terminals associated with eMBB slice are mainly battery-powered smartphones.
  - On the other side, given that eMBB is expected to provide high throughput, it can be expected that the BWP associated with this slice will be wide. Given that the number of points in the FFT/iFFT is inversely proportional to the SCS, the highest the SCS, the lowest the number of points and complexity of the FFT/iFFT. Therefore, with regard to the expected width of the BWP associated with eMBB, it is worth selecting the highest possible SCS to minimize the complexity of the time to frequency domain transforms.

Figure 6.2 represents the three slices together with their BWPs and associated SCS.

The second key configuration aspect of the PHY layer is the TDD UL/DL configuration. As represented in 6.2, depending on the slice, the downlink to uplink switching periodicity is not the same. This is fundamental for achieving network slicing. Indeed, between uplink and downlink symbols, it is better to add a guard period as recommended by GSMA in [61] (recommendation number 10). The guard period enables to mitigate the interference between uplink and downlink symbols, and from one cell to another. Therefore, the longer the downlink to uplink switching period, the fewer resources are lost in guard periods. On the other hand, the smaller it is, the higher the network's reactivity. The downlink to uplink switching periodicity must be dedicated to slices with heterogeneous constraints.

For the eMBB and mMTC slices, the DL/UL switching is long and set to 0.5ms. There is a 1 slot (for eMBB) and 0.5 slot (for mMTC) of downlink data followed by a guard period and then by uplink symbols on the next slot/half slot. This uplink to downlink switching assumes the uplink and downlink traffics are balanced for both eMBB and mMTC as there are almost the same amount of resources in uplink and downlink. Between eMBB and uRLLC BWPs, we propose to add a guard band as the DL/UL switching is not the same. This guard band prevents downlink symbols of the two BWPs from interfering with the uplink symbols of the other. If the UL/DL switching configuration

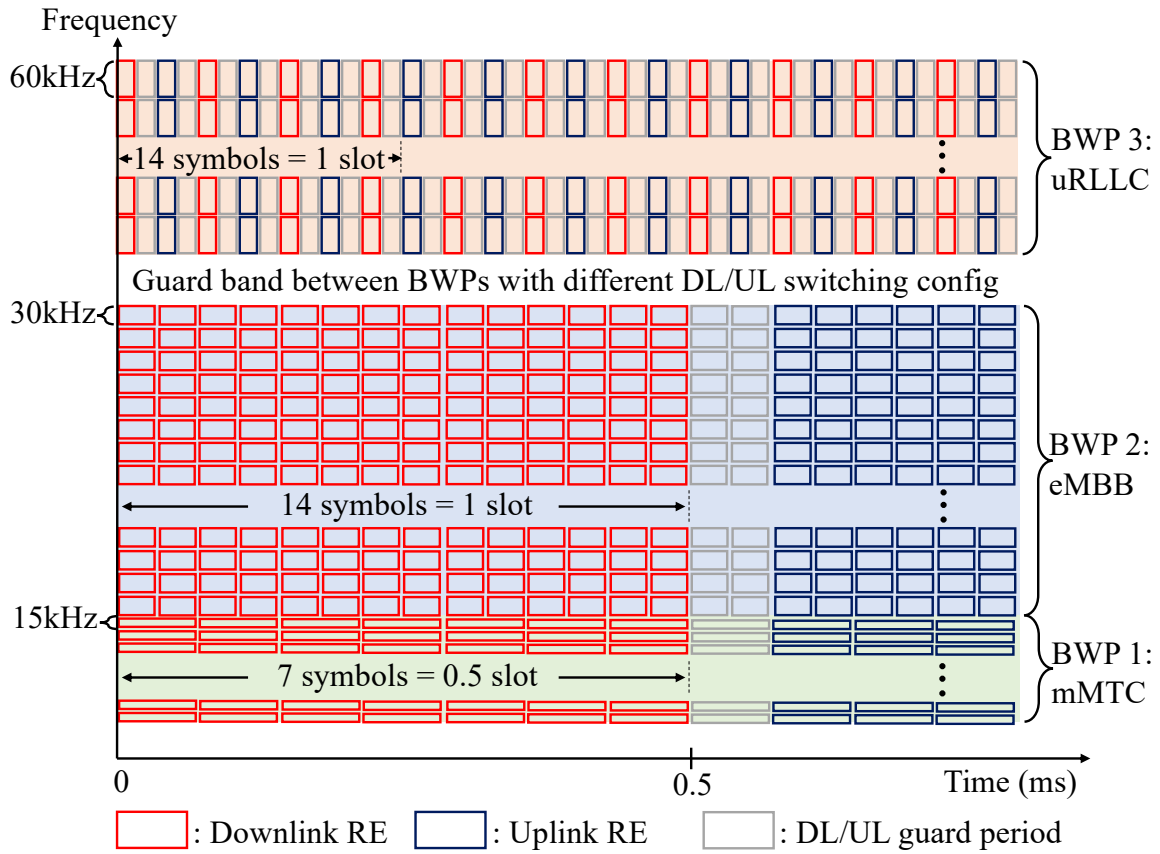


Figure 6.2: 5G radio frame structure from chapter 1

is not the same between eMBB and mMTC, another guard band should be added between the two BWPs. For example, if the mMTC slice is used to connect sensors that feedback data, more uplink resources are needed, so the balance will not be the same as the eMBB slice. For the uRLLC slice, the DL/UL switching is as quick as possible as it is done at the symbol level. One downlink symbol is followed by an uplink symbol, with a guard symbol between them. The guard symbol prevents downlink symbols spread delay from interfering with uplink symbols. The use of BWPs eases the DL/UL switching differentiation between the slices.

The last configuration aspect to be investigated in this section is resource allocation. When there is data to exchange between the UE and the gNodeB, the gNodeB allocates resources to the UE for reception or transmission. This is done at the MAC layer by the scheduling algorithm. Depending on the direction (uplink or downlink), the scheduling process is not precisely the same. For downlink data, the gNodeB transmits resource allocation to the UE using the PDCCH. On the other hand, for uplink transmission, the UE sends a Scheduling Request (SR) to the gNodeB using the PUCCH and the gNodeB sends back the allocation via the PDCCH. The PDCCH and PUCCH configurations are given by the RRC layer.

As UEs are always listening for resource allocation in the PDCCH and could be trying to send SR in the PUCCH, the resources allocated to those two channels cannot be used for other communications when there is no scheduling information to transmit. Therefore, for eMBB and mMTC slices, the PDCCH and PUCCH must be configured with a long periodicity to optimize resources usage. A typical configuration is to have one PDCCH and PUCCH occasion per slot, or even less. However, the longer the periodicity is, the longer the transmission delay. Indeed, when a UE must transmit an uplink transport block, it waits for the first PUCCH occasion to transmit SR, and the gNodeB waits for the first PDCCH occasion to transmit the resource allocation, which implies a high communication overhead. Therefore, long periodicity configurations must be avoided for delay-constrained slices. The PDCCH and PUCCH should be present once a symbol to enable UEs and gNodeB to initiate communications at the finest possible scale.

6.6 Conclusion

In this chapter, a network slice has been defined as being a logical network that provides specific network capabilities and network characteristics. A slice instance was introduced as the resources associated with one slice, and finally, network slicing was defined as the ability of the 5G physical infrastructure to operate multiple logical networks by allocating resources. This work focused on the base station's perspective. It did not investigate the impact of network slicing on the CN and how UEs can move from one slice to another.

Furthermore, the different resources associated with network slices have been reviewed. Those resources are bandwidth parts at the radio interface level, RAN instances at the network level, CPUs and memory and the system level, algorithms and threads at the software level, and physical nodes and links at the transport network level. For each resource, we introduced technologies and techniques that can be used to share or dedicate resources to one or another slice.

Finally, we went through three configuration aspects of the radio interface and the physical layer that can be tuned to fit heterogeneous traffics requirements.

Chapter 7

Network slicing modeling

Contribution

In this chapter, we introduce a model for resource allocation at the RAN level. This model considers network slicing and various constraints like capacity, UE's density, latency, and reliability. The main contribution is to relate the network's reliability with the coverage quality. Furthermore, simplicial homology is used to study the coverage quality of a deployment and validate that the constraints of the different slices are respected. Finally, this model is applied to power optimization as it is fundamental to respect heterogeneous traffics constraints.

7.1 Introduction

As described in chapter 6, network slicing mainly relies on resource association and allocation. Many solutions have been proposed in the literature for slice-aware resource allocation, mainly from a data center perspective. The resources allocated are CPU, memory, and bandwidth. Allocating those resources can be done using classic linear optimization algorithms like in [62] and [63]. However, the previous methods do not take into account RAN specific constraints where coverage is of the utmost importance. As resource allocation relies on system modeling, this chapter investigates how network models can include slices. Beyond capacity and density, which can be easily modeled, latency and reliability are critical parameters for such models and cannot be trivially studied. Reliability is modeled as a multiple connectivity constraint. For that matter, simplicial homology is used for studying network's coverage and connectivity. It has already been successfully applied for energy-saving methods, like in [64].

Furthermore, transmission power optimization is a paramount concern. Indeed, a too low transmission power leads to coverage holes, and such decreases the network's reliability, and a too high transmission power could increase interference and decrease the network's capacity. Given that coverage constraints (implied by slices reliability requirements) are hardly linear by nature, classic optimization methods cannot be used to solve the power optimization problem. Specific heuristics have to be developed. This work provides a near-optimal heuristic for 5G slice-aware systems based on simulated annealing. It can be used to determine the required power budget for a set of slices deployed on a given physical network.

The first part of this chapter details the system model where 5G flexible numerology is used to fit slices latency and throughput requirements, the second one describes simplicial homology, the third one explains the method used for power optimization and the last part presents our simulations results and analysis.

Notation	Definition	Unit
C	Set of cells in the network	-
i or c	A cell	-
S	Set of slices	-
s	A slice	-
W_T	Total bandwidth	Hz
W_s	Bandwidth allocated to slice s	Hz
κ_s^i	Capacity offered by cell i to slice s	b/s
P_t^i	Transmission power of cell i	W
$P_r^i(d)$	Received power from cell i at distance d	W
I^i	Interference undergone by cell i	W
N_s	Noise for slice s	W
η_s^i	SINR undergone by slice s on cell i	dB
d_0	Reference distance for the Path Loss model	m
γ	Path Loss exponent	-
λ	Signal wavelength	m
D_{xy}	Distance between cell x and cell y	m
p_{xy}	Critical point where interference of cell y over cell x is the higher	-
d_{xy}	Distance between cell y and critical point p_{xy}	m
k_B	Boltzmann constant	J / K
T	Ambient air temperature	K
ρ_s	UEs density for slice s	UE / km^2
a_s	UEs activity for slice s	-
t_s	Per UE throughput required for slice s	b/s
l_s	Latency importance for slice s	-
r_s	Connectivity and reliability requirement for slice s	-
K	Maximum simplicial complex dimension	-
k	Simplicial complex dimension	-
σ	A simplicial complex	-
β_k	Betti number of dimension k	-
B	Set of Betti numbers for all dimensions	-
$\partial_k(\sigma)$	The boundary operator of dimension k applied to simplicial complex σ	-
BWP_s	BWP allocated to slice s	-
ω_s	SCS for slice s	Hz
N_s^{RB}	Number of RBs allocated to slice s	-
M	Number of temperature steps for simulated annealing process	-
R	Set of cells radius	-
L	Temperature step length for simulated annealing process	-
α	Cooling factor for simulated annealing process	-
T_0	Initial temperature for simulated annealing process	-
ΔP	Transmission power variation for simulated annealing process	-
G and G_l	Set of Betti numbers for simulated annealing process	-

Table 7.1: Notations used in chapter 7

7.2 System model

7.2.1 Cellular network

We consider a cellular network in which circles represent cells. The circles centers are base stations, and the circles radii represent the cells' coverage zone. The set of all the cells in the network is noted C.

We define the capacity κ_s^i offered by a cell i to a specific slice s with allocated bandwidth W_s and undergoing a Signal over Signal over Interference plus Noise Ratio (SINR) η_s^i as being the Shannon capacity:

$$\kappa_s^i = W_s \cdot \log_2(1 + \eta_s^i) \tag{7.1}$$

The capacity offered by a cell is expressed in bits per second and corresponds to the maximum capacity the cell could deliver to a specific slice at a given time.
We define the SINR by equation (7.2), where $P_r^i(d)$ is the received power from cell i at distance d, I^i the interference and N_s the noise.

$$\eta_s^i = \frac{P_r^i(d)}{I^i + N_s} \tag{7.2}$$

7.2.1.1 Received power

We consider the following channel model. The power received from a cell i is given by the Path Loss model, which is described in [65] section IV. The received power at distance d, $P_r^i(d)$, can be written:

$$P_r^i(d) = P_t^i \cdot \left(\frac{d_0}{d}\right)^{\gamma} \cdot \left(\frac{\lambda}{4\pi d_0}\right)^2 \tag{7.3}$$

where P_t^i is the transmission power of cell i, λ is the signal wavelength and d_0 is the reference distance, below which the Path Loss model is not realistic.

The Path Loss exponent, noted γ, is a parameter which reflects the propagation conditions.

7.2.1.2 Interference

Interference of cell y on cell x is computed based on the received power of cell y at a critical point p_{xy} represented in Figure 7.1 which is the point where interference of y on x is the higher.

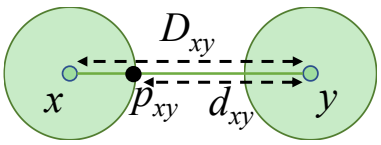


Figure 7.1: Representation of critical point p_{xy}

The distance d_{xy} between y and p_{xy} is given by:

$$d_{xy} = |D_{xy} - r_x|$$

Where r_x is the radius of cell x and D_{xy} the distance between x and y.

Interference I^i on cell i can finally be written as the sum over all the cells c inside C (excluding cell i) of the received power at distance d_{ic} using Equation (7.3). This gives an upper bound of interference as it comes to considering that all the base stations communicate over all their resource blocks simultaneously.

$$I^i = \sum_{c \in C \setminus \{i\}} P_t^c \cdot \left(\frac{d_0}{d_{ic}}\right)^{\gamma} \cdot \left(\frac{\lambda}{4\pi d_0}\right)^2 \tag{7.4}$$

7.2.1.3 Noise

The noise N_s is computed as the thermal noise, which depends on the Boltzmann constant k_B, the ambient air temperature T in Kelvin (generally taken at 290 Kelvin), and the allocated slice bandwidth W_s.

$$N_s = k_B.T.W_s$$

In general, $k_B \cdot T = 4.0039 \cdot 10^{-20}$W/Hz.

7.2.2 Flexible numerology and network slicing

7.2.2.1 Bandwidth parts:

5G NR flexible numerology enables an operator to split its bandwidth in different BWPs. All the BWPs have a different set of parameters that are BWP width (number of RBs) and BWP SCS. BWP width is related to the capacity, as described in section 7.2.1 and BWP SCS is related to the communication latency. Those two parameters can be tuned by the operator depending on the requirements of the UEs that will connect to the network through the allocated BWP. As described in chapter 6, it makes sense to consider that BWPs will be dedicated to each slice or service type.

7.2.2.2 Network slicing:

Consequently, a slice is allocated a BWP depending on its capacity and latency requirements. Therefore, we define slices as a set of parameters representing their capacity and latency requirements.

For capacity requirements, we define three parameters:

- UE density (noted ρ_s): This parameter represents the density of UEs that might connect to a base station. This parameter is the number of UEs per surface unit.
- Activity (noted a_s): This represents the time proportion a UE might be in connected mode.
- Per UE throughput (noted t_s): This represents the capacity required by a single UE while in connected mode.

To take into account latency requirements, we define one parameter:

- Latency importance (noted l_s): This parameter represents the latency requirement level, and is represented by 0,1 or 2. This is used to compute BWP SCS which will be equal to 15, 30 or 60kHz respectively.

Finally, as some network slices might require ultra reliable communications, we define a parameter to take this aspect into account:

- Connectivity (noted r_s): This represents the number of antennas to which each UE should be connected simultaneously. As the handover procedure can take a lot of time compared to the latency requirements, network reliability can be improved by increasing the number of antennas to which each UE is connected, avoiding handovers communication overhead. This parameter is a crucial reason for using simplicial homology, which enables us to compute coverage quality, including multiple network connectivity constraints.

In the evaluation framework described in section 7.5, we define three typical 5G network slices:

- Slice 1 (eMBB): Legacy mobile communications for making calls, surfing on the web and video streaming.
- Slice 2 (uRLLC): Industry 4.0 for connecting critical actuators and sensors.
- Slice 3 (mMTC): Smart cities that collect data from ultra dense sensors.

7.3 Simplicial Homology

The need for a method for analyzing network coverage and computing network connectivity led us to use simplicial homology, which is a mathematical tool from algebraic topology. We model a cellular network as a set of cells represented by a vertex (the base station) and a radius r. We consider that the intersection between two vertices is not empty if the coverage zones of the corresponding cells intersect each other, as depicted in Figure 7.3.

The first step for studying a cellular network is to get its simplicial complex representation. A simplicial complex is a combination of vertices (i.e., a combination of base stations) that intersect each other. The number of elements in the combination gives the dimension of the simplicial complex. A k dimension simplicial complex is called a k-simplex. Thus, a vertex is a 0-simplex, an edge is a 1-simplex, a triangle a 2-simplex, a tetrahedron a 3-simplex, etc.

There are two main simplicial complexes for network topology representation: the Cěch complex and the Rips complex. The real network representation is given by the Cěch complex, but its computation complexity is high compared to the Rips complex. Indeed, for each dimension k, Cěch complex ensures there is at least one point in the topology which is covered by $(k+1)$ cells. On the other side, Rips complex in dimension k is a combination of $(k+1)$ $(k-1)$-simplexes. Therefore, to build the Cěch complex, it is required to loop over all the points of the topology, whereas the Rips complex is made by combining lower-order simplexes, which has a much lower algorithmic complexity.

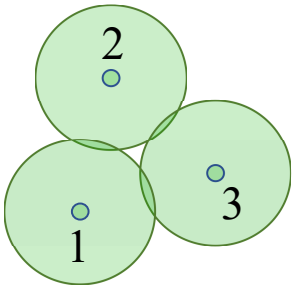


Figure 7.2: Rips complex error representation

However, Rips complexes are not fully accurate as they can capture simplexes that do not exist. For the topology represented in Figure 7.2, for both the Cěch and Rips complex, the 0-simplexes are $\{1, 2, 3\}$ and the 1-simplexes are $\{[1, 2], [1, 3], [2, 3]\}$. For 2-simplexes, following the definition of Rips complex, 2-simplexes are a combination of 3 1-simplexes which intersect each other. Therefore, using the Rips complex, there is one 2-simplex in this topology, which is $\{[1, 2, 3]\}$. However, as it can be seen, there is no point in the topology connected to the three vertices, and there is, therefore, no 2-simplex when using Cěch complex. When looking at the topology, the Cěch complex is accurate as there is one coverage hole. The Rips complex has captured a non-existing 2-simplex which implies that it was not able to detect the coverage hole in the middle of the topology.

Nevertheless, it has been shown by [66] that between 0% and 11% of the time, the Rips complex captures k-simplexes that do not exist compared to the Cěch complex. Therefore, regarding the error rate and the complexity gain, the Rips complex provides a good engineering approximation and is used in this chapter.

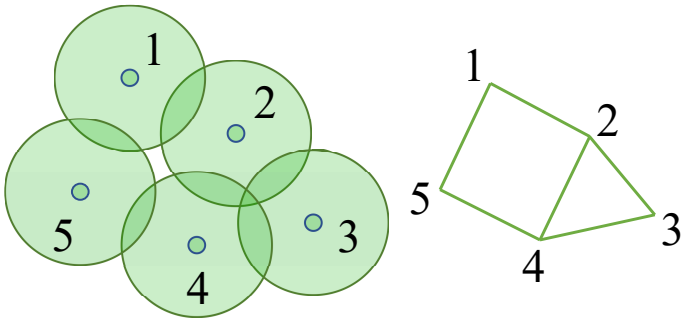


Figure 7.3: Cell deployment example and corresponding two-by-two cells intersections. Points represent base stations and circles represent cells coverage.

In the example shown in Figure 7.3, using Rips complexes, the 0-simplexes are the vertices $\{1, 2, 3, 4, 5\}$, 1-simplexes are $\{[1, 2], [2, 3], [3, 4], [2, 4], [4, 5], [1, 5]\}$ and there is one 2-simplex: $\{[2, 3, 4]\}$.

Betti numbers (noted β_0, β_1,...,β_k) are the dimension of each homology group whose geometric meaning is the number of k-dimension holes in the network. Thus, β_0 represents the number of

connected components in the network, β_1 represents the number of coverage holes, β_2 represents the number of zones where there is no 2-connectivity etc.

Betti numbers are computed based on equation:

$$\beta_k = rank(ker(\partial_k)) - rank(Im(\partial_{k+1}))$$

Where ∂_k is the boundary operator of dimension k applied to simplex σ which is defined by:

$$\partial_k(\sigma) = \sum_{i=0}^{k} (-1)^i \cdot (\sigma_0, ..., \sigma_{i-1}, \sigma_{i+1}, ..., \sigma_{k-1})$$

The rank nullity theorem states that:

$$rank(ker(\partial_k)) + rank(Im(\partial_k)) = dim(\partial_k)$$

Betti numbers can therefore be computed by:

$$\beta_k = dim(\partial_k) - rank(Im(\partial_k)) - rank(Im(\partial_{k+1}))$$

The rank of $Im(\partial_k)$ is equal to the rank of the matrix (noted H_k) of the elements of $Im(\partial_k)$ in the base defined by the $(k-1)$-simplexes.

With the above example, β_1 can be computed with the following steps:

$$\partial_2([2,3,4]) = (-1) \cdot [3,4] + (1) \cdot [2,4] + (-1) \cdot [2,3]$$

Therefore:

$$\partial_2([2,3,4]) = 0 \cdot [1,2] + (-1) \cdot [2,3] + (-1) \cdot [3,4] + 1 \cdot [2,4] + 0 \cdot [4,5] + 0 \cdot [1,5]$$

And:

$$H_2 = \begin{pmatrix} 0 \\ -1 \\ -1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Similarly:

$$H_1 = \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & -1 \\ 1 & -1 & 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Finally:

$$\beta_1 = dim(\partial_1) - rank(H_1) - rank(H_2) = 1$$

In the example shown in Figure 7.3, β_0 and β_1 equal one: there is one related component and one coverage hole.

7.4 Energy saving algorithm

In this section, the model presented above is applied to the power optimization problem. Power optimization is critical for two main reasons. First, it enables to reduce the network's transmission power budget. Foremost, it enables to mitigate interference while maintaining a good coverage quality.

7.4.1 Optimization problem

The value to be minimized is the total network transmission power, and the constraint is network coverage. The Betti numbers give the coverage quality.

The global optimization problem is defined as follows:

$$\min \quad \sum_{c \in C} P_t^c \tag{7.5a}$$

$$\text{subject to} \quad \beta_k \leq \beta_k^{init}, \forall k \in [0, K-1] \tag{7.5b}$$

with P_t^c the transmission power of cell c, K the maximum simplicial complexes dimension and β_k^{init} the initial k^{th} Betti number. The set of initial Betti numbers β_k^{init} is noted B^{init}. Slices do not appear here as there are taken into account while computing cells radius (in section 7.4.2.2), which is not directly the optimization problem.

7.4.2 Sub-optimal heuristic

As a power increase might imply new interference and thus decrease coverage quality, the optimization problem described in section 7.4.1 is not convex and cannot be solved with classic linear optimization methods. Therefore, the metaheuristic we use is the simulated annealing method which is a probabilistic algorithm that gives a sub-optimal solution. We use this method as it has already shown promising results in different similar projects such as [64] and [67].

Algorithm 9 gives an overview of the simulated annealing algorithm. Cells are initialized with an transmission power which is an algorithm input. At each algorithm iteration, a cell is randomly chosen, and we try either to reduce or increase its transmission power. An transmission power decrease can only be accepted if there is no lack of k-connectivity in the ending state. A power increase will be accepted with a probability given by a defined temperature cooled over iterations. The parameters of this algorithm are M (number of temperature steps), L (temperature step length), α (cooling factor), T_0 (initial temperature) and ΔP (transmission power variation). The statement $if\ B^l > B^{init}$ (line 25) has to be understood as: *if one new Betti number from* 0 *to* K *(*β_0*,...,*β_K*) is greater than initial Betti numbers*. It corresponds to the optimization problem issued in section 7.4.1. K is the maximum connectivity dimension we check and is equal to the maximum slices connectivity requirement (defined in section 7.2.2.2). C and P_t^c are the set of cells and the transmission power of cell c.

The different functions used in Algorithm 9 are explained below:

7.4.2.1 BWP allocation (*allocate_bwps* line 2)

Each slice s is allocated a BWP noted BWP_s. The SCS, ω_s, is allocated depending on slice latency importance. The number of RBs (N_s^{RB}) is allocated proportionally to the total available bandwidth W_T with the following equation. t_s, a_s and ρ_s are respectively the per UE throughput, activity and density for slice s.

$$N_s^{RB} = \left\lfloor \frac{1}{12.\omega_s} \cdot \frac{t_s \cdot a_s \cdot \rho_s}{\sum_{x \in S} t_x \cdot a_x \cdot \rho_x} \cdot W_T \right\rfloor$$

7.4.2.2 Computing cells radii (*compute_radius* lines 4 and 22)

The process for computing the cells radii is the following (made cell by cell):

1. For each cell, we initialize the current radius at d_0, which is the minimal cell's radius.
2. Then we compute the maximum radius of the cell using a minimum reception threshold. This is made using the Path Loss model by reversing equation (7.3). The distance d (maximum radius of the cell) is expressed in function of the received power P_r^i (which is set to the minimum reception threshold). This distance corresponds to the maximum distance from the base station for which a UE could decode the radio signal. Beyond this distance, no UE could connect the

Algorithm 9 Simulated annealing

▷ Allocate BWP to slices
$allocate\_bwps()$
▷ Compute initial cells radius
$R = compute\_radius(C)$
▷ Compute initial network's connectivity
$B^{init} = compute\_connectivity(C, R)$
▷ Loop over all temperature steps
for $m \in [0, M]$ **do**
    ▷ Compute current temperature and probability to accept a power increase
    $T_m = T_0.\alpha^m$
    $p = e^{\frac{-\Delta P}{T_m}}$
    ▷ Iterate over one temperature step
    for $l \in L$ **do**
        ▷ Randomly select a cell and a direction (power increase or decrease)
        $c = rand(C)$
        $s = rand(\{-1, 1\})$
        ▷ If power decrease
        if $s = -1$ **then**
            ▷ Compute new cell's transmission power
            $P_t^c = P_t^c - \Delta P$
            ▷ Compute new cells radius and network's connectivity
            $R = compute\_radius(C)$
            $B^l = compute\_connectivity(C, R)$
            ▷ If the connectivity is worse than the initial one
            if $B^l > B^{init}$ **then**
                ▷ Cancel power decrease
                $P_t^c = P_t^c + \Delta P$
            end if
        else
            ▷ If power increase
            if $s = 1$ **then**
                ▷ Increase cell's transmission power with a probability of p
                $v = rand([0, 1])$
                if $v < p$ **then**
                    $P_t^c = P_t^c + \Delta P$
                end if
            end if
        end if
    end for
end for

base station, so this distance is the maximum cell's radius. The circle defined by the base station and the maximum cell's radius defines the maximum coverage zone.

3. We create sub-circles inside the maximum coverage zone and study the cell's capacity for each sub-circle. The number of sub-circles to be tested inside the maximum coverage zone is an input parameter. As we observed that the cell's capacity decreases exponentially over distance, we create a geometric series to increase the sub-circles radius exponentially so that capacity over sub-circles is expected to decrease linearly.

4. For each sub-circle:

   (a) We compute the received power depending on the cell's transmission power following Equation (7.3). The distance d is set to the current sub-circle radius. Interference is also calculated by computing the received power from all the other cells of the network. It is computed following Equation (7.4), by considering that the cell's radius is the sub-circle radius.

   (b) We loop over each slice, and for each slice, we compute the required capacity (given by slices parameters, see section 7.2.2.2) and the offered capacity (described in section 7.2.1) using W_s and undergone SINR.

   (c)
   - If, for each slice, the offered capacity is greater than the required capacity, the sub-circle can serve the different slices, so the current radius is set to the current sub-circle radius, and the algorithm continues.
   - Otherwise, the sub-circle cannot serve the different slices, and the algorithm ends. The cell's radius is set to the last validated sub-circle radius.

7.4.2.3 Computing connectivity (*compute_connectivity* lines 6 and 23)

This function computes simplicial complexes based on the topology and return the derived Betti numbers as explained in section 7.3.

7.5 Simulations and results

We have investigated four different scenarios. The first three scenarios are based on single slice simulations. eMBB and mMTC simulations have been done for a connectivity requirement equal to one and uRLLC simulation for a connectivity requirement equal to three. The last scenario integrates all the previously defined slices with a connectivity requirement of three. The network is made of a set of antennas randomly deployed inside a finite square area according to a Poisson Point Process (PPP). Table 7.2 gives the deployment parameters of the cells used in the different scenarios. Table 7.3 describes the cells' radio parameters associated with different slices. Ten thousand simulations are performed for each scenario to compare power budget and transmission energy efficiency. The power budget is defined as the total power consumed by cells of a given deployment. The code and parameters used for these simulations are available online (https://github.com/adejavel/5GSliceAwareEnergySaving).

Table 7.4 gives the different slices parameters used in the four simulation scenarios: ρ_s is the UEs density, a_s is the UE activity factor, t_s is the required per UE throughput, l_s is the slice latency importance, r_s is the connectivity requirement and the last column is the total throughput per squared kilometer required by the slice. Slice 1 capacity requirements are low compared to what has been announced in 5G as massive MIMO and beam-forming are not considered. Simulations have been performed for the single slice scenarios in their specific frequency bands. In the last scenario, the three slices share the same frequency band in a context where bandwidth is limited.

The different results obtained during the simulations are shown in Table 7.5. The mean power budget is expressed per squared kilometers, and energy efficiency represents the power budget per transmitted bit. Figures 7.4a, 7.4b, 7.4c and 7.4d are histograms representing the overall distribution of the antennas transmission power (expressed in Watts) at the end of the algorithm for all the

Slice	**Vertical**	**Square width (m)**	**Cells density (cells/km^2)**
Slice 1	eMBB	500	50
Slice 2	uRLLC	300	500
Slice 3	mMTC	500	30
All slices	All	300	500

Table 7.2: Deployment parameters

Slice	**Initial power**	ΔP	**Band**	**Bandwidth**	γ
Slice 1	40 W	0.5 W	3.5GHz	400 MHz	3
Slice 2	0.5 W	5 mW	26GHz	500 MHz	6
Slice 3	1 W	10 mW	700 MHz	10 MHz	3
All slices	10 W	0.1 W	3.5 GHz	400 MHz	4

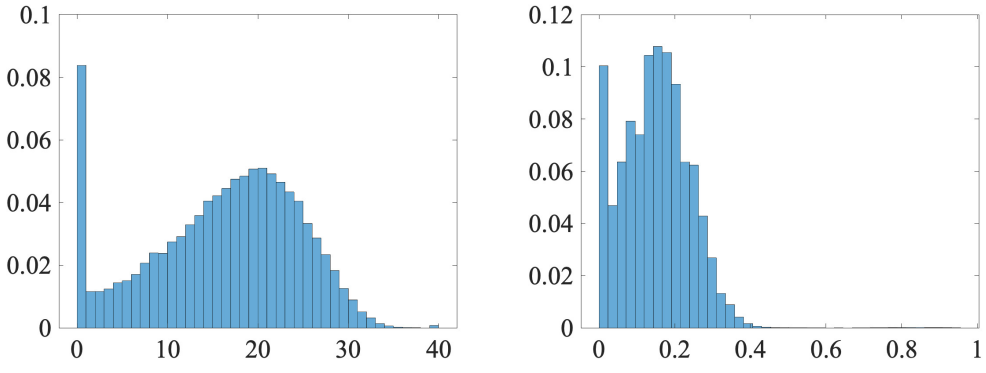
Table 7.3: Simulations parameters

Slice s	ρ_s **(UE/km^2)**	a_s	t_s **(b/s)**	l_s	r_s	**Total throughput (b/s)**
Slice 1	10^2	0.1	$100 \cdot 10^6$	0	1	10^9
Slice 2	$2 \cdot 10^4$	0.05	10^3	2	3	10^6
Slice 3	10^5	0.05	10^3	0	1	$5 \cdot 10^6$

Table 7.4: Slices parameters

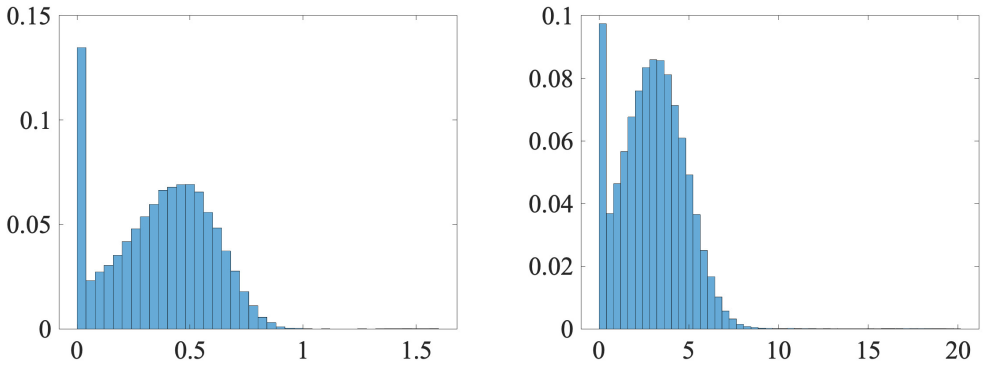
Slice	**Power budget (W/km^2)**	**Energy efficiency (W/b)**
Slice 1	777	$7.77 \cdot 10^{-7}$
Slice 2	70	$7 \cdot 10^{-5}$
Slice 3	10.8	$2.16 \cdot 10^{-6}$
All slices	1425	$1.41 \cdot 10^{-6}$

Table 7.5: Simulations results



(a) Slice 1: eMBB (W)

(b) Slice 2: uRLLC (W)



(c) Slice 3: mMTC (W)

(d) All slices (W)

Figure 7.4: Simulations results: histograms representing cells power distribution

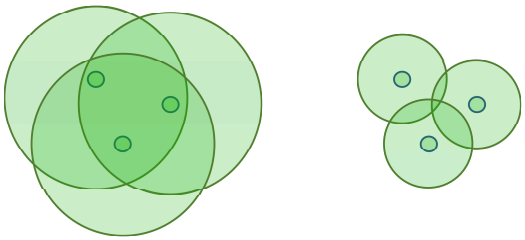


Figure 7.5: Cells cluster illustration

simulation cases. It can be observed that all the histograms have the same shape: an initial peak followed by a second one around the median.

The initial peak is due to the clustering inherent to PPP. This process often creates clusters of close cells. Inside these clusters, base stations can reduce their transmission power to the minimum without impacting the number of coverage holes and connectivity. This is illustrated in Figure 7.5 where Betti numbers are equals in both the left and right deployments for different transmission power.

The results obtained with those simulations can be analyzed to understand the behavior of the 5G system in one or another deployment scheme. However, the results for the different slices cannot be strictly compared as the simulation parameters are not the same. Therefore, it is impossible to directly compare the impact of the slices requirements on the power budget. Nevertheless, those simulations enable to compare four cases. In the first three cases, the network is dedicated to one slice, and the associated parameters are tuned to best meet the slices requirements. In the last case, all the slices run on a single network whose parameters provide a balance between the requirements of the three slices.

What can be deduced from the results is that the power budget required for eMBB slice is the highest compared to the mMTC and uRLLC ones. This can be expected as the noise is proportional to the bandwidth from Equation 7.2.1.3. Given that the capacity of a network depends on the cell's

bandwidth, the highest the capacity requirement, the highest the required bandwidth, and the highest the noise. To compensate for the noise, the transmission power has to be increased to meet this slice's high capacity requirement.

Furthermore, it can be observed that the slice with the worst energy efficiency is uRLLC. It seems normal as the energy efficiency gives the energy required per unit of transmitted data. The capacity requirement is low for the uRLLC, and a high power budget is mainly required to provide the best coverage to UEs with multiple connectivity constraints. Therefore, it is normal to have a bad energy efficiency for this slice as the multiple connectivity constraint considerably impacts the power budget even if the amount of transmitted data is low.

Finally, for the last simulation, where all the slices are deployed on a single network, the power budget explodes. This can be explained as the high capacity requirement of eMBB has an effect on noise which is antagonist with the high connectivity and coverage requirement of uRLLC, which is compensated by increasing the transmission power of the different cells. However, the energy efficiency is of the same order as that of the eMBB and mMTC (and not as high as for uRLLC). It can be explained by the fact that, unlike for the uRLLC slice, where the power budget is mainly used to increase the connectivity, the power budget here is used both to increase the coverage and provide high capacity to UEs.

7.6 Conclusion

In this chapter, we introduced a model for network slicing in 5G. Beyond classic parameters such as capacity, reliability has been introduced as a multiple connectivity constraint, and simplicial homology has been used for computing coverage's quality.

Furthermore, a solution for determining the per-slice power budget has been proposed. It takes into account slices capacity, latency, and reliability constraints. Simplicial homology was introduced as a way to model reliability requirement. A simulated annealing method was used to reduce RAN power budget. Extensive simulations has been realized for three different types of slices : eMBB, uRLLC and mMTC. Simulations have shown that reliability constraints imply a huge loss of energy efficiency and that deploying slices independently with a dedicated radio configuration preserves power budget.

Chapter 8

Implementation of a 5G probe

Contribution

The contribution of this chapter is the implementation of a 5G probe that can be used for supervising the network. This probe is derived from the *free5GRAN* project. It is a standalone probe as it does not require information from the network. It can be used to extract data from a 5G SA cell. The data extracted by the probe is the PHY layer DCI and data transport blocks (DL/UL-SCH). The MAC layer SDUs and control elements can be parsed from the transport blocks. This probe can be used by supervision tools to monitor the network and increase the network's reliability.

8.1 Introduction

The major evolution of the 5G system is the huge diversity of the terminals that connect the network. It has been detailed in chapter 1. The diversity of the terminals has two impacts on the network infrastructure that network slicing must manage. First, given that the different terminals do not connect to the same data network, the infrastructure must implement tunnels that transport communications between terminals and data networks. Furthermore, the various terminals will have highly heterogeneous user traffic, which implies that the network must support many different QoS profiles.

Among those heterogeneous user traffics, some have strong constraints regarding reliability and network control. The network control constraint requires that the QoS that can be offered to the UE is strictly mastered by the network at every moment. This is where a major challenge is raised. Indeed, the purpose of 5G networks is to support those diverse user traffics on top of a single physical infrastructure. However, complete network control cannot be ensured when the infrastructure is shared between critical and non-critical slices. Some techniques can be used to increase the reliability and control of the network when the infrastructure is shared by dedicating resources to slices (investigated in chapter 6). Nevertheless, even if the infrastructure can be optimized to provide the highest possible network control, it cannot reach full control and reliability. The only way to have complete control over the network for critical user traffics is to deploy a dedicated infrastructure for the associated terminals, but this is not the vision of the 5G system.

Therefore, supervision tools are required to provide the highest possible network control over an infrastructure shared between critical and non-critical slices. Those tools are expected to add a second level of reliability and control on top of the first level provided by the network itself. To supervise the network, they must be fed with the highest possible volume of data in order to reach high accuracy. The infrastructure can provide the data using network APIs. However, for highly critical services and especially for services that might be exposed to security issues, it is worth that the supervision tools also rely on data from outside the network. Indeed, in case of a security attack over the network, the data provided by the APIs might be corrupted, and the supervision tools might not be able to detect the attack. On the other side, using data from outside the network guarantees that the data is not corrupted even in case of a security attack.

In this chapter, the work done for the PHY layer is derived to implement a standalone 5G probe that can extract all the downlink and uplink communications of all the UEs of a cell. The purpose of

this probe is to provide supervision tools with data from outside the network to increase the network's reliability and control. The first section details how the probe has been implemented, and the second section investigates what is the data extracted by the probe and how supervision tools can use it to provide network control.

8.2 Implementation and software architecture

After implementing all the PHY functions introduced in chapters 2 and 3, the *free5GRAN* code architecture needs to be adapted in order to implement a probe that can decode the downlink and uplink communications of all the UEs. For downlink communications, the probe is derived from the receiver's side of the UE, and for the uplink direction, it is derived from the receiver's side of the gNodeB. Furthermore, it is required for the probe to decode all the communications of all the UEs, whereas the PHY layer decodes transmissions on a per UE basis. This section details the architecture of the probe and some implementation specificity.

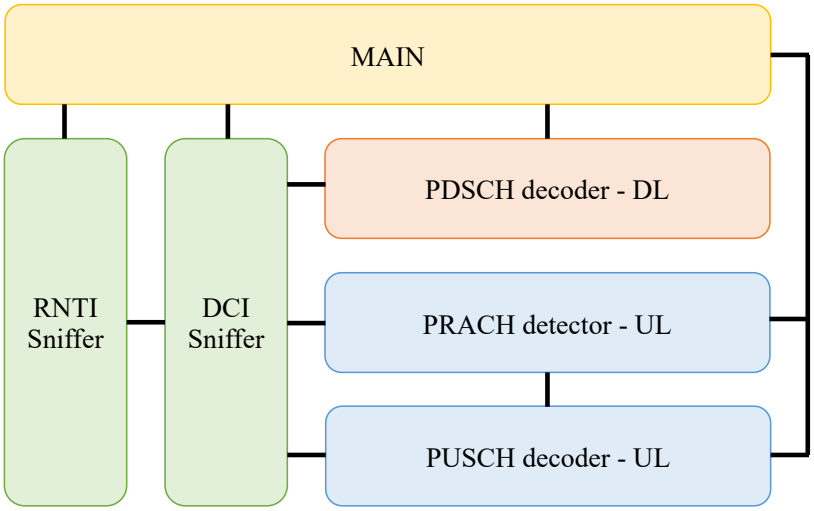


Figure 8.1: Overall probe's architecture

Figure 8.1 represents the architecture of the probe. In this section, all the probe components are reviewed to explain how it is implemented and expose the architecture.

8.2.1 Main

The main thread is responsible for managing all the different components of the probe. Its main responsibility is to perform the initial cell's synchronization. It includes the PSS and SSS search, the MIB decoding and the SIB1 decoding. Once the probe is fully synchronized with the cell, the main thread can start the other components to start listening the transmissions.

8.2.2 RNTI sniffer

The second component is the RNTI sniffer. The purpose of this component is that the first step towards downlink and uplink transport block decoding is DCI decoding. The issue is that the CRC of all the DCI payloads are masked with the UE's RNTI to which it is intended. Therefore, decoding the DCI payloads is impossible without having a list of all the RNTIs being used on the network. Furthermore, this component is responsible for decoding the RRC Setup messages. This message is critical as it contains all the RRC configuration used for exchanges between the UE and the gNodeB.

The RNTI sniffer decodes two transmissions:

- The first one is the RAR. Indeed, as explained in section 3.4, the RAR is the response of the gNodeB to the PRACH transmission of a UE. The RAR contains different information, among

which TC-RNTI is the main one used here. The TC-RNTI is the RNTI used by the UE for completing the RA procedure. If the procedure succeeds, the TC-RNTI becomes the C-RNTI, which is the actual RNTI used for traffic communications in uplink and downlink.

- The second one is the RRC Setup message, which is the last message of the RA procedure. This message is decoded for two reasons. The first one is that it enables to confirm that the TC-RNTI decoded while decoding the RAR is used and actually corresponds to a UE. The second one is that it contains the RRC configuration used for communications between the UE and the gNodeB. In the current version of the probe, it is considered that the RRC configuration is the same for all the UEs, so the message is decoded once to complete the probe configuration, and then it is not used anymore. In a future version of the probe, the RRC configuration could be stored for each UE in order to support per UE RRC configuration.

To decode those two transmissions, the first step is to extract the search space and CORESET used for RA. It can be found in SIB1. Those two objects give the configuration used for the transmission of the DCI during the RA process.

When the search space and CORESET are determined, the RNTI sniffer can start searching for DCI transmissions. It performs blind search as explained in section 3.2.5.2. For the RAR, the RNTI used for CRC masking is RA-RNTI which can be computed based on the PRACH occasion that has been used by the UE. As the probe does not know what is the occasion used by UEs, it computes the set of all the RA-RNTI corresponding to the possible PRACH occasions. For RRC Setup transmission, the RNTI used for CRC masking is the TC-RNTI transmitted in the RAR message. The probe builds a set of possible RNTIs that are the possible RA-RNTI and the TC-RNTI for which RRC Setup has not been transmitted. DCI blind search is done for all the PDCCH candidates and CRC validation is tried for each RNTI.

Once the probe has extracted and decoded one DCI in the RA search space, it determines if it is an allocation for RAR or for RRC Setup based on the RNTI type (RA-RNTI or TC-RNTI). In both cases, it extracts and decodes the corresponding PDSCH and DL-SCH transport block. Finally, for RAR, it parses the MAC PDU and for RRC Setup, it performs unaligned PER decoding to recover the ASN1 RRC Setup message.

The last step is to extract the TC-RNTI in the RAR message and to validate that it is actually used by decoding the RRC Setup. This enables to build a set of C-RNTIs used in the cell. When the first UE connects the cell, the RRC Setup message is used by the probe to determine the cell's detailed RRC configuration.

8.2.3 DCI sniffer

Once the RNTIs used on the cell have been determined by the probe, the probe can start to search for DCI messages. The DCI messages are the first information that has to be extracted as they contain the resource allocation for UL-SCH and DL-SCH. The two DCI formats the probe is looking for are DCI Format 1_1 and 0_1 with C-RNTI.

The first step is also to determine the search space where the PDCCH transmissions are located. This information can be found either in the SIB1 or RRC Setup message. Once the search space has been determined, the size of the two DCI payloads must be determined to perform a blind search.

When all the required information have been determined, the DCI sniffer can start to continuously blind search for PDCCH candidates. One successful candidate is a DCI message which size is one of the DCI payloads sizes and which CRC is validated after de-masking with one of the C-RNTI used in the cell (determined in section 8.2.2).

Given that blind searching for DCI transmissions for each UE has a high processing complexity, the DCI sniffer component is multi-threaded. The main thread will run one DCI sniffer thread for each downlink slot. Each DCI sniffer element is responsible for blind searching DCI payloads in one slot.

The RNTI and DCI sniffers give the first version of the probe. All the information for physical layer resource assignment and usage is known at this level.

Figure 8.2 represents the overall procedure for the RNTI and DCI sniffer.

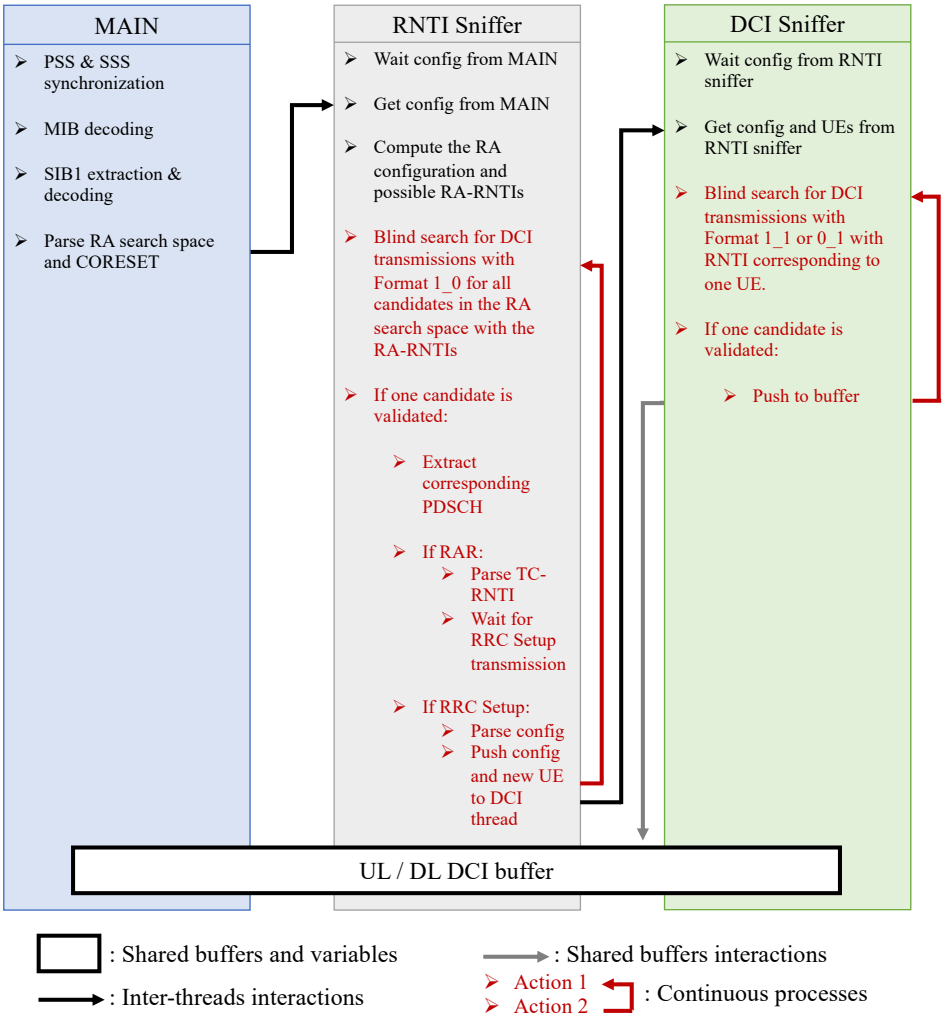


Figure 8.2: Overall procedure for RNTI and DCI sniffer

8.2.4 DL-SCH decoding

Once the probe decodes the DCI, the corresponding PDSCH / DL-SCH transmissions can be extracted and decoded. Figure 8.3 represents the processing steps for the decoding of the PDSCH / DL-SCH transport blocks. The input of this component is the OFDM grid together with the resource allocation. The resource allocation information contains the following information:

- Symbols and subcarriers where PDSCH can be found. It enables the probe to extract the IQ samples corresponding to PDSCH and DMRS, to perform channel estimation and equalization.
- It also contains information about redundancy version, modulation scheme, and code rate. The modulation scheme is used for modulation de-mapping for IQ samples to bit transformation, the redundancy version is used for rate recovering, and the code rate is used for LDPC decoding.
- The last important information is the HARQ process. After DL-SCH decoding, if the CRC is not validated, the current transport block is not dropped but kept in a buffer. The UE (and therefore the probe) waits for a second transmission of the transport block with another redundancy version. The UE (and the probe) combines the previous buffer with the new transmission of the transport block and tries the LDPC decoding and CRC validation again. This process is done in parallel for multiple transport blocks, and the HARQ process identifies the process to which the transmission belongs.

Given that LDPC decoding is highly consuming and that multiple transmissions can be done in parallel (for each HARQ process), the PDSCH decoder is multithreaded. One occurrence of this function is responsible for one HARQ process.

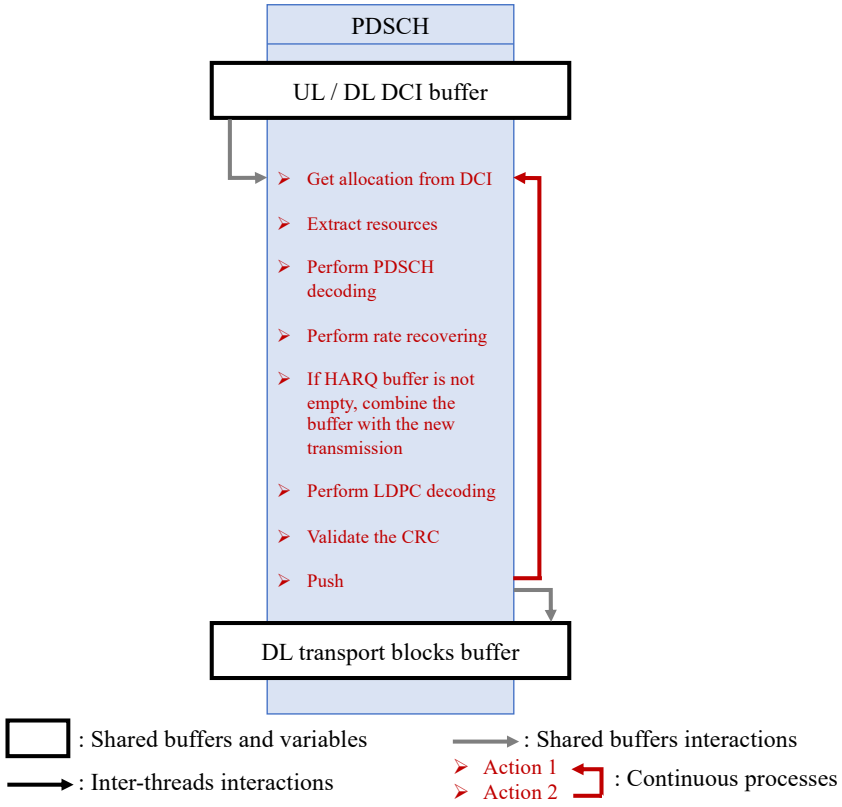


Figure 8.3: Procedure for PDSCH decoding

8.2.5 PRACH detection

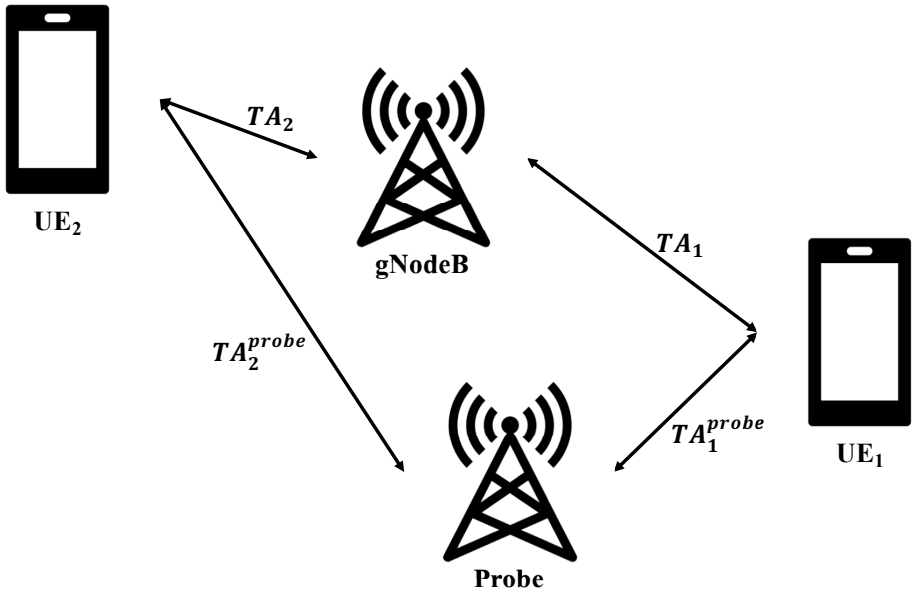


Figure 8.4: UE to probe TA computation

Given that the probe is not at the same location as the gNodeB, the uplink synchronization must be done on a per UE basis. Indeed, within a cell, the uplink synchronization is made with regard to the gNodeB: the PRACH transmission enables the gNodeB to compute the TA, and the UE transmits all the transmissions in advance, with a TA offset. With this kind of synchronization, all the uplink transmissions of all the UEs are synchronized with the cell when they arrive at the gNodeB. In Figure 8.4, as UE_1 sends its transmissions with TA_1 offset and UE_2 with TA_2, the uplink transmissions are synchronized at the gNodeB. On the opposite, as the probe is not in the same location as the gNodeB,

the TA with regard to the probe (noted TA_x^{probe}, where x identifies a UE) is different, which means that the uplink transmissions will not be synchronized at the probe. Therefore, the probe has to compute the synchronization offset (noted Δ_x^{probe}), which is the synchronization offset of the UEs with regard to the probe. This offset is computed with Equation:

$$\Delta_x^{probe} = TA_x^{probe} - TA_x$$

The uplink synchronization of the probe follows the steps:

1. First, UE x transmits PRACH signal towards the gNodeB.
2. The gNodeB answers with the RAR which contains the TA. The probe extracts and decodes the RAR and is able to determine the UE's TA with regard to the cell (noted TA_x is Figure 8.4).
3. As the probe implements the PRACH detection, it will also detect the UE's PRACH transmission and estimate the TA of the UE with regard to the probe, noted TA_x^{probe} in Figure 8.4.
4. The final offset Δ_x^{probe} is computed.
5. For extracting uplink resources, the probe extracts the time domain signal based on the UE's offset, performs OFDM demodulation and decodes the PUSCH.

8.2.6 PUSCH detection

Once the probe is synchronized in uplink with the UE, it can use the information extracted from the DCI sniffer to extract and decode the PUSCH. Here is the overall process:

- The probe gets the uplink DCI payload from the DCI buffer. As for PDSCH (section 8.2.4), the important information is the resource allocation, the transmission configuration and the HARQ process.
- It extracts the corresponding time-domain signal and apply the uplink synchronization offset (Δ_x^{probe}) before performing OFDM demodulation.
- Once the frequency domain grid is recovered, the probe extracts the allocated resources.
- Finally, it performs PUSCH and UL-SCH decoding. In case of HARQ re-transmission, the current transmission must be combined with the previous transmission that happened with the same HARQ process.

As for PDSCH, given that LDPC decoding is highly consuming and that multiple transmissions can be done in parallel (for each HARQ process), the PUSCH decoder is multithreaded. One occurence of this functions is responsible for one HARQ process.

Figure 8.5 represents the process for extracting UL-SCH transport blocks and interactions between PRACH detector and PUSCH decoder.

8.3 Extracted data and supervision

The different components of the probe enable it to extract a significant amount of data. Supervision tools can use this data to increase the overall network control. This section introduces the different information extracted from the cell with the probe and the possible understanding of the network that can be derived from the data and leveraged by supervision tools.

The objective of this probe is to gather as much data as possible from the cell. Given that the raw data is extracted at the PHY layer, we try to parse and decode the most possible upper layers. Figure 8.6 represents the layers from which data can be extracted. The PHY data is the data actually

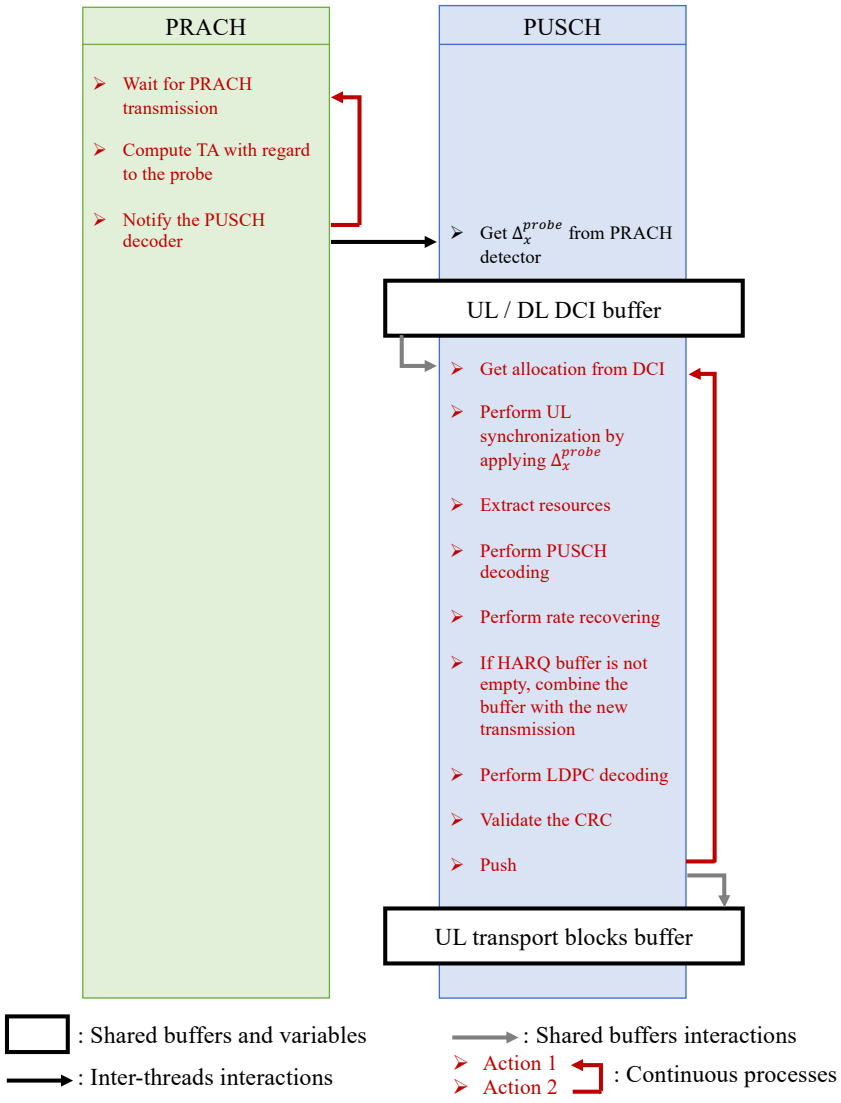


Figure 8.5: Overall procedure for uplink synchronization and decoding

extracted by the probe and does not need to be parsed. At the MAC layer, the data is accessible after parsing the control elements and SDUs. After the MAC layer, it becomes highly challenging. First, given that all the transmissions are ciphered at the PDCP level, no data can be extracted from the layers above the PDCP layer. Finally, the data is not ciphered at the RLC layer, but the format of the SDUs depends on the transmission mode (acknowledged, non-acknowledged or transparent). It is then required to implement an inference mechanism to determine the transmission mode used. With such a mechanism, the probe could parse the RLC SDUs and extract data from this layer.

Given that there is a huge complexity gap between the DCI extraction and corresponding uplink and downlink transport block decoding, the probe can be restricted to the DCI extraction when the computing resources are limited and extended to transport block decoding when the number of available resources is high. Therefore, the first part of this section focuses on the data accessible when only extracting the DCIs, and the second part investigates the data accessible after decoding the transport blocks.

8.3.1 DCI

The DCI extraction enables to recover all the DCI payloads for both the uplink and downlink directions and for all the UEs. As introduced in section 3.2, the DCI is carried by the PDCCH and is used by the gNodeB to grant resources to UEs. It can be downlink grant for transmitting PDSCH to UEs or uplink grant when UEs have to transfer PUSCH to the gNodeB. The DCI payload contains different

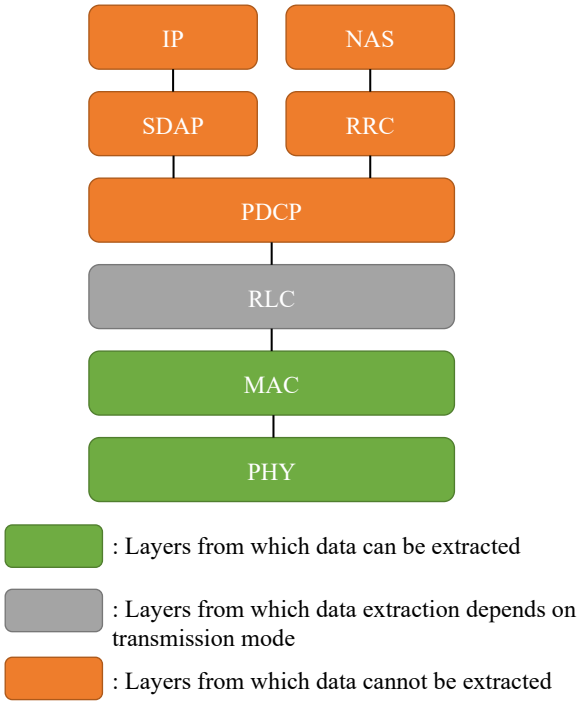


Figure 8.6: Layers in the stack from which data can be extracted

fields, as defined in TS38.212 section 7.3.1 [11]. Among the different fields, some are critical:

- Time and frequency domain resource allocation. Those two fields give the position of the data transmission in the OFDM grid. The time-domain allocation gives the first symbol and number of symbols in a slot where the PDSCH / PUSCH is located, and the frequency-domain allocation gives the first RB and the number of RBs in the cell's bandwidth. Therefore, those two fields define a rectangle within the OFDM grid where data is transmitted to or from one UE. The frequency-domain information is a single value called RIV and the number of RBs and first RB are recovered using an equation given in TS38.214 section 5.1.2.2.2 [24]. The time-domain information is an index in a list of possible time-domain allocations. This list is given in RRC messages like SIB1 or RRC Setup.

- MCS. This field gives the way PDSCH and PUSCH are transmitted. The modulation scheme gives the number of bits transmitted in a single IQ sample, whereas the code rate gives the proportion between the number of information bits and the number of code bits. Those two values are given by a single field which gives an index in Table 5.1.2.1-1 from TS38.214 [24].

- HARQ process and redundancy version. Those two fields give information about the upper MAC HARQ process. For one process (identified by the HARQ process field), the redundancy version equals zero for the first transmission of a transport block. A redundancy version higher than zero means that the transport block is being re-transmitted.

Supervision tools can use that information for different purposes:

- First, given that the number of IQ samples being transmitted in PDSCH / PUSCH is known by the resource allocation field and that the number of bits per IQ sample and code rate are known, the transport block size of every data transmission can be computed using the method given in TS38.214 section 5.1.3.2 [24].

  Given that the DCI is usually transmitted with a granularity of the slot, it enables to determine the number of bits being transmitted each millisecond or half-millisecond, depending on the numerology. Therefore, computing the transport block size of the data transmissions enables to build, for each UE and direction, a traffic profile that reflects the data transmissions of a UE

over the time. Those users' uplink and downlink profiles can be used to infer the type of traffic transmitted on the network. Indeed, some user traffics have precise payload size and periodicity profiles. For example, voice streaming might produce traffic with low transport block sizes but high regularity, whereas Hypertext Transfer Protocol (HTTP) traffic might produce uneven traffic with peaks. Supervision tools can leverage this data to identify the service associated with each user profile. However, this only works for terminals that do not transfer different traffics simultaneously.

Machine learning or deep learning algorithms can be used to classify the user traffics and identify the related services. Those algorithms must be trained by generating and labeling data sets and can then be used to classify the traffic based on a user profile. Traffic classification based on DCI extraction has already been investigated in 4G in [68]. The probe introduced in this chapter can be used to derive this work to 5G.

- The time and frequency position of the data transmissions can also be used to estimate the overall load of the cell. Indeed, given that the allocations of all the UEs can be decoded in both directions, it becomes possible to determine the available resources at every time. The load of the cell at a given time can be further analyzed to determine whether network congestion is due to an uneven peak or to a long-term overload of the cell. Furthermore, it can be used by supervision tools to raise alerts when the load of the cell reaches a given threshold.

- If the two previous KPIs are combined, the cell load can also be estimated per service, which can be helpful for network planning and optimization. Indeed, the operator may not position and allocate the resources similarly for different services. When the load of one service increases, the operator can re-deploy resources collocated with the resources already dedicated to this service.

- Furthermore, the time and frequency positions, together with the traffic classification, enable to infer how the MAC scheduler works and which scheduling algorithm is implemented. Supervision tools can leverage this data by controlling that the MAC scheduler implements the expected scheduling strategy for each service. It can raise an alarm if the scheduling algorithm used in the cell does not match the requirements of the services.

- Finally, the quality of the radio channel of a UE can be estimated by looking at the PHY configuration used for data transmissions. Indeed, the scheduler might use a low modulation scheme and code rate when the communication quality decreases. Furthermore, the HARQ process information and redundancy version can be combined to determine the number of re-transmissions required for a transport block. It can be estimated that the highest the number of re-transmissions, the lowest the channel's quality. This is essential information as critical services can be notified by supervision tools when the quality of the channel of a terminal accessing this service is decreasing.

8.3.2 Data decoding

When the number of computing resources on the probe is not limited or high, it is worth decoding the transport blocks corresponding to the previously extracted resource allocations. Once decoded, the MAC PDUs can be parsed, and the following information can be recovered:

- MAC control elements: un-encrypted MAC payload which contains the MAC control data.

- MAC SDUs: sub payload of the data to be transmitted. The headers include the MAC Logical Channel ID (LCID) and the SDU's size. A logical channel is a MAC layer channel that carries one specific traffic type. Some channels are reserved for network control and signaling, and others are used for user traffic.

The MAC layer headers and control elements provide much information about the network. Given that heterogeneous traffics use different logical channels at the MAC layer, the different QoS flows (and also signaling) can be split.

Therefore, the probe can build the traffic profile of each UE in each direction and for each type of traffic by combining the LCID information with the corresponding SDU size. This is a significant evolution compared to what can be done by just extracting the DCIs as the different traffics of one UE can be differentiated, and the user's profile can be built on a per traffic basis.

Finally, the MAC control elements can be used to understand the state and behavior of the network. The most interesting control element is the Buffer Status Report (BSR). It is used by UEs to report the buffer's state to the network. This is critical information that supervision tools can leverage. Indeed, it is interesting to provide critical services (especially services with delay constraints) with this information so that the service can adapt itself to the buffer state. Furthermore, the supervision application can estimate the required time to transmit the whole buffer by combining the available information: channel quality, overall cell load, and MAC scheduling strategy. It can raise an alarm when the estimated transmission delay exceeds the maximum service's delay constraint.

8.4 Simulations

In this section, we present the tests that have been done to validate the probe. Figure 8.7 represents the system architecture used for our test-bed. There are three components in the setup:

- The RF device: a USRP X310, which is connected to the host server through an optical fiber.
- The host device (server in Figure 8.7). Given that decoding all the downlink and uplink transport blocks of all the UEs can be highly resources consuming, the probe is not implemented in a local server close to the RF device but inside a VM in a datacenter.
- A telemetry stack (database in Figure 8.7). It is used to gather, store and display the data. Different stacks exist, and the one used in this setup is TICK Stack from Influx Data [69]. It includes a data collector (Telegraf), a time-series database (Influx DB), a graphing tool (Chronograf), and an alerting tool (Kapacitor).

The probe uses gRPC [70] to transfer data from the host to the telemetry stack. It is used as it is a modern and one of the most used protocols for data streaming.

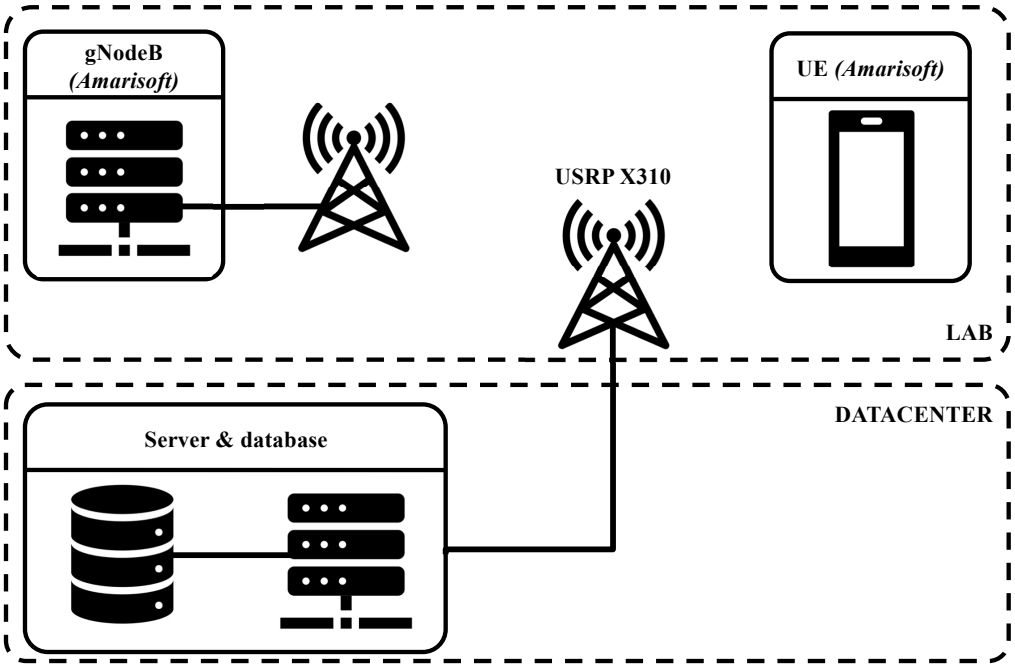


Figure 8.7: Probe testing system overview

The probe is tested and validated against an Amarisoft setup [71]. On one side, there is the Amarisoft 5G network which comprises the CN, the gNodeB and the SDR devices. On the other side, there is the Amarisoft UE's simulator.

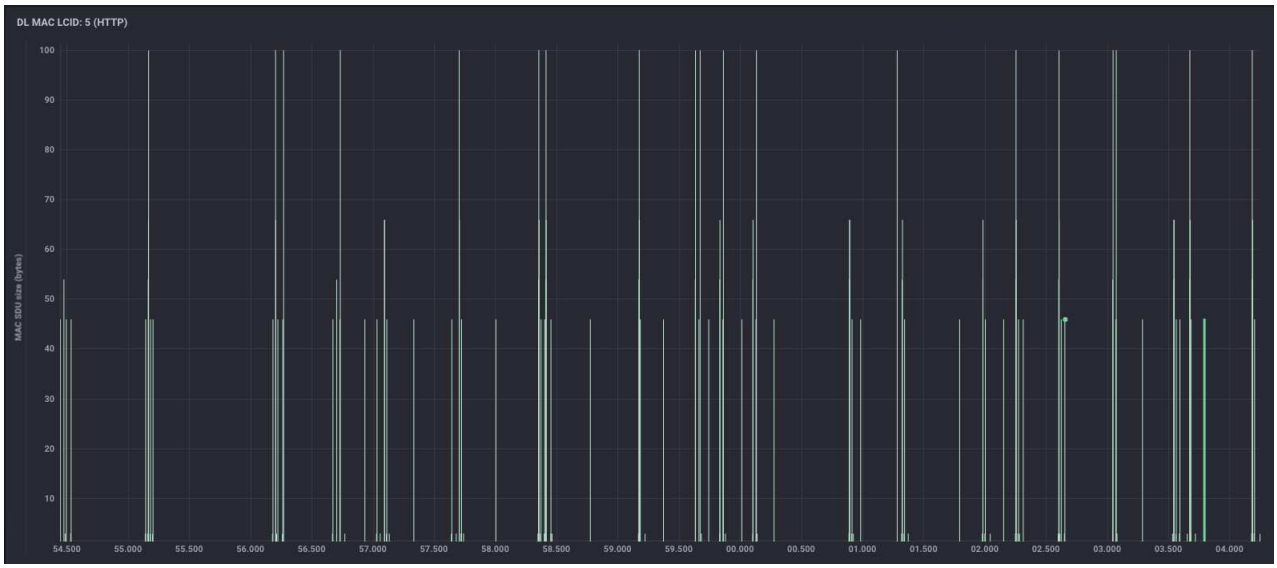


Figure 8.8: User's downlink profile for LCID 5 (HTTP traffic)

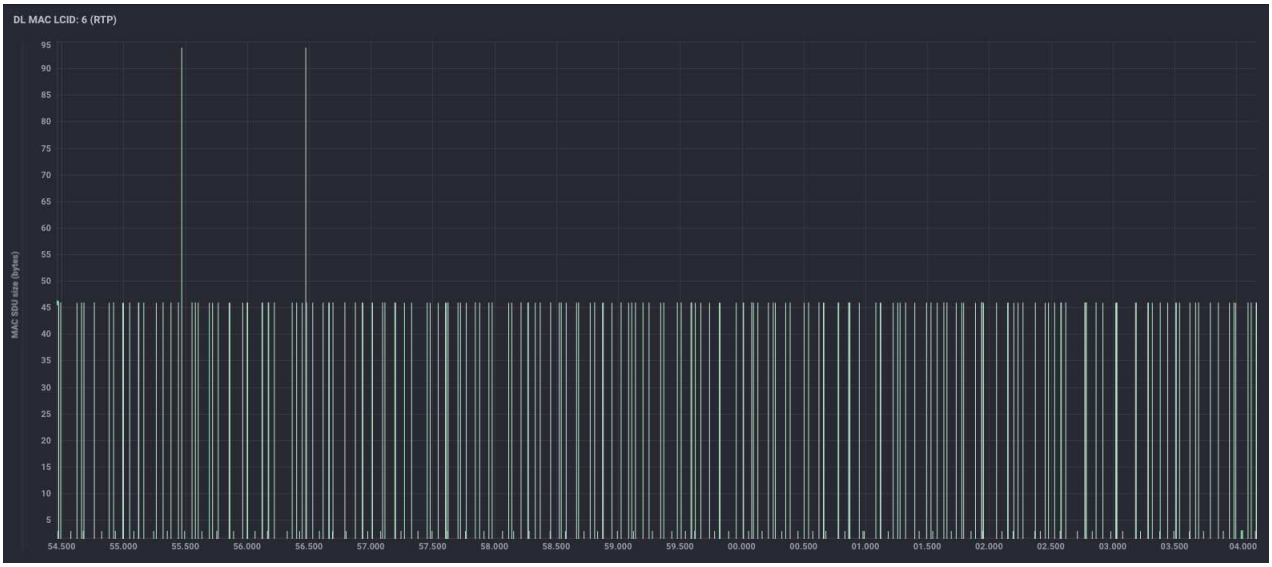


Figure 8.9: User's downlink profile for LCID 6 (RTP traffic)

The purpose of the simulation is to validate the probe in a generic case where a UE is doing voice calls and web browsing simultaneously. In this simulation, we focus on traffic classification and try to validate that classification algorithms can use the probe to recognize the different types of traffic within one cell. Therefore, we trigger a UE which connects the cell and generates 10 seconds of Real-time Transport Protocol (RTP) and HTTP traffic. RTP is the protocol used to carry the voice during Voice over IP (VoIP) calls and HTTP carries the web traffic. Using the Amarisoft's API, we determine that the LCID 5 is used for HTTP transfer and that LCID 6 is used for RTP transfer. The downlink and uplink profiles for the 2 MAC LCIDs of the user are shown in Figures 8.8, 8.9, 8.10 and 8.11, where the x-axis is the reception's time of the MAC SDU and the y-axis is the number of bytes of the corresponding SDU.

It can be observed that HTTP and RTP traffics produce very different downlink and uplink profiles. The simulations performed enable to validate that the data extracted from the cell indeed contains accurate and relevant information. It can be expected that classification algorithms will be able to identify the different traffics on the cell.

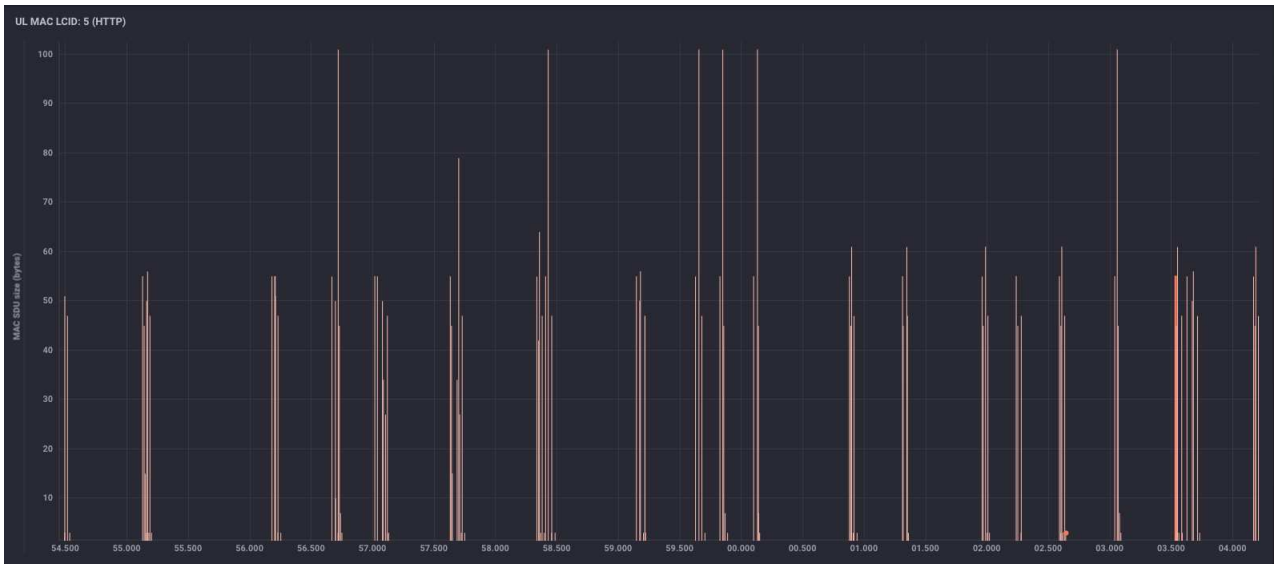


Figure 8.10: User's uplink profile for LCID 5 (HTTP traffic)

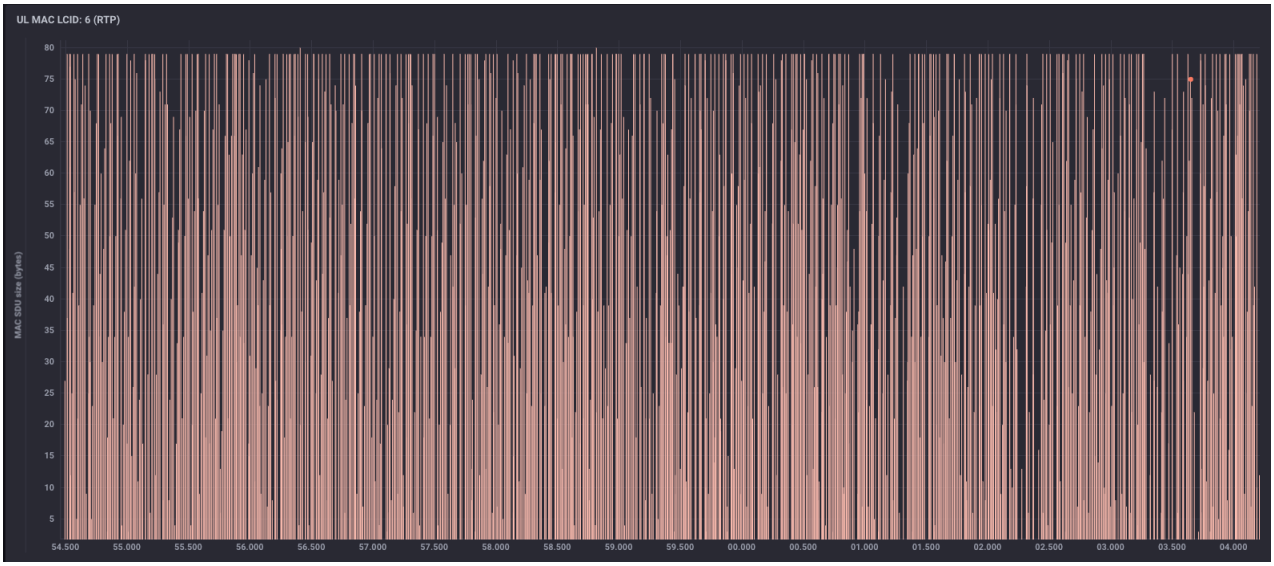


Figure 8.11: User's uplink profile for LCID 6 (RTP traffic)

8.5 Conlusion

In this chapter, we proposed the implementation of a 5G probe. This probe is derived from *free5GRAN*. It can be used to extract data from a 5G SA cell and does not require a connection to the network from which the data is extracted. The data extracted from the probe is two-fold. First, the DCIs, which provide resource allocation, are extracted. Then, the downlink and uplink transmissions are extracted. Depending on the requirements and available computing resources, the probe can be used to only decode the DCIs or to decode both the DCIs and the associated transmissions. Decoding the data transmissions has a high impact on the required computing resources.

The data extracted from the probe can be used to address different use cases. First, the resource allocation and MAC LCIDs can be used to infer the traffic type which is being transmitted. This information can be used to estimate the load of the network per traffic type but also to optimize the network's deployment. Furthermore, there are a lot of other Key Performance Indicator (KPI) that can be extracted with the probe, like the channel's quality (derived from the code rate and modulation scheme) or the users' buffer state (derived from the BSR MAC CE). Supervision tools can use all that information to monitor the network and validate that the network's state is good enough to

respect the different slices requirements. This brings a second level of control over the network, which increases the network's reliability.

Conclusion

In this thesis, network slicing was investigated. This technology enables the operators to deploy logical networks on top of a single physical infrastructure. Those logical networks can be dedicated to one type of service and one customer. Network slicing has been introduced to support the highly diversified terminals that connect the 5G network and require access to different data networks with heterogeneous user traffics constraints. It has received much attention as it is the technology that enables operators to address the new verticals introduced in 5G. Given that network slicing is achieved by associating resources to network slices, resource allocation is of the utmost importance for network slicing deployment. Network slicing implementation raises multiple problems, among which a significant part resides at the RAN level. This thesis addressed two challenges raised by network slicing at the RAN level.

The first one is the network modeling for resource allocation. Indeed, many models exist for resource allocation of the RAN but we are missing models which take into account new constraints implied by network slicing. The first contribution of this thesis was to define a new model for network slicing at the RAN level. This model, introduced in chapter 7, takes into account diverse slices constraints such as capacity, UEs density, latency, and reliability. Network coverage was introduced as being a fundamental criteria for network reliability. Simplicial homology was used to validate slices constraints fulfillment. Finally, this model was applied to power optimization, which is a critical aspect of network deployment. For future research, it would be interesting to include new radio technologies such as massive MIMO and beam-forming in this model. Furthermore, while the optimization algorithm introduced in this thesis only tries to decrease power consumption, it would be worth deriving the current algorithm so that it can also optimize the placement of the different cells.

The second challenge that has been addressed in this work is the network's supervision and control. Indeed, some new verticals have ultra high control requirements, and the network itself might not be able to satisfy this constraint fully. Therefore, in chapter 8, we introduced a probe that can extract data from the network to feed supervision tools for the network's monitoring and control. This probe was designed to be resilient to cyber-attacks and is thus independent of the network. In today's version, the probe can extract all the DCI messages which carry resource allocation and the associated uplink and downlink transmissions. Based on this raw data extracted by the probe, the MAC layer SDUs and control elements can be recovered. Further work could investigate the possibility of extracting more data, especially to parse the RLC SDUs. This would require inferring the RLC transmission mode (transparent, acknowledged, or un-acknowledged) and would enable supervision systems to leverage new KPIs such as RLC DRBs. Furthermore, it would be worth implementing BWPs in the probe as it is a fundamental technology for network slicing. Finally, future works could concentrate on using the data extracted by this probe. Indeed, the probe extracts a massive volume of data that can be streamed to supervision systems. Therefore, implementing algorithms that can analyze the different KPIs to control the network's health and classify the traffic is of paramount interest. We received a fund from the IP Paris doctoral school for this purpose, and this work has started within the laboratory to study the different algorithms that can be applied to the extracted data. The first use-case being investigated during this work is traffic classification.

The last main contribution of this thesis is the introduction of an open-source 5G physical layer called *free5GRAN*. This project (described in chapters 2, 3, 4 and 5) has been developed in parallel with the probe, as the two projects are closely related. The physical layer provides all the minimal procedures and algorithms for communications between the gNodeB and UEs. The project's structure was built so that one can easily modify it and implement new features. The software architecture

has been designed so that the physical layer is modular and can be derived to implement the open-RAN split 7.2. Further works could implement new radio technologies like MIMO and beamforming. Furthermore, it would be interesting to support FR2 bands, as those bands might raise a lot of new challenges. Finally, in the context of RAN virtualization and cloud deployments, it appears that software implementations paired with hardware acceleration are a good trade-off between flexibility and performance. Therefore, implementing resource consuming functions like channel coding and decoding on dedicated hardware devices like Field Programmable Gate Array (FPGA) and Graphics Processing Unit (GPU) is promising. This work has started by one intern in our laboratory. He worked on the implementation of the LDPC decoder in FPGA (leveraging USRP X3xx and X4xx FPGA cards) and integrated it in *free5GRAN*.

In this thesis, we showed that network slicing can be implemented with the current technologies. However, it requires much attention and the consideration of all the network's levels (like network functions, hardware resources, transport network, or radio interface). Furthermore, we exposed all the main procedures and algorithms involved in the 5G physical layer and proposed a concrete open-source implementation called *free5GRAN*.

Publications and contributions

Papers

- A. de Javel, J.S. Gomez, P.Martins, J.L. Rougier, P. Nivaggioli, *Slice-aware energy saving algorithm for 5G networks based on simplicial homology*, paper published in *IEEE VTC Spring 2021*.
- A. de Javel, J.S. Gomez, P.Martins, J.L. Rougier, P. Nivaggioli, *Towards a new open source 5G development framework: an introduction to free5GRAN*, paper published in *IEEE VTC Spring 2021*.
- Part I will be subject to future publication.
- Chapter 6 will be subject to future publication.

Open source code

- A. de Javel, P.Martins, *free5GRAN, An open-source 5G RAN PHY layer*, code available at: `https://github.com/free5G/free5GRAN`.
- The probe introduced in chapter 8 will be available publicly under the *free5G* organization of Github: `https://github.com/free5G/`

Appendix A

PHY layer implementation details

A.1 Synchronization

Synchronization relies on PSS and SSS signals. The sequences x, x_0 and x_1 (used below) are m-sequences. For PSS, a single m-sequence is used to generate the PSS sequence whereas, for SSS, a combination of two m-sequences (which produces a Gold sequence) is used. Those signals are generated based on two parameters $N_{ID}^{(1)}$ and $N_{ID}^{(2)}$ which depend on the cell's ID (N_{ID}^{cell}), such as:

$$N_{ID}^{cell} = 3 \cdot N_{ID}^{(1)} + N_{ID}^{(2)}$$

On the gNodeB's side, $N_{ID}^{(1)}$ and $N_{ID}^{(2)}$ are computed and the PSS and SSS signals can be generated accordingly. On the UE's side, N_{ID}^{cell} is not known and the UE generates all the possible PSS and SSS signals corresponding to the possible $N_{ID}^{(1)}$ and $N_{ID}^{(2)}$ values.

A.1.1 PSS sequence generation

PSS sequence, noted $d_{pss}(n)$, is defined in TS38.211 section 7.4.2.2.1 [21] and follows Equation:

$$d_{pss}(n) = 1 - 2 \cdot x(m)$$

Where

$$m = (n + 43 \cdot N_{ID}^{(2)}) \pmod{127}$$

and

$$x(i+7) = (x(i+4) + x(i)) \pmod{2}$$

with

$$[x(0), ..., x(6)] = [0, 1, 1, 0, 1, 1, 1]$$

A.1.2 SSS sequence generation

SSS sequences can be generated by the following equation:

$$d_{sss}(n) = [1 - 2 \cdot x_0((n + m_0) \pmod{127})] \times [1 - 2 \cdot x_1((n + m_1) \pmod{127})]$$

Where

$$m_0 = 15 \cdot \left\lfloor \frac{N_{ID}^{(1)}}{112} \right\rfloor \cdot 5 \cdot N_{ID}^{(2)} \ , \ m_1 = N_{ID}^{(1)} \pmod{112}$$

and

$$x_0(i+7) = (x_0(i+4) + x_0(i)) \pmod{2}$$
$$x_1(i+7) = (x_1(i+1) + x_1(i)) \pmod{2}$$

with

$$[x_0(0), ..., x_0(6)] = [1, 0, 0, 0, 0, 0]$$
$$[x_1(0), ..., x_1(6)] = [1, 0, 0, 0, 0, 0]$$

A.1.3 Frequency tracking implementation

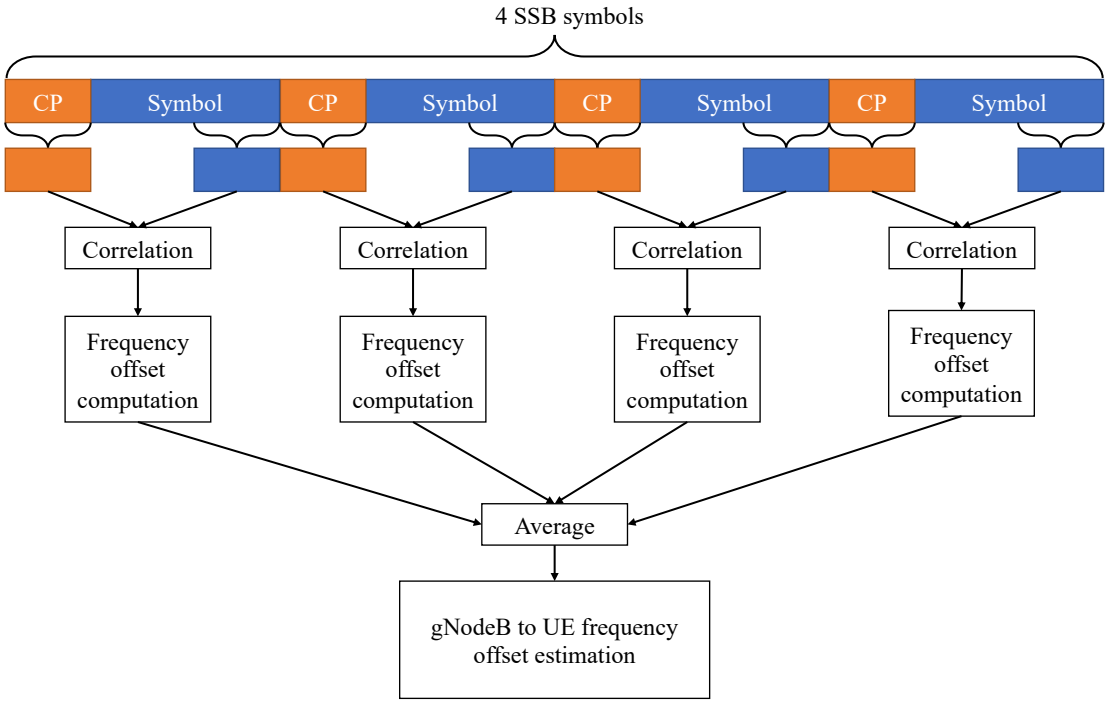


Figure A.1: Frequency offset estimation process

This section details how the fine frequency offset between the gNodeB and the UE can be computed. Figure A.1 represents the process to estimate the frequency offset between the gNodeB and the UE. The frequency offset between the gNodeB and the UE is estimated by averaging the frequency offsets of the four SSB symbols. Indeed, the frequency offset over one symbol is due to the spread delay and the actual frequency offset with the gNodeB. Given that delay spread varies from one symbol to another and that frequency offset is expected to be consistent over symbols, averaging the frequency offset over the four SSB symbols enables mitigation of the effects of spread delay over the frequency offset estimation.

The first step to compute the frequency offset for one symbol is to compute the phase difference (noted $d\Phi$) between the cyclic prefix and the corresponding part of the symbol, which is made by correlating the cyclic prefix with the symbol. Indeed, the result of the correlation, noted c, is a complex such as:

$$c = \rho_c \cdot e^{2\pi j \cdot d\Phi}$$

Therefore:

$$d\Phi = \frac{arg(c)}{2\pi}$$

Finally, given that the frequency offset for one symbol (noted f_{offset}^{symb}) equals:

$$f_{offset}^{symb} = \frac{d\Phi}{dt}$$

And that the duration of a symbol is equal to the inverse of the SCS, we obtain:

$$f_{offset}^{symb} = \frac{arg(c)}{2\pi} \cdot SCS$$

The frequency offset between the gNodeB and the UE (noted f_{offset}) is computed by averaging the frequency offsets of the four SSB symbols. Here is the *free5GRAN* code for frequency offset

Parameter	Length	Payload position
SFN	10	a_1, a_2, a_3, a_4, a_5, a_6, a_{24}, a_{25}, a_{26}, a_{27}
PDCCH config	8	a_{13}, a_{14}, a_{15}, a_{16}, a_{17}, a_{18}, a_{19}, a_{20}
Common SCS	1	a_7 (0: 15kHz, 1: 30kHz for FR1 bands)
Cell barred	1	a_{21}
DMRS Type A pos.	1	$2 + a_{12}$
k SSB	5	a_{29}, a_8, a_9, a_{10}, a_{11}
Int. Freq. Reselection	1	a_{22}

Table A.1: MIB payload for $L_{max} = 8$

computation:

```
// input_signal is the time domain received signal
// fft_size is the number of samples in a symbol (without cyclic prefix)
// cp_length is the number of samples in the cyclic prefix
// symbol_duration is the number of samples in a symbol with cyclic prefix = cp_length + fft_size
// avg_frequency_offset is the final output frequency offset
float avg_frequency_offset = 0;
// Looping over all the symbols
for (int symbol = 0; symbol < num_symbols; symbol++) {
    // Initialize correlation to 0
    complex<float> correlation = 0;
    // Compute correlation between the cyclic prefix and the symbol
    // Loop over all the samples of the cyclic prefix
    for (int i = 0; i < cp_length; i++) {
        correlation += conj(input_signal[i + symbol * symbol_duration]) *
            input_signal[i + symbol * symbol_duration + fft_size];
    }
    // phase_offset_symbol is the symbol's phase offset
    float phase_offset_symbol = arg(correlation);
    // Compute symbol frequency offset from phase offset
    // M_PI is the PI value in C++
    // scs is the subcarrier spacing
    float frequency_offset_symbol = scs * phase_offset_symbol / (2 * M_PI);
    avg_frequency_offset += frequency_offset_symbol;
}
// Average frequency offsets over all the symbols
avg_frequency_offset /= num_symbols;
```

A.2 Master Information Block

A.2.1 MIB payload compositions

The generic MIB payload size is 24 bits, but it is added another 8 bits for finer timing information, which leads to a 32 bits payload. The MIB payload is generated depending on L_{max}, as specified in TS38.212 section 7.1.1 [11]. Table A.1 gives an example of MIB payload composition for $L_{max} = 8$. Some indexes do not appear are they are reserved and not used for MIB payload.

A.2.2 MIB processing functions

Figure A.2 represents the overall BCH and PBCH processing functions together with the size of the associated sequences.

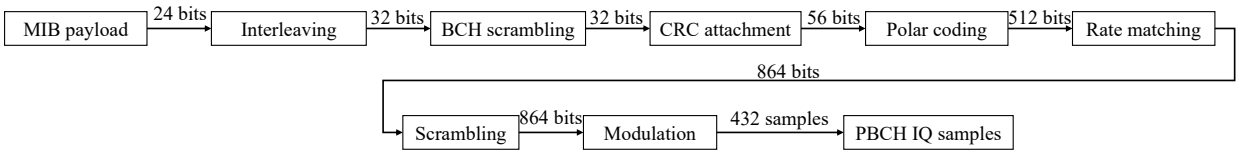


Figure A.2: Size of the sequences involved in BCH/PBCH encoding

A.2.2.1 Bits interleaving *(Transport channel processing - TS38.212 section 7.1.1)*

Bits interleaving is the process by which the initial 24 MIB bits are interleaved with the 8 other bits to form a 32 bits payload. In our case, this is not a specific function as it is done while computing the MIB payload: Table A.1 shows the MIB payload composition after bits interleaving.

A.2.2.2 BCH scrambling *(Transport channel processing - TS38.212 section 7.1.2)*

BCH scrambling is based on the LSB of the SFN, and guarantees that no confusion is possible between two MIB payloads with a close SFN. The first step is to choose a v value depending on SFN, using TS 38.212 Table 7.1.2-1. Then, scrambling can be performed following Algorithm 4.5, where s sequence is generated based on v using the algorithm described in TS 38.212 section 7.1.2.

A.2.2.3 CRC attachment *(Transport channel processing - TS38.212 section 7.1.3)*

CRC attachment and validation is made using g_{CRC24C} polynom, as detailed in section 4.2. If the receiver do not validate the CRC, the MIB decoding process is failed.

A.2.2.4 Channel coding *(Transport channel processing - TS38.212 section 7.1.4)*

The BCH channel coding algorithm is polar coding. This algorithm is detailed in section 4.3.1.

A.2.2.5 Rate matching *(Transport channel processing - TS38.212 section 7.1.5)*

Rate matching is detailed in section 4.4.1. The parameters are $E = 864$ (BCH channel size) and $I_{BIL} = 0$. After rate matching, BCH encoding is done and the channel can be sent to the PBCH encoder.

A.2.2.6 Scrambling *(Physical channel processing - TS38.211 section 7.3.3.1)*

Scrambling is the first function of the PBCH encoding. It consists in scrambling the 864 bits with a sequence determined by the PCI in order to avoid confusion at the cells' borders. Scrambling is made according section 4.5, by setting $s_i = c_{i+i_{SSB} \cdot M_{bit}}$, where $M_{bit} = E = 864$. For FR2 bands, the parameter i_{SSB} is encoded on 6 bits, and another parameter called $\bar{i}_{SSB}$ is introduced as being the 3 LSB of i_{SSB} ($\bar{i}_{SSB} = i_{SSB} \pmod 8$) and i_{SSB} is replaced by $\bar{i}_{SSB}$ in the previous formula. c is a Gold sequence computed according to TS38.211 section 5.2 [21] using $c_{init} = N_{ID}^{CELL}$ (PCI).

A.2.2.7 Modulation *(Physical channel processing - TS38.211 section 7.3.3.2)*

The 864 bits are modulated according section 4.6 using QPSK modulation scheme, which returns a set of 432 IQ samples.

A.2.2.8 DMRS generation *(Physical signal processing - TS38.211 section 7.4.1.4)*

DMRS is a pilot signal that will be used by UEs to correct the received signal. As detailed earlier, there is 144 available positions for DMRS in the PBCH allocation within SSB. The 144 elements of the DMRS sequence can be generated according TS38.211 section 7.4.1.4.1. It depends on i_{SSB} and the cell's PCI (N_{ID}^{cell}).

A.3 PDCCH

A.3.1 PDCCH processing functions

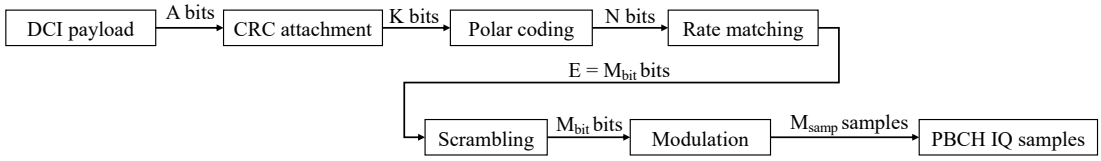


Figure A.3: Size of the sequences involved in DCI/PDCCH encoding

Figure A.3 represents the overall DCI and PDCCH processing functions together with the size's notation of the associated sequences.

A.3.1.1 Channel coding *(Transport channel processing - TS38.212 section 7.3.3)*

The DCI channel coding algorithm is polar coding. This algorithm is detailed in section 4.3.1.

A.3.1.2 Rate matching *(Transport channel processing - TS38.212 section 7.3.4*

Rate matching is detailed in section 4.4.1. After rate matching, DCI encoding is completed and the payload can be moved to the PDCCH encoder.

A.3.1.3 Scrambling *(Physical channel processing - TS38.211 section 7.3.2.3)*

Scrambling process is made according section 4.5, by setting $s_i = c_i$, where c is computed according to TS38.211 section 5.2. c_{init} depends on the cell's configuration. In a default configuration, $c_{init} = N_{ID}^{cell}$.

A.3.1.4 Modulation *(Physical channel processing - TS38.211 section 7.3.2.4)*

Modulation is defined in section 4.6 and modulation scheme is QPSK.

A.3.1.5 DMRS generation *(Physical signal processing - TS38.211 section 7.4.1.4)*

DMRS signal is computed using equation A.1.

$$r_l(m) = \frac{1}{\sqrt{2}} \cdot (1 - 2 \cdot c(2 \cdot m)) + j \cdot \frac{1}{\sqrt{2}} \cdot (1 - 2 \cdot c(2 \cdot m + 1)) \tag{A.1}$$

Where c is computed according to TS38.211 section 5.2 using :

$$c_{init} = \left[2^{17} \cdot (N_{symb}^{slot} \cdot n_{s,f}^{\mu} + l + 1) \cdot (2 \cdot N_{ID} + 1) + 2 \cdot N_{ID}\right] \pmod{2^{31}}$$

Where N_{symb}^{slot} is the number of symbols per slot (i.e. 14), $n_{s,f}^{\mu}$ is the slot number within the frame, l is the symbol number within the slot and $N_{ID} = N_{ID}^{cell}$ in a default configuration.

A.3.2 CCE-to-REG mapping

When CCE-to-REG mapping is activated, the CCEs selected for PDCCH transmission must be interleaved to be mapped to REG bundles.

The interleaver function is defined in TS38.211 section 7.3.2.2 [21]. Figure A.4 represents the mapping between CCEs and REG bundles when $R = 2$ (Interleaver size) and $n_{shift} = N_{ID}^{cell} = 250$ for two CORESETs (one with 48 RBs by 1 symbol and another one with 24 RBs by 2 symbols). C is defined by $C = N_{REG}^{CORESET}/(6 \cdot R)$ where $N_{REG}^{CORESET}$ is the number of REGs in the CORESET. In this example, $C = 4$. If CCEs $\{4, 5, 6, 7\}$ are selected, then PDCCH element IQ samples will be placed in REG bundles $\{4, 0, 5, 1\}$, i.e. in REGs $\{24, 25, 26, 27, 28, 29, 0, 1, 2, 3, 4, 5, 30, 31, 32, 33, 34, 35, 6, 7, 8, 9, 10, 11\}$.

The REGs order is important, as the IQ samples are placed in increasing order of CCE, i.e. respecting the order after CCE-to-REG mapping.

Once the REGs are selected and ordered, the PDCCH IQ samples can be placed into the REGs. REs $\{1, 5, 9\}$ of each RB are reserved for PDCCH DMRS placement and PDCCH IQ samples can be placed in the 9 other positions. With the same example and with a 24 RBs by 2 symbols CORESET, PDCCH IQ samples will be placed in the positions (i, j) (which corresponds to subcarrier i of symbol j in the CORESET) from: $(144, 0)$, $(146, 0)$, $(147, 0)$, $(148, 0)$, $(150, 0)$, $(151, 0)$, $(152, 0)$, $(154, 0)$, $(155, 0)$, $(144, 1)$, $(146, 1)$, $(147, 1)$, $(148, 1)$, $(150, 1)$, $(151, 1)$, $(152, 1)$, $(154, 1)$, $(155, 1)$ for REG 24 to positions $(60, 0)$, $(62, 0)$, $(63, 0)$, $(64, 0)$, $(66, 0)$, $(67, 0)$, $(68, 0)$, $(70, 0)$, $(71, 0)$, $(60, 1)$, $(62, 1)$, $(63, 1)$, $(64, 1)$, $(66, 1)$, $(67, 1)$, $(68, 1)$, $(70, 1)$, $(71, 1)$ for REG 11.

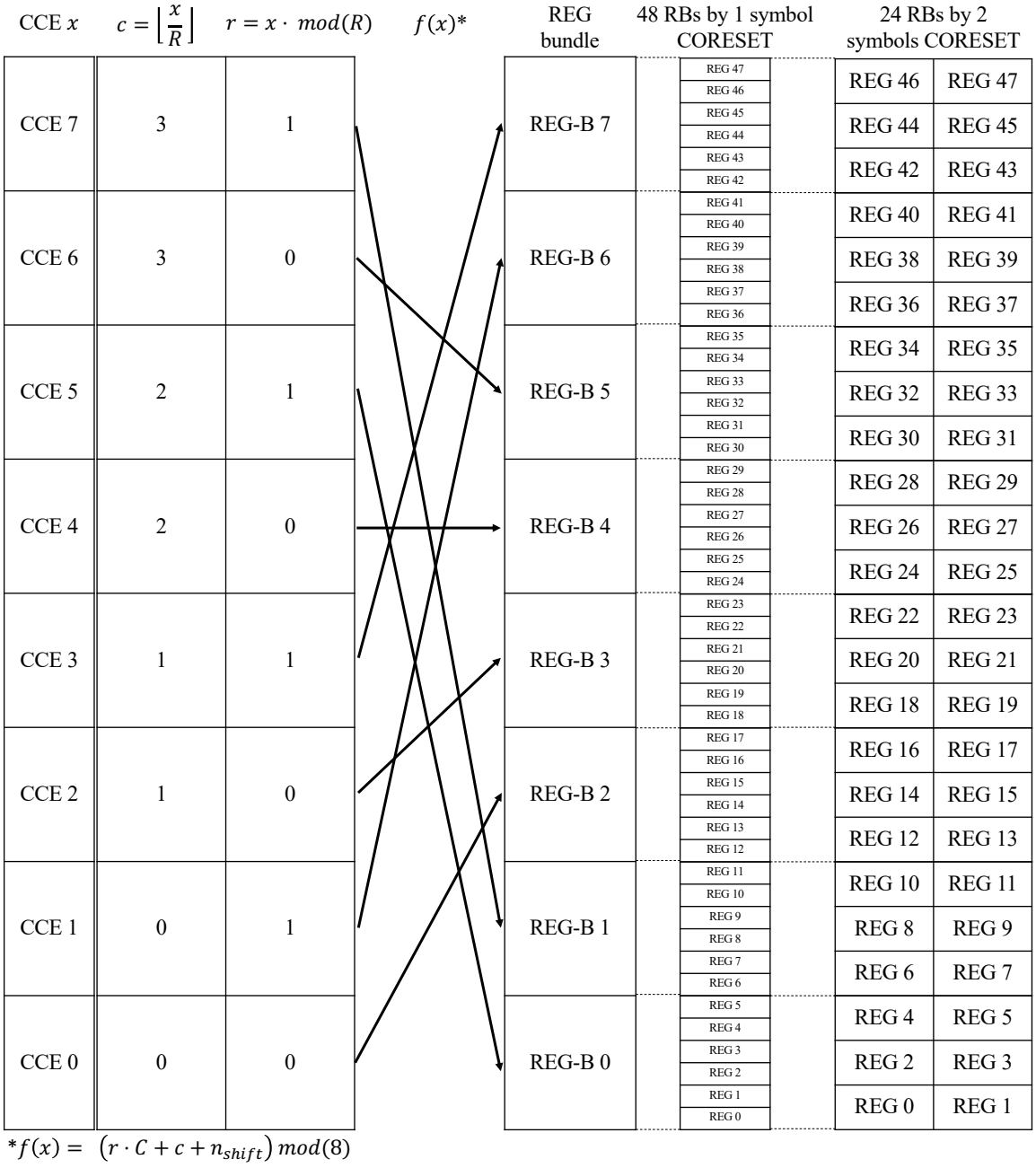


Figure A.4: Example of CCE-to-REG mapping for two possible CORESETs (with $n_{shift} = N_{ID}^{cell} = 250$ and $R = 2$)

A.4 PDSCH

A.4.1 PDSCH processing functions

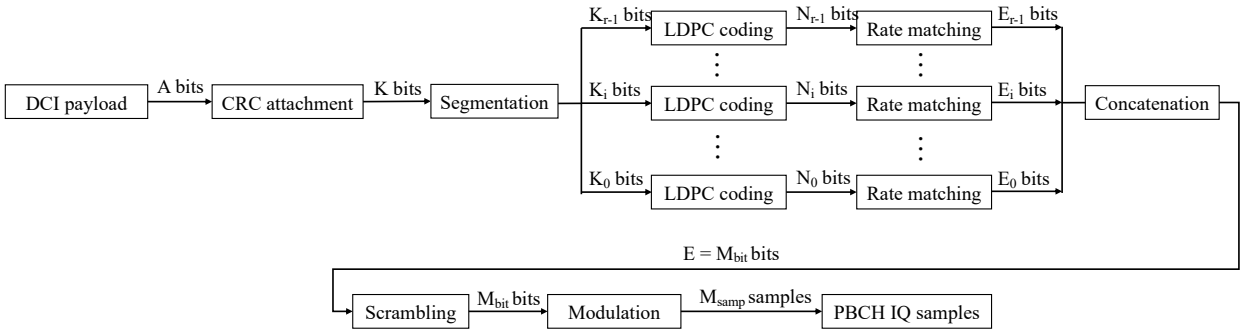


Figure A.5: Size of the sequences involved in DL/UL-SCH and PDSCH/PUSCH encoding

Figure A.5 represents the overall DCI and PDCCH processing functions together with the size's notation of the associated sequences.

A.4.1.1 CRC attachment *(Transport channel processing - TS38.212 section 7.2.1)*

Depending on the transport block (DL-SCH payload) size A, the CRC polynom to be used can be either g_{CRC24A} when $A > 3824$ or g_{CRC16} otherwise. Once computed, the CRC is appended to the end of the payload and the new payload size is $B = A + 24$ or $B = A + 16$ depending on the CRC polynom.

A.4.1.2 Base graph selection *(Transport channel processing - TS38.212 section 7.2.2)*

The BG is the parity check matrix used for LDPC encoding and decoding. Its selection is based upon transport block size A and the code rate R and impacts the channel coding parity check matrix to be used. If $A > 3824$ and $R > 0.67$, BG 1 is used and BG 2 is used otherwise.

A.4.1.3 Code block segmentation *(Transport channel processing - TS38.212 section 7.2.3)*

Before performing channel coding, DL-SCH payload can be split into different code blocks when the payload size is high. When splitting the payload into different code blocks, each code block will be added another CRC (using g_{CRC24B} polynom). Given that the code block size can only take a given number of values, some filler bits (also called NULL bits) are added to the end of each code block. The code block segmentation process is described in TS38.212 section 5.2.2. The two following procedure steps, channel coding and rate matching, are done for each code block before code blocks are concatenated after rate matching.

A.4.1.4 Channel coding *(Transport channel processing - TS38.212 section 7.2.4)*

DL-SCH channel coding method is LDPC. Each code block is added a set of parity bits, as detailed in sections 4.3.2 and 1.3.2.3.2.

A.4.1.5 Rate matching *(Transport channel processing - TS38.212 section 7.2.5)*

After channel coding, the size of each code block has to be adjusted to match the number of available resources in the allocation. This is done following section 4.4.2.

A.4.1.6 Code block concatenation *(Transport channel processing - TS38.212 section 7.2.6)*

After rate matching, all the code blocks are concatenated to give a final physical layer payload.

A.4.1.7 Scrambling *(Physical channel processing - TS38.211 section 7.3.1)*

After DL-SCH encoding, PDSCH encoding can be performed. The first step is scrambling. Scrambling is made according section 4.5 by setting $s_i = c_i$, where c is computed according to TS38.211 section 5.2. In a default configuration, we consider there is only one code word per transmission and c_{init} is then given by $c_{init} = n_{RNTI} \cdot 2^{15} + n_{ID}$ where n_{RNTI} is the UE's identifier and $n_{ID} = N_{ID}^{cell}$ by default.

A.4.1.8 Modulation *(Physical channel processing - TS38.211 section 7.3.2)*

Modulation is made following section 4.6. For PDSCH, modulation order m (or Q_m) is given by the MCS field of the DCI.

A.4.1.9 DMRS generation *(Physical channel processing - TS38.211 section 7.4.1.1.1)*

DMRS sequence r is generated using Equation:

$$r(n) = \frac{1}{\sqrt{2}} \cdot (1 - 2 \cdot c(2 \cdot n)) + j \cdot \frac{1}{\sqrt{2}} \cdot (1 - 2 \cdot c(2 \cdot n + 1))$$

Where n is the subcarrier index within the BWP. c is computed according to TS38.211 section 5.2 using different parameters. In a default configuration:

$$c_{init} = \left(2^{17} \cdot (N_{symb}^{slot} \cdot n_{s,f}^{\mu} + l + 1) \cdot (2 \cdot N_{ID}^{cell} + 1) + 2 \cdot N_{ID}^{cell}\right) \pmod{2^{31}}$$

Where N_{symb}^{slot} is the number of symbols per slot (i.e. 14), $n_{s,f}^{\mu}$ the slot number within the frame and l the symbol number within slot.

A.5 PDCCH with CORESET 0 and search space 0 placement

The configuration of CORESET 0 and search space 0 used for SIB1 transmission are given by the parameter $PDDCHconfigcommon$ from the MIB. This parameter is an 8 bits sequence where the 4 first bits give an index (between 0 and 15) in Tables 13-1 to 13-10 of TS38.213 [5] and the 4 last bits give an index in Tables 13-11 to 13-15. The appropriate tables to be used depend on the cell's configuration and are either fixed for a specific band or provided by the MIB. For example, in band $n78$ and with a $CommonSCS$ (from MIB) of 30kHz, the two tables to be used are Table 13-4 and 13-11. If the $PDDCHconfigcommon$ is set to 160, then the index in the Table 13-4 is 10 and the index in Table 13-11 is 0 ($10 \cdot 16 + 0 = 160$). The first Table gives the configuration of CORESET 0. For example, in Table 14-4 with index 10, CORESET 0 is a 1 symbol per 48 RBs CORESET ($N_{symb}^{CORESET} = 1$ and $N_{RB}^{CORESET} = 48$). The offset column defines the offset, in RBs, between the lowest RB of the SSB and the lowest RB of the CORESET. This defines the position of the CORESET with regard to the SSB (CORESET 0 is a specific CORESET as it is used at a very initial connection step, and the frequency position is thus not given with regard to the BWP but with regard to the SSB as the UE does not know yet the BWP configuration). The second table provides information used to build search space 0, which is not a usual search space, but rather a simplified search space used to locate CORESET 0. The four values given by the second table are two variables called O and M as well as the number of occurrences per slot and the first symbol index of the CORESET within the slot. O and M are used to determine the frames where CORESET 0 can be found and the slot index within frame n_0, using Equations:

$$n_0 = (O \cdot 2^{\mu} + \lfloor i_{SSB} \cdot M \rfloor) \pmod{N_{slot}^{frame,\mu}}$$

Where μ is the numerology indicator and $N_{slot}^{frame,\mu}$ the number of slots per frame when numerology is μ. The frames where CORESET 0 is sent must respect:

$$SFN \pmod 2 = \left\lfloor \frac{O \cdot 2^{\mu} + \lfloor i_{SSB} \cdot M \rfloor}{N_{slot}^{frame,\mu}} \right\rfloor \pmod 2$$

For example, with index 0 in TS38.213 Table 13-11, $O = 0$ and $M = 1$. Given that $\mu = 1$ (SCS is 30kHz), $N_{slot}^{frame,\mu} = 20$ and $i_{SSB} = 6$, CORESET 0 must be placed every even frame ($\lfloor (0 \cdot 2^1 + \lfloor 6 \cdot 1 \rfloor)/20 \rfloor \pmod 2 = 0$). Within even frames, it is placed on the first symbol of slot $n_0 = 6$.

A.6 PRACH signal generation

A good introduction of the PRACH signal generation can be found in [72].

A.6.1 Preamble selection

The first step for a UE willing to access a cell is to choose a preamble for RA. This process is detailed in TS38.213 section 8.1. Indeed, the existence of multiple preambles enables multiple UEs to transmit PRACH on the same occasion without interfering with each other. All the possible 5G preambles are between 0 and 63, but the available ones may be restricted to a subset depending on the configuration. Indeed, the preambles might be split into different groups in order to reduce the contention probability. Moreover, for the sake of beamforming, preambles are allocated to each SSB (and thus to beams) so that the gNodeB can detect what is the beam chosen by the UE. For example, it is considered that there are no preamble groups (i.e., the 64 preambles are available for the UE). If there are two SSBs per SSB period and *ssbperRACHOccasionAndCBPreamblesPerSSB* equals $(two, n8)$, which means that there is two SSBs per PRACH occasion and 8 preambles per SSB, then the number of available preambles is 16 and preambles 0 to 7 are used when the first SSB in the period is selected by the UE and 8 to 15 when the second one is used. In a simpler case, if there is only one SSB per SSB period and *ssbperRACHOccasionAndCBPreamblesPerSSB* equals $(one, n8)$, which means that there is one SSB per PRACH occasion and 8 preambles per SSB, then the number of available preambles is 8 and preambles 0 to 7 must be used. Once the set of available preambles is determined by the UE, it has to select one randomly.

A.6.2 Frequency domain sequence generation

Once a preamble p is chosen, a sequence root index u' and a cyclic shift index v can be determined. Then, an initial time-domain sequence is generated based on u' and shifted v times by N_{CS}. N_{CS} is called cyclic shift size, giving the number by which the initial sequence is shifted. Before determining u' and v, UE has to compute the number of possible shifts per root sequence n_{shift}^{seq} using following Equation:

$$n_{shift}^{seq} = \left\lfloor \frac{L_{RA}}{N_{CS}} \right\rfloor$$

Where L_{RA} is the size of the PRACH sequence and N_{CS} is the size of the cyclic shift.

For example, for a TDD FR1 cell with $prachConfigurationIndex = 160$, then the PRACH format is $B4$ and therefore $L_{RA} = 139$. Furthermore, Table 6.3.3.1-7 from TS38.211 must be used to determine N_{CS}. If $zeroCorrelationZoneConfig = 15$, then $N_{CS} = 69$. In this case, $n_{shift}^{seq} = 2$.

u' and v can be determined from p such as $p = u' \cdot n_{shift}^{seq} + v$ (i.e. $u' = \lfloor p/n_{shift}^{seq} \rfloor$ and $v = p \pmod{n_{shift}^{seq}}$). Using the same example, if the preamble is $p = 5$, then $u' = 2$ and $v = 1$. The initial time domain sequence is generated using a parameter u, which is deduced from u' (u' is an index in a Table whereas u is the value associated with this index). u is the $(u' + prachRootSequenceIndex + 1)$-th element of one table (Tables 6.3.3.1-3 to 6.3.3.1-4B in TS38.211). In our example, if $prachRootSequenceIndex = 1$, $u' = 2$ and $L_{RA} = 139$, then u is the element of Table 6.3.3.1-4 with index $(2 + 1)$ which is 137.

The initial time domain sequence $x_{u,v}$ is computed using:

$$x_{u,v}(n) = x_u((n + C_v) \pmod{L_{RA}})$$

Where $C_v = v \cdot N_{CS}$ (for a simple configuration) and:

$$x_u(i) = e^{-j\frac{\pi \cdot u \cdot i \cdot (i+1)}{L_{RA}}}$$

Once the initial time domain sequence is computed, it can be converted to frequency domain sequence $y_{u,v}$ using an FFT:

$$y_{u,v}(n) = \sum_{m=0}^{L_{RA}-1} x_{u,v}(m) \cdot e^{-j\frac{2 \cdot \pi \cdot m \cdot n}{L_{RA}}}$$

with $n, i \in [0, L_{RA} - 1]$ for all the above Equations.

A.6.3 Time domain signal generation *(TS38.211 sections 5.3.2 and 6.3.3)*

Before generating time domain signal, a frequency domain PRACH occasion must be selected. The RRC parameter $msg1FDM$ gives the number of frequency domain PRACH occasions. A PRACH occasion is selected by picking a value $o \in [0, msg1FDM - 1]$.

The time-domain PUSCH signal can be generated from the frequency domain sequence. The first step is to create an OFDM grid of one symbol with a subcarrier spacing equal to $msg1SubcarrierSpacing$. The L_{RA} elements of the sequence can be placed contiguously in the frequency domain from the first subcarrier of the selected frequency domain PUSCH occasion SC_0:

$$SC_0 = msg1FrequencyStart \cdot 12 \cdot \frac{SCS_{PUSCH}}{SCS_{PUSCH}} + o \cdot (12 \cdot N_{RB}^{RA} \cdot \frac{SCS_{PUSCH}}{SCS_{PRACH}}) + \bar{k} \tag{A.2}$$

Where $\bar{k}$ and N_{RB}^{RA} are determined from Table 6.3.3.2-1 of TS38.211. $\bar{k}$ is the first subcarrier in the RB where PRACH can be transmitted and N_{RB}^{RA} is the number of RBs occupied by one PRACH occasion. o corresponds to the selected frequency domain PRACH occasion ($o \in [0, msg1FDM - 1]$), SCS_{PUSCH} is the SCS used for PUSCH (given in SIB1) and SCS_{PRACH} is the SCS used for PRACH transmission ($SCS_{PRACH} = msg1SubcarrierSpacing$).

For example, for $msg1FDM = 1$ (i.e. there is only one frequency domain PRACH occasion), $msg1FrequencyStart = 0$ (i.e. the first RB allocated for the first PRACH occasion is the first RB of the band) and $L_{RA} = 139$ (i.e. $\bar{k} = 2$), then the 139 elements of the sequence are placed from subcarriers 2 to 140 in the OFDM symbol. Once the elements are placed in the OFDM symbol, iFFT can be performed to get the time domain sequence noted s'.

After time domain sequence generation, the PRACH signal has to be built according to the PRACH format using Tables 6.3.3.1-1 and 6.3.3.1-2 (with $\kappa = FFT_{size}/1024$). Final time domain PRACH signal can be generated by repeating s' several times and by copying the N_{RA}^{CP} last IQ samples of s' in front of the repetitions. The number of repetitions is given by the above tables. For example, for $FFT_{size} = 1024$, $\mu = 1$ and PRACH format $B4$, $N_{RA}^{CP} = 936 \cdot 1024/1024 \cdot 2^{-1} = 468$. The PRACH signal s can be generated by repeating 12 times the 1024 elements of s' and by pushing in front of it the 468 last elements of s'. The free space between the end of the PRACH signal and the start of the next symbol is reserved for guard period.

Once the PRACH signal s has been generated, it must be transmitted by selecting a time-domain PRACH occasion. As described in section 3.4.1.1, time-domain PRACH occasions are given by the configuration in Tables 6.3.3.2-2 to 6.3.3.2-4 of TS38.211. The first step is to determine PRACH occasions within an eligible slot, and the second one is to determine the eligible slots. We propose two examples in order to explain how time domain PRACH occasions are determined (both of those examples are for $\mu = 1$ with TDD FR1 cells: table 6.3.3.2-3 is used):

- First, for $prachConfigurationIndex = 74$. PRACH duration is 2 symbols and there is 6 occasions per eligible slots starting at symbol 0 (6×2 symbols from symbol 0 to symbol 11). Then, it is specified that there is two eligible slots per subframes. This is possible only when there is more that 1 slot per subframe. In our example, $\mu = 1$ which means that the SCS is 30kHz and that there is two slots per subframe. The two slots of subframes 8 and 9 (slots 16, 17, 18 and 19 within a frame) are eligible for PRACH occasions. Finally, $x = 2$ and $y = 1$ which

means that PRACH occasions occur every odd frames (i.e. $SFN \pmod 2 = 1$). The PRACH occasions are then the 6 pairs of two symbols from 0 to 11 of slots 0 and 1 of subframes 8 and 9 within every odd frames.

- For the second example, we consider $prachConfigurationIndex = 160$. PRACH duration is 12 and there is 1 occasion per slot which starts at symbol 2 (12 symbols from 2 to 13). The number of PRACH slots per subframe is 1. As $\mu = 1$ (2 slots per subframe), slot 1 is selected as detailed in TS38.211 section 5.3.2. If there were only one slot per subframe, slot 0 would have been mandatory selected. Eligible subframes are subframes 9 and PRACH occasions occur every frame ($SFN \pmod 1 = 0$). The PRACH occasions are therefore the 12 symbols 2 to 13 of slots 1 of subframes 9 (i.e. slot 19 within a frame) of every frames.

Once UE has determined the time domain PRACH occasions, it can select one of them and transmit s.

A.7 CRC validation code

CRC validation is made by dividing the received transport block with the polynom used for CRC computation. CRC is validated if the remainder equals 0. Here is the code in *free5GRAN*:

```
// CRC VALIDATION
// length_crc is the number of bits in the polynom
// Vector that will contain the division remainder
vector<uint8_t> remainder(length_crc);
// Vector that will contain the temporary input
vector<uint8_t> temp_input(length_input);
// temp_input is initialized with the input sequence
for (int i = 0; i < length_input; i++) {
    temp_input[i] = input_bits[i];
}
// crc_validated is the returned boolean, initialized to true (CRC validated)
crc_validated = true;
// Number of iterations
int num_steps = length_input - length_crc + 1;
// Loop over all the iterations
for (int step = 0; step < num_steps; step++) {
    // If the step-th temp_input element is 1 (otherwise, move to next step)
    if (temp_input[step] != 0) {
      // Loop over the CRC polynom
      for (int j = 0; j < length_crc; j++) {
        // crc_polynom is the polynom used for CRC computation
        // XOR the temp_input element with the polynom
        remainder[j] = temp_input[j + step] ^ crc_polynom[j];
        // Update temp_input element
        temp_input[j + step] = remainder[j];
      }
      // Early return is case there is still 1 in position that
      // will not be updated anymore, CRC is not validated
      if (temp_input[step] == 1){
        crc_validated = false;
        break;
      }
    }
}
// Loop over the final remainder, if there is 1, CRC is not validated
for (int i = 0; i < length_crc; i++) {
    if (temp_input[i + (length_input - length_crc)] == 1){
      crc_validated = false;
      break;
    }
}
```

A.8 Polar coding *(Transport channel processing - TS38.212 section 5.3.1)*

This section introduces the main processing steps and algorithms used for polar coding. The case where $n_{PC} > 0$ (n_{PC} is the number of parity bits) is not covered in this section. Table A.2 gives an overview of the polar coding inputs and outputs. In the encoding direction, the processing steps follow the order of this section whereas, in the decoding direction, the order is reversed, which means that the first step is to recover u from d, the second step is to recover c' from u, and the last one is to recover c from c'.

Before starting, the output sequence size N has to be computed following Algorithm given in TS38.211 section 5.3.1.

Name	Input or output	Description
K	Input	Input bits sequence size
c_i, $i \in [0, K-1]$	Input for coding, output for decoding	Input bits sequence
E	Input	Rate matching output size (i.e. number of PBCH bits)
N	Input	Polar coding output size
d_i, $i \in [0, N-1]$	Output for coding, input for decoding	Output bits sequence
n_{PC}	Input	Number of parity bits
I_{IL}	Input	Indicates whether or not interleaving should be performed

Table A.2: Polar coding and decoding input parameters

A.8.1 Interleaving

The first step while performing polar coding is to interleave input sequence c into sequence c'. In the decoding direction, de-interleaved is the last step before recovering input bits sequence c. Interleaving is activated if input parameter $I_{IL} = 1$ and deactivated if $I_{IL} = 0$. The interleaving process is based on a sequence called Π that can be computed following Algorithm 10, by setting $K_{max} = 164$ and getting $\Pi_{IL}^{max}(n)$ sequence from TS38.212 Table 5.3.1.1-1.

Algorithm 10 Π sequence computation

1: $k = 0$
2: **for** $n \in [0, K_{max} - 1]$ **do**
3: **if** $\Pi_{IL}^{max}(n) \geq K_{max} - K$ **then**
4: $\Pi(k) = \Pi_{IL}^{max}(n) - (K_{max} - K)$
5: $k = k + 1$
6: **end if**
7: **end for**

A.8.1.1 Encoding

Interleaving process is detailed in Algorithm 11.

Algorithm 11 Interleaving

1: **for** $k \in [0, K-1]$ **do**
2: $c'_k = c_{\Pi(k)}$
3: **end for**

A.8.1.2 Decoding

De-interleaving is done by replacing $c'_k = c_{\Pi(k)}$ by $c_{\Pi(k)} = c'_k$ in Algorithm 11, line 2.

The following code snippet is extracted from *free5GRAN* receiver for Π sequence generation and

de-interleaving:

```
// POLAR DECODING: PI SEQUENCE GENERATION AND DE-INTERLEAVING
// free5GRAN::INTERLEAVING_PATTERN is TS38.212 Table 5.3.1.1-1
// K_max is the maximum size after CRC attachment
// K is the payload size after CRC attachment
// int count_seq = 0;
for (int m = 0; m < K_max; m++) {
    if (free5GRAN::INTERLEAVING_PATTERN[m] >= K_max - K) {
      pi_seq[count_seq] = free5GRAN::INTERLEAVING_PATTERN[m] - (K_max - K);
      count_seq++;
    }
}
// De-interleaving
// c_p is c' sequence
// output_bits is c sequence
for (int k = 0; k < K; k++) {
    output_bits[pi_seq[k]] = c_p[k];
}
```

A.8.2 Polar encoding

A.8.2.1 Computing u sequence (selecting the most reliable positions of the channel)

A.8.2.1.1 Encoding: The first step for polar encoding is to compute an intermediate sequence u following the algorithm detailed in TS38.212 section 5.3.1.2. This procedure is based on a sequence $\bar{Q}_I^N$. This sequence can be computed based on the algorithm described in TS38.212 section 5.4.1.1. It contains the most reliable positions of the channel. This process is defined in the rate matching section as bit positions might be changed by puncturing or shortening (see section 4.4.1). Anticipating the rate matching process enables to know what will be the most reliable positions after rate matching. Sequence u takes bits from the c' sequence and places them at the most reliable position of the channel.

The code snippet below shows how to compute $\bar{Q}_I^N$. For performances reasons, all the possible

$\bar{Q}^N_{I,tmp}$ sequences are pre-computed at initialization, which leads to a pretty simple algorithm.

```
// POLAR DECODING: Q_I_N COMPUTATION
// First, select the pre-computed q_itmp_n sequence that corresponds to the current case.
// polar_decoding_struct contains all the possible values of q_itmp_n.
// The different cases are:
// - Q_ITMP_N_CASE_0[E] for E >= N (there is more space in the allocation than
// in the coded block, rate matching will perform repetition).
// - Q_ITMP_N_CASE_1[E] for E < N and K / E <= 7 / 16 (There is too much bits to
// be sent regarding the allocation size, future rate matching will perform puncturing)
// - E < N and K / E > 7 / 16 : not supported (There is too much bits to be sent
// regarding the allocation size, future rate matching will perform shortening)
// E is the payload size after rate matching
// K is the payload size after CRC attachment
if (E < N) {
    if ((float)K / (float)E <= 7.0 / 16.0) {
      q_itmp_n = polar_decoding_struct->Q_ITMP_N_CASE_1[E];
    } else {
      // The shortening case is not supported
      cout << "POLAR CODE PARAM NOT SUPPORTED" << endl;
    }
} else {
    q_itmp_n = polar_decoding_struct->Q_ITMP_N_CASE_0;
}
// n_pc is the number of parity bits
// Take the (K + n_pc) most reliable elements of q_itmp_n,
// which corresponds to the (K + n_pc) last elements
for (int n = 0; n < K + n_pc; n++) {
    q_i_n[n] = q_itmp_n[q_itmp_n.size() - (K + n_pc) + n];
}
```

Once $\bar{Q}^N_I$ is computed, Algorithm 12 can be used to compute u sequence.

Algorithm 12 Polar encoding: computing u sequence

1: $k = 0$
2: **for** $n \in [0, N-1]$ **do**
3: **if** $n \in \bar{Q}^N_I$ **then**
4: $u_n = c'_k$
5: $k = k + 1$
6: **end if**
7: **end for**

A.8.2.1.2 Decoding: In the decoding direction, recovering c' from u can be done using Algorithm 12 by replacing $u_n = c'_k$ by $c'_k = u_n$ in line 4.

A.8.2.2 Computing output sequence d (polar transform)

A.8.2.2.1 Encoding: Finally, polar transform can be applied to u with following Equation:

$$d = u \cdot G_N$$

Where G_N is the n-th Kronecker power of G_2, with $N = 2^n$ and :

$$G_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

For example:

$$G_4 = \begin{pmatrix} 1 \cdot \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} & 0 \cdot \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \\ 1 \cdot \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} & 1 \cdot \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

A.8.2.2.2 Decoding: In the decoding direction, successive cancellation algorithm introduced in section 4.3.1.1 has not been implemented yet. In the current version, u sequence is recovered from d using equation:

$$u = d \cdot G_N^{-1}$$

This method does not enable to correct errors in the channel but works well if there are no errors. It is used in the first version as the physical layer implementation is tested in a Faraday cage, which is a noise-limited system and where the channel's quality is good. However, a successive cancellation algorithm must be implemented for the PHY layer to work in real environments, which are interference-limited systems, and where the channel's quality is bad.

A.9 Rate matching

A.9.1 Rate matching for polar codes *(Transport channel processing - TS38.212 section 5.4.1)*

Rate matching for polar codes is a three-step process made of a first sub-block interleaving, followed by the bit selection and second bits interleaving. In this section, we consider only the case where $I_{BIL} = 0$, which means that the last interleaving is deactivated.

A.9.1.1 Sub-block interleaving *(TS38.212 section 5.4.1.1)*

The first rate matching function is the sub-block interleaver. The rate matching input sequence is d and the interleaver output is y. The details about interleaving procedure are given is TS38.212 section 5.4.1.1. In the decoding direction, d can be recovered from y by replacing $y_n = d_{J(n)}$ by $d_{J(n)} = y_n$. The following code snippet is the de-interleaver from *free5GRAN* receiver:

```
// RATE RECOVERING FOR POLAR CODES: DE-INTERLEAVING
// N is the payload size after polar coding
// output_bits is the output of rate recovering process
// y is the input to de-interleaving function
for (int n = 0; n < N; n++) {
    int i = floor(32 * (double)n / (double)N);
    // free5GRAN::SUB_BLOCK_INTERLEAVER_PATTERN is TS38.212 Table 5.4.1.1-1
    int j_n = free5GRAN::SUB_BLOCK_INTERLEAVER_PATTERN[i] * N / 32 + n % (N / 32);
    // output_bits is d sequence
    output_bits[j_n] = y[n];
}
```

A.9.1.2 Bit selection *(TS38.212 section 5.4.1.2)*

The bit selection function is the core of the rate matching process as it is where the bits are actually selected if $N > E$ or repeated if $E > N$ (where E is the output size of rate matching and N is the input size). The method is described in TS38.212 section 5.4.1.2. In the decoding direction, rate recovering is done in a slightly different way, as repetition consists in taking the first sequence occurrence, and on the opposite, punctured bits cannot be recovered and are set to 0. The receiver

implementation illustrates the rate recovering process.

```
// RATE RECOVERING FOR POLAR CODES: BITS DE-SELECTION
// N is the payload size after polar coding
// E is the payload size after rate matching
// K is the payload size after CRC attachment
// e is the input of rate recovering function
// y is the output of bit de-selection function
// In case repetition was done at the transmitter side,
// Take only the first repetition occurence
if (E >= N) {
    for (int n = 0; n < N; n++) {
      y[n] = e[n];
    }
} else {
    // In case puncturing was done at the transmitter side,
    // non-transmitted bits are set to 0.
    if ((float)K / (float)E <= 7.0 / 16.0) {
      for (int n = 0; n < N - E; n++) {
        y[n] = 0;
      }
      // Transmitted bits are recovered
      for (int n = 0; n < E; n++) {
        y[n + N - E] = e[n];
      }
    }
    // Other case is not supported
}
```

A.9.2 Rate matching for LDPC *(Transport channel processing - TS38.212 section 5.4.2)*

A.9.2.1 Bits selection *(TS38.212 section 5.4.2.1)*

The two first steps for bits selection are to compute N_{cb}, the size of the rate matching input internal buffer and E_r, the output size for the current code block. N_{cb} can be either the input size N when Limited Buffer Rate Matching (LBRM) is not activated or the minimum between N and N_{ref} when LBRM is activated. An introduction on the LBRM concept can be found in [73]. In a default configuration, $N_{ref} = 25344$ which is the maximum code block size ($N = 66 \cdot Z_c$ for BG 1, and the maximum possible Z_c is 384). The way E_r is computed is defined in the standard (TS38.212 section 5.4.2.1 [11]).

Then, the bit selection process itself can be performed. The index of the first element to be transmitted in the input sequence is given by k_0, which depends on N_{cb}, Z_c (LDPC lifting size), RV_{ID} (HARQ redundancy version) and base graph. The E_r bits of d after index k_0 are copied to temporary output e and filler bits are pruned.

Here is the *free5GRAN* code snippet for bits recovering at the receiver's side after bit

de-interleaving has been performed:

```
// RATE RECOVERING FOR LDPC: BITS DE-SELECTION
// N_cb is the size of the internal buffer
// E is the payload size after rate matching
// N is the code word size after channel coding
// K is the code word size after segmentation
// Zc is the lifting size
// K_p is the number of information bits in the code word
// k0 is the first index of the buffer which is transmitted
// with current  redundancy version
// select k0
// graph is the BG to be used
int k0;
if (id_rv == 0) {
    k0 = 0;
} else if (id_rv == 1) {
    k0 = (graph == 1) ? floor((17.0 * N_cb) / (66.0 * Zc)) * Zc
                      : floor((13.0 * N_cb) / (50.0 * Zc)) * Zc;
} else if (id_rv == 2) {
    k0 = (graph == 1) ? floor((33.0 * N_cb) / (66.0 * Zc)) * Zc
                      : floor((25.0 * N_cb) / (50.0 * Zc)) * Zc;
} else if (id_rv == 3) {
    k0 = (graph == 1) ? floor((56.0 * N_cb) / (66.0 * Zc)) * Zc
                      : floor((43.0 * N_cb) / (50.0 * Zc)) * Zc;
} else {
    cout << "LDPC RATE RECOVERING PARAM NOT SUPPORTED" << endl;
}
// output_sequence is the output of rate recovering
// Initialize output with 0
for (int i = 0; i < N; i++) {
    output_sequence[i] = 0;
}
// e is the input of this function
// Actual bits recovering
int j = 0;
int k = 0;
while (k < E && j < N_cb) {
    // Condition (j < N_cb) is added as at the transmitter side,
    // bits are circularly repeated for j greater than N_cb
    int index = (k0 + j) % N_cb;
    // From TS 38.212 section 5.2.2, NULL bits are placed in positions
    // (K_p - 2 * Zc) to (K - 2 * Zc)
    if (index >= K_p - 2 * Zc && index < K - 2 * Zc) {
      // if NULL bit, replace with 0
      // As manipulated bits are soft bits,
      // 0 is replaced by infinite
      output_sequence[index] = numeric_limits<double>::infinity();
    } else {
      // else, recover bit
      output_sequence[index] = e[k];
      k++;
    }
    j++;
}
```

A.9.2.2 Bits interleaving *(TS38.212 section 5.4.2.2)*

Interleaving is performed to mix the input and parity bits and spread them across the whole buffer. The process is defined in TS38.212 section 5.4.2.2 [11]. Here is the code implemented in *free5GRAN*

for the receiver de-interleaving:

```
// RATE RECOVERING FOR LDPC: RECEIVER DE-INTERLEAVING
// E is the payload size after rate matching
// i_lbrm is whether or not LBRM is activated
// mod_order is the modulation scheme used for transmission
// Compute rate matching internal buffer size
int N_cb = (i_lbrm == 0) ? N : min(N, 25344);
// e is the output sequence of de-interleaving
double e[E];
int E_Q = (int)((float)E / (float)mod_order);
// input_bits is the input of rate matching
// Input de-interleaving
for (int j = 0; j < E_Q; j++) {
    for (int i = 0; i < mod_order; i++) {
      e[i * (E_Q) + j] = input_bits[i + j * mod_order];
    }
}
```

A.10 Soft modulation de-mapping

Soft modulation de-mapping is the process by which a sequence of soft bits is recovered from a sequence of received IQ samples. Here is the code for 16-QAM soft modulation de-mapping in *free5GRAN*:

```
// 16-QAM SOFT DEMODULATION
// The complex plane is split in squares around each constellation point, this forms a grid
// ref2 is the width of a square
// 1000 is the resolution
ref2 =  2.0 * 1000.0 / sqrt(10);
// For each IQ sample
for (int i = 0; i < signal_length; i++) {
  // Take the i and j indexes in the grid (the grid origin (0, 0)
  // corresponds to the lowest I and Q values)
  // max and min functions bound the indexes between 0 and the size of the grid
  // num_linear_points is the size of the grid
  // signal is the input frequency-domain signal
  index_i = max(min((int)(real(signal[i]) * 1000 + ref2 * 2) / ref2,
                    (num_linear_points - 1)), 0);
  index_j = max(min((int)(imag(signal[i]) * 1000 + ref2 * 2) / ref2,
                    (num_linear_points - 1)), 0);

  // Take the corresponding square in pre-computed structure
  free5GRAN::modulation_point_info &closest_point =
      free5GRAN::QAM_16_SCHEME[index_i][index_j];
  // free5GRAN::QAM_16_SCHEME contains, for each square of the grid
  // - The closest point position (position) and bit values (bit_value)
  // - The closest points (neighbor_points) for each alternate bit value
  //       (if bit value is 1, gives closest point with bit value 0)

  // Compute the distance between the received IQ sample and the closest point
  // (it can be d0 or d1, depending on the constellation point)
  min_dist = norm(signal[i] - closest_point.position);

  // modulation_order is the number of bits per IQ sample
  // For each bit
  for (int m = 0; m < modulation_order; m ++){
    // Compute distance from the received sample to the alternate position
    // (it can be d1 or d0, depending on the constellation point)
    other_dist = norm(signal[i] - closest_point.neighbor_points[m]);
    // Compute the difference between the two distances
    // (d1 - d0) or (d0 - d1)
    // soft_bits is the output soft bits sequence
    soft_bits[modulation_order * i + m] = other_dist - min_dist;
    if (closest_point.bit_value[m] == 1){
      // If the closest point's bit value is 1, then:
      // - min_dist is d1
      // - other_dist is d0
      // So (other_dist - min_dist) is (d0 - d1)
      // Reverse to obtain (d1 - d0)
      soft_bits[modulation_order * i + m] = - soft_bits[modulation_order * i + m];
    }
  }
}
```

A.11 Channel estimation

Channel estimation enables the receiver to estimate the effect of noise and interference over the channel. It is based on DMRS signal. Here is the code for channel estimation in *free5GRAN*:

```
// CHANNEL ESTIMATION
// pilot_indexes is an array that contains the position of the pilots REs in OFDM grid
// pilot_indexes[0][p] is the symbol position of RE p
// pilot_indexes[1][p] is the subcarrier position of RE p
// pilots contains the received DMRS IQ samples
// reference_pilot contains the expected DMRS IQ samples

// Initializing variables
// found_indexes will contain, for each symbol, the DMRS positions
vector<vector<int>> found_indexes(num_symbols);
// symbols_with_pilot will contain the symbols where there is pilot
// sc_with_pilots will contain the subcarriers with pilot
vector<int> symbols_with_pilot, sc_with_pilots;
// variables used later
int upper_index, lower_index, step, step_width;
float re_int, im_int;

// is_dmrs_symb will contain whether or not a symbol contains DMRS
// is_dmrs_sc will contain whether or not a subcarrier contains DMRS
// Those two vectors are initialized to false
vector<bool> is_dmrs_symb(num_symbols, false);
vector<bool> is_dmrs_sc(num_sc, false);
// Computing pilots coefficients h_ij
// For each DMRS sample
for (int i = 0; i < pilot_size; i++) {
    // Compute corresponding channel coefficient
    // coefficients contains the output matrix of channel estimates
    coefficients[pilot_indexes[0][i]][pilot_indexes[1][i]] =
        pilots[i] / reference_pilot[i];
    // Store that subcarrier pilot_indexes[1][i] of symbol pilot_indexes[1][0]
    // is a DMRS RE
    found_indexes[pilot_indexes[0][i]].push_back(pilot_indexes[1][i]);
    // There is at least on DMRS RE in symbol pilot_indexes[0][i]
    is_dmrs_symb[pilot_indexes[0][i]] = true;
    // There is at least on DMRS RE in subcarrier pilot_indexes[1][i]
    is_dmrs_sc[pilot_indexes[1][i]] = true;
    // Push pilot_indexes[0][i] in the list of symbols with DMRS, if not done yet
    if (find(symbols_with_pilot.begin(), symbols_with_pilot.end(),
             pilot_indexes[0][i]) == symbols_with_pilot.end()){
      symbols_with_pilot.push_back(pilot_indexes[0][i]);
    }
    // Push pilot_indexes[1][i] in the list of subcarriers with DMRS, if not done yet
    if (find(sc_with_pilots.begin(), sc_with_pilots.end(),
             pilot_indexes[1][i]) == sc_with_pilots.end()){
      sc_with_pilots.push_back(pilot_indexes[1][i]);
    }
}
// num_sc is the number of subcarriers in the grid
// Frequency domain linear interpolation
for (int sc = 0; sc < num_sc; sc++) {
    // If the current resource element is not a pilot
    if (!is_dmrs_sc[sc]) {
      // Get lower and upper pilot indexes for interpolating. If no lower,
      // take the two closest upper pilots and respectively if no upper
      // pilot
      if (sc < sc_with_pilots[0]) {
        lower_index = sc_with_pilots[0];
        upper_index = sc_with_pilots[1];
      } else if (sc > sc_with_pilots[sc_with_pilots.size() - 1]) {
        lower_index = sc_with_pilots[sc_with_pilots.size() - 2];
```

```
        upper_index = sc_with_pilots[sc_with_pilots.size() - 1];
      } else {
        for (int k = 0; k < sc_with_pilots.size() - 1; k++) {
          if (sc_with_pilots[k] < sc && sc_with_pilots[k + 1] > sc) {
            lower_index = sc_with_pilots[k];
            upper_index = sc_with_pilots[k + 1];
            break;
          }
        }
      }
      // Compute the number of subcarriers between the current one
      // and the lower DMRS subcarrier
      step = sc - lower_index;
      // Compute number of subcarriers between upper and lower DMRS subcarrier
      step_width = upper_index - lower_index;

      // For all the symbols that have DMRS
      for (int i = 0 ; i < num_symbols; i ++){
        if (is_dmrs_symb[i]){
          // Interpolate
          coefficients[i][sc] = complex<float>(step, 0) * (coefficients[i][upper_index] -
                coefficients[i][lower_index]) / complex<float>(step_width, 0) +
                                    coefficients[i][lower_index];
        }
      }
    }
}
// num_symbols is the number of symbols in the OFDM grid
// Until now, all the subcarriers of all the symbols that have DMRS RE are done
// Now, compute all the subcarriers of all the symbols that do not carry DMRS
for (int s = 0 ; s < num_symbols; s ++) {
    // If the symbol does not carry DMRS
    if (!is_dmrs_symb[s]){
      // Get lower and upper pilot indexes for interpolating. If no lower,
      // take the two closest upper pilots and respectively if no upper
      // pilot
      if (s < symbols_with_pilot[0]) {
        lower_index = symbols_with_pilot[0];
        upper_index = symbols_with_pilot[1];
      } else if (s > symbols_with_pilot[symbols_with_pilot.size() - 1]) {
        lower_index = symbols_with_pilot[symbols_with_pilot.size() - 2];
        upper_index = symbols_with_pilot[symbols_with_pilot.size() - 1];
      } else {
        for (int k = 0; k < symbols_with_pilot.size() - 1; k++) {
          if (symbols_with_pilot[k] < s && symbols_with_pilot[k + 1] > s) {
            lower_index = symbols_with_pilot[k];
            upper_index = symbols_with_pilot[k + 1];
            break;
          }
        }
      }
      // Compute the number of subcarriers between the current one and the lower DMRS subcarrier
      step = s - lower_index;
      // Compute number of subcarriers between upper and lower DMRS subcarrier
      step_width = upper_index - lower_index;
      // For all the subcarriers of the current symbol
      for (int sc = 0; sc < num_sc; sc++) {
        // Interpolate
        re_int = step * (real(coefficients[upper_index][sc]) -
                    real(coefficients[lower_index][sc])) /
```

```
                    step_width + real(coefficients[lower_index][sc]);
        im_int = step * (imag(coefficients[upper_index][sc]) -
                    imag(coefficients[lower_index][sc])) /
                    step_width + imag(coefficients[lower_index][sc]);
        coefficients[s][sc] = complex<float>(re_int, im_int);
      }
    }
}
```

A.12 Channel de-mapping

Channel de-mapping enables the receiver to extract IQ samples from the OFDM grid according to resource allocation. *free5GRAN* implementation of the channel de-mapping:

```
// input_signal is the input OFDM grid
// output_channels is a 2D array containing the physical channels
// output_indexes is a 3D array containing the positions of each sample of the
// physical channels (3 dimensions = physical channel, symbol and subcarrier)
// ref is the reference grid which contains the position of the physical channel samples
// Loop over each symbol in the OFDM grid
for (int symbol = 0; symbol < num_symbols; symbol++) {
    // Loop over each subcarrier of the OFDM grid
    // symbol and sc are pointing to a precise RE
    for (int sc = 0; sc < num_sc; sc++) {
        // Loop over each physical channel
        for (int channel = 0; channel < num_channels; channel++) {
            // If the current resource element belongs to the physical channel,
            // store the element and corresponding grid index
            if (ref[channel][symbol][sc] == 1) {
              // Add the current RE sample to the physical channel
              output_channels[channel][channel_counter[channel]] =
                  input_signal[symbol][sc];
              // Save the symbol and subcarrier position
              // of the sample for future processing
              output_indexes[channel][0][channel_counter[channel]] = symbol;
              output_indexes[channel][1][channel_counter[channel]] = sc;
              // Increment the channel counter,
              // which counts the number of elements
              // in each physical channel.
              channel_counter[channel]++;
            }
        }
    }
}
```

A.13 OFDM demodulation

The OFDM demodulation is the receiver's side function by which the time domain signal is transformed to OFDM grid. It includes cyclic prefix deletion and FFT. Implementation of the OFDM

demodulation in *free5GRAN*:

```
// OFDM DEMODULATION
// time_domain_signal is the time domain input signal
// output_signal is the output OFDM grid
// Loop over all the symbols of the signal
for (int symbol = 0; symbol < num_symbols; symbol++) {
    // Compute symbol index
    int symb_index = (first_symb_index + symbol)
        % free5GRAN::NUMBER_SYMBOLS_PER_SLOT_NORMAL_CP;
    // Filling fft input signal with current symbol IQ
    // Perform cyclic prefix deletion by only extracting
    // the data part of a symbol
    // (take the fft_size elements of the symbol
    // shifted by the symbol CP = cp_lengths[symb_index])
    for (int i = 0; i < fft_size; i++) {
        fft_in[i][0] = real(time_domain_signal[i + offset +
            cum_sum_symb[symb_index] + cp_lengths[symb_index]]);
        fft_in[i][1] = imag(time_domain_signal[i + offset
            + cum_sum_symb[symb_index] + cp_lengths[symb_index]]);
    }
    // Execute the fft
    fftw_execute(fft_plan);
    // Recover RE grid from FFT output
    // In FFTW3, positive and negative
    // frequencies are twisted
    for (int i = 0; i < num_sc_output / 2; i++) {
        output_signal[symbol][num_sc_output / 2 + i] =
            complex<float>(fft_out[i][0], fft_out[i][1]);
        output_signal[symbol][num_sc_output / 2 - i - 1] =
            complex<float>(fft_out[fft_size - i - 1][0],
                fft_out[fft_size - i - 1][1]);
    }
}
```

Résumé en français

Introduction

Alors que la 4G se concentre sur l'accès à internet, la 5G s'adresse à un large éventail de cas d'utilisation (appelés verticaux), allant de l'industrie 4.0 aux villes intelligentes et à la réalité augmentée. Ces verticaux peuvent être regroupés en trois cas d'utilisation principaux : eMBB, uRLLC et mMTC. L'agrégation de ces cas d'utilisation sur une seule infrastructure physique tout en respectant les contraintes associées est nécessaire pour optimiser les déploiements et la consommation des ressources. Les différents cas d'utilisation ont des exigences réseau antagonistes. En effet, alors que eMBB se concentre sur le débit, uRLLC se concentre sur la latence, et mMTC doit supporter un grand nombre de dispositifs connectés. Ces contraintes, et plus particulièrement la latence et le débit, ne sont pas compatibles car l'augmentation de la capacité implique souvent une perte de délais, et réciproquement, les systèmes axés sur la latence peinent à fournir des débits élevés. Dans ces conditions, il faut s'attendre à ce que des stations de base ne soient pas en mesure de répondre à des contraintes de différents verticaux par défaut. D'autre part, le déploiement d'infrastructures physiques indépendantes pour chaque verticale aura un coût économique et écologique insupportable.

Dans ce contexte, une nouvelle technologie est nécessaire pour prendre en charge les terminaux hétérogènes sur une infrastructure unique. Cette technologie s'appelle le slicing du réseau et décrit la capacité d'un réseau à exploiter différents verticaux et cas d'utilisation, appelés slices, sur une seule infrastructure physique. Un slice représente un réseau logique qui relie des terminaux ayant des contraintes de réseau proches à un ou plusieurs réseaux de données. Elle est associée à un ensemble de ressources. En fonction des terminaux qui connectent une tranche, un ensemble d'exigences et de contraintes (comme le débit, le retard ou la fiabilité) est défini.

Cette technologie devrait fournir deux caractéristiques essentielles pour gérer les différents usages. La première est l'optimisation et la flexibilité du déploiement, car elle permettrait de déployer plusieurs verticaux sur une infrastructure physique partagée. En outre, elle permettrait un gain en consommation d'énergie et en coût, qui sont les deux principales préoccupations lors du déploiement de réseaux mobiles. D'autre part, le slicing du réseau doit également garantir l'isolation entre les tranches afin que les différentes tranches soient respectées.

Du côté du RAN, on peut souligner qu'il existe un manque considérable de technologies pour le slicing du réseau. Aucun algorithme ou méthode garantissant l'isolation entre les slices et le respect des exigences n'a été proposé. En particulier, les systèmes qui gèrent les communications à haut débit et à faible latence manquent. Le travail présenté dans cette thèse émerge de ce constat et se concentre sur le côté RAN. En effet, au niveau RAN, le slicing du réseau nécessite la conception de nouvelles méthodes et algorithmes et l'application des technologies de virtualisation aux systèmes RAN, ce qui n'a jamais été fait auparavant. Dans l'état actuel, on peut supposer que le slicing du réseau sera réalisé par le provisionnement et l'association de ressources à des tranches ou à un groupe de tranches partageant les mêmes exigences en termes de latence et de débit. Dans ce contexte, l'allocation des ressources semble être critique pour l'optimisation du découpage du réseau afin de maximiser l'efficacité du réseau tout en respectant les exigences des tranches.

Cette thèse aborde deux défis principaux soulevés par le slicing du réseau du RAN. Le premier est que l'allocation des ressources nécessite la définition de modèles de réseau qui considèrent des contraintes RAN diverses et hétérogènes. Au-delà de la capacité et de la densité UE, qui peuvent être facilement modélisées, la latence et la fiabilité sont des paramètres critiques pour de tels modèles et ne peuvent être étudiés de manière triviale. De nombreux modèles existent déjà pour l'allocation des ressources du RAN. Cependant, les modèles qui prennent en compte les contraintes de découpage du réseau n'existent pas. De plus, l'impact du slicing du réseau sur la couverture n'a pas encore été étudié, alors qu'il s'agit d'un aspect critique de la fiabilité du réseau. Une des contributions de cette thèse est donc de proposer un modèle de slicing RAN. Il prend en compte les nouvelles contraintes apportées par le slicing, parmi lesquelles la couverture est de la plus haute importance.

Le deuxième grand défi soulevé par le slicing du réseau est la supervision de l'allocation des ressources. En effet, tant que l'infrastructure physique est partagée entre les différentes slices, il est impossible de garantir un contrôle complet du réseau et le respect des exigences, même si les ressources sont soigneusement allouées. Il s'agit d'un problème important pour le slicing du réseau, car certains verticaux ont des exigences très élevées en matière de contrôle du réseau. Dans ce contexte, des

systèmes de supervision sont nécessaires pour apporter un second niveau de contrôle sur le réseau. Les outils de supervision visent à contrôler que les contraintes des différents slices peuvent être respectées en fonction de l'état actuel du système grâce à l'analyse des données extraites du réseau. Les données peuvent être fournies par les API du réseau, mais cela n'est pas fiable concernant les cyber-attaques potentielles. Dans cette thèse, nous proposons d'utiliser un système d'extraction de données externe au réseau (appelé sonde). Par conséquent, une autre contribution de cette thèse est l'implémentation d'un prototype d'une telle sonde.

Étant donné que la sonde est à l'extérieur du réseau, elle doit décoder les différents signaux d'une cellule afin d'extraire des données. Le mécanisme d'extraction de données qui doit être mis en œuvre correspond au côté récepteur de la couche PHY d'un UE pour le décodage des transmissions descendantes et au côté récepteur de la couche PHY d'un gNodeB pour le décodage des transmissions montantes. Par conséquent, la sonde peut être dérivée d'une couche PHY 5G. La dernière contribution de ce travail est l'introduction d'une couche physique 5G open-source à partir de laquelle la sonde est dérivée. Cette couche physique est conçue en fonction de deux caractéristiques principales. La première est le slicing du réseau, ce qui implique que l'implémentation de la couche PHY doit être modulaire afin de pouvoir s'adapter à différents types de slices. La seconde est le découpage fonctionnel. En effet, en 5G, les différentes couches du RAN peuvent être divisées et désagrégées sur le réseau. La couche PHY est donc construite avec une architecture qui permet la mise en œuvre future du fractionnement RAN. Ce projet est nommé *free5GRAN* et est disponible publiquement sur Github [7].

Introduction aux réseaux 5G

Le système 5G peut être présenté de différentes manières. Dans ce chapitre, il est présenté comme l'infrastructure réseau conçue pour supporter une grande variété de terminaux. En effet, en 4G, le réseau est construit pour fournir aux utilisateurs finaux des services de données comme le streaming vidéo ou le trafic web, alors qu'en 5G, le réseau est construit pour servir des terminaux très hétérogènes comme les véhicules connectés, les capteurs ou les smartphones. La principale différence entre les différents trafics d'utilisateurs n'est pas le profil QoS associé mais plutôt le réseau de données auquel le service nécessite un accès. En effet, une voiture connectée peut nécessiter un accès au réseau interne du constructeur, tandis qu'un capteur pour l'agriculture peut se connecter au cloud pour la collecte de données. Par conséquent, en 5G, le trafic utilisateur est d'abord traité en fonction du réseau de données auquel un terminal doit accéder, puis en fonction du profil QoS requis par le service. Lors de l'accès au réseau, un terminal est associé avec un tunnel qui le relie directement au réseau de données. Le réseau est responsable de la mise en place du tunnel et de la prise en charge de différents services avec différents profils QoS au sein d'un même tunnel.

Outre la diversité des terminaux, la 5G permet de répondre à de nombreux cas d'utilisation. Ces utilisations sont une combinaison de trois cas d'utilisation fondamentaux qui sont eMBB, uRLLC et mMTC. Premièrement, eMBB est l'usage le plus courant et correspond à l'évolution naturelle des réseaux 4G. Il est conçu pour permettre aux utilisateurs finaux d'accéder au réseau avec un débit élevé, rendant possibles des applications telles que la réalité augmentée. uRLLC est conçu pour prendre en charge des applications critiques telles que la chirurgie à distance ou les véhicules autonomes. Ces applications nécessitent une grande fiabilité du réseau et des communications à faible latence. Enfin, mMTC est un cas d'utilisation où le réseau est utilisé pour connecter des millions de dispositifs. Le réseau doit être capable de desservir un nombre énorme de UEs, et la communication doit être hautement optimisée en termes de consommation d'énergie. Ces trois cas d'utilisation fondamentaux délimitent un triangle dans lequel se trouvent tous les autres cas d'utilisation. Tous les cas d'utilisation de la 5G peuvent être considérés comme une combinaison de ces trois cas d'utilisation fondamentaux. Le triangle et les différents cas d'utilisation sont représentés dans la figure A.6. Tous les terminaux connectés par la 5G sont associés à un cas d'utilisation.

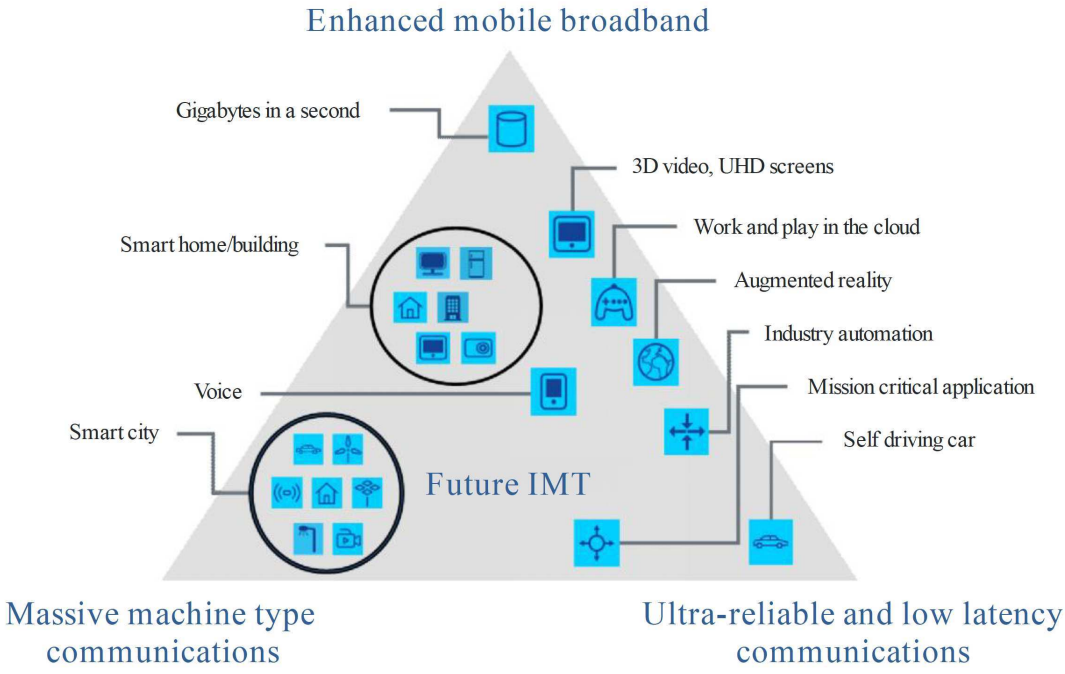


Figure A.6: Cas d'usages 5G (ITU-R IMT 2020 requirements)

Procédures de synchronisation de la voie descendante

La synchronisation de la liaison descendante entre le gNodeB et les UEs est le premier mécanisme qui doit être mis en œuvre côté UEs. Elle est nécessaire car les différents composants de la cellule n'ont pas la même horloge interne. Par conséquent, une synchronisation descendante est nécessaire pour synchroniser l'horloge interne de UEs avec celle de gNodeB.

Tout d'abord, l'UE doit déterminer quels sont les symboles, les slots et les limites des trames afin de récupérer la grille OFDM qui permet à l'UE d'extraire et de décoder les données. De plus, dans les systèmes OFDMA, les ressources sont allouées en temps et en fréquence, et la synchronisation permet également aux UEs de récupérer le timing du réseau avant de recevoir les allocations de ressources. Enfin, comme l'horloge interne du gNodeB et celle des UEs sont différentes, la fréquence de transmission des deux composants peut être légèrement différente, et la synchronisation est utilisée pour déterminer et compenser le décalage de fréquence entre le gNodeB et le UE.

Ce chapitre présente la procédure de synchronisation de la liaison descendante qui ets basée sur un bloc de signaux appelé SSB, représenté en Figure A.7.

A partir de maintenant, on considère que les UEs sont synchronisés avec la cellule afin qu'ils puissent récupérer la grille OFDM de la cellule, et leur horloge est réglée sur celle du réseau. Cette procédure de synchronisation est effectuée périodiquement, et les UEs ne perdent pas la synchronisation avec la cellule. Après avoir décodé les informations de base de la cellule, les UEs sont prêts à extraire et décoder les données sur la liaison descendante. Ils peuvent effectuer la synchronisation sur la liaison montante et commencer à transmettre des données.

Procédures de transmission de données

Une fois les UEs synchronisés avec la cellule, les données peuvent être transmises entre le gNodeB et les UEs. Cependant, les ressources des domaines temporel et fréquentiel sont partagées entre les différents UEs. Par conséquent, un mécanisme doit être mis en œuvre entre le gNodeB et les UEs afin que le gNodeB puisse notifier aux UEs les ressources allouées pour les transmissions de données. De plus, étant donné que les différents UEs ne se trouvent pas au même endroit dans la cellule, une synchronisation sur la liaison montante doit être effectuée afin que les transmissions de la liaison montante arrivent simultanément à la station de base. Le processus de synchronisation de la liaison

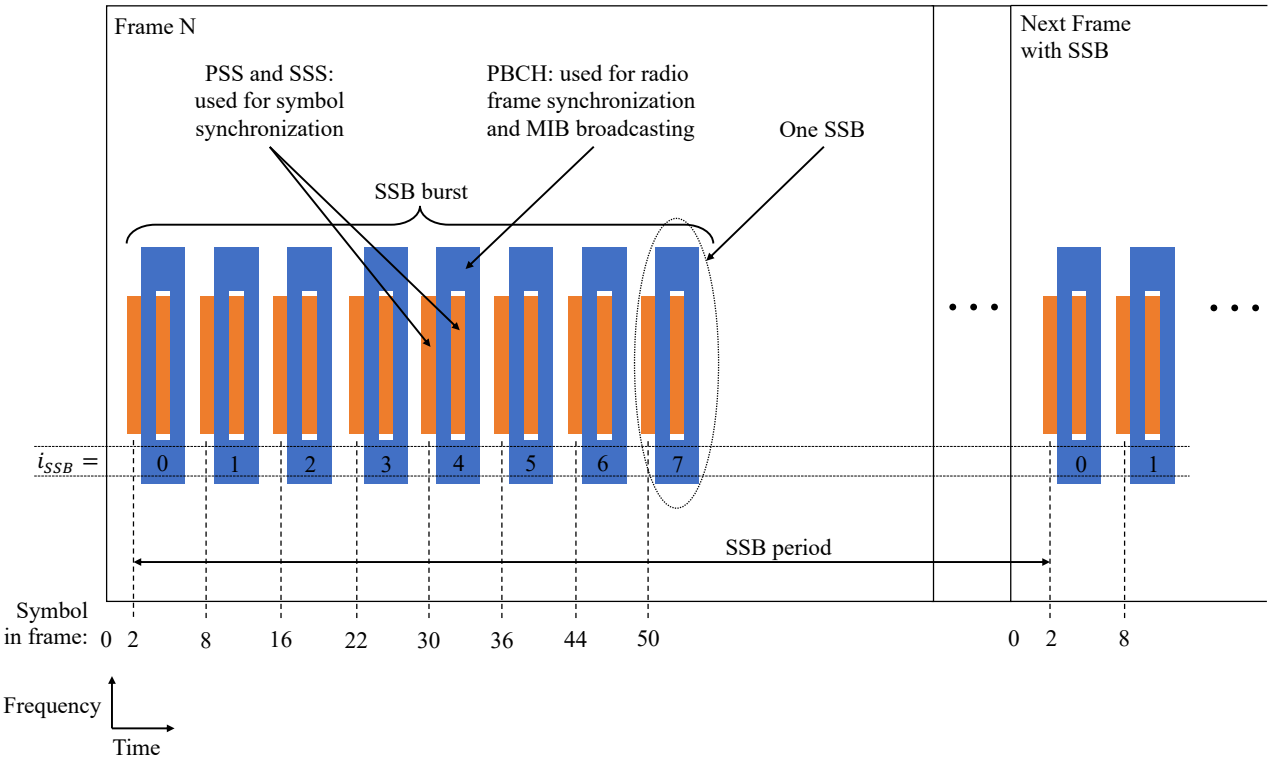


Figure A.7: Bloc SSB

montante est intégré à la procédure RA, exécutée par les UEs pour accéder à la cellule.

La Figure A.8 représente les différents signaux impliqués dans la transmission de données. Le canal PDCCH contient les données de contrôle et les canaux PDSCH et PUSCH contiennent les données utilisateur.

Algorithmes de couche physique

Toutes les procédures mises en œuvre dans les chapitres précédents sont précisément définies dans la norme et permettent au gNodeB et aux UEs de se synchroniser entre eux et d'échanger des données. Cependant, ces procédures reposent sur des fonctions qui ne sont pas précisément définies dans la norme. L'entrée, la sortie et le comportement global des fonctions sont définis, mais les algorithmes mis en œuvre dans les fonctions pour obtenir le comportement normalisé ne sont pas définis. En effet, l'objectif de la norme est de permettre à des composants de différents fabricants de fonctionner ensemble sur un même réseau en définissant les procédures, les mécanismes et le chaînage des fonctions. Néanmoins, la manière dont les fonctions sont mises en œuvre est propre à chaque fabricant, car c'est là que réside la valeur essentielle de la mise en œuvre : les fabricants disposant de meilleurs algorithmes fournissent de meilleurs produits.

Le but de ce chapitre n'est pas de donner une compréhension globale de la norme et des procédures mais de présenter les principales fonctions utilisées à travers les procédures. Pour chaque fonction, nous délimitons ce qui est défini par la norme et ce qui est spécifique au fabricant. De plus, nous proposons des algorithmes simples pour les fonctions non standardisées. Les algorithmes présentés sont ceux utilisés dans *free5GRAN*. Ils ont été sélectionnés pour bien fonctionner dans notre environnement, une cage de Faraday, où les conditions radio sont bonnes. De plus, elles offrent un bon équilibre entre optimisation et lisibilité.

Les principaux algorithmes de couche physique sont :

- Codage cyclic (CRC).
- Codage et décodage canal : codes polaire et LDPC.

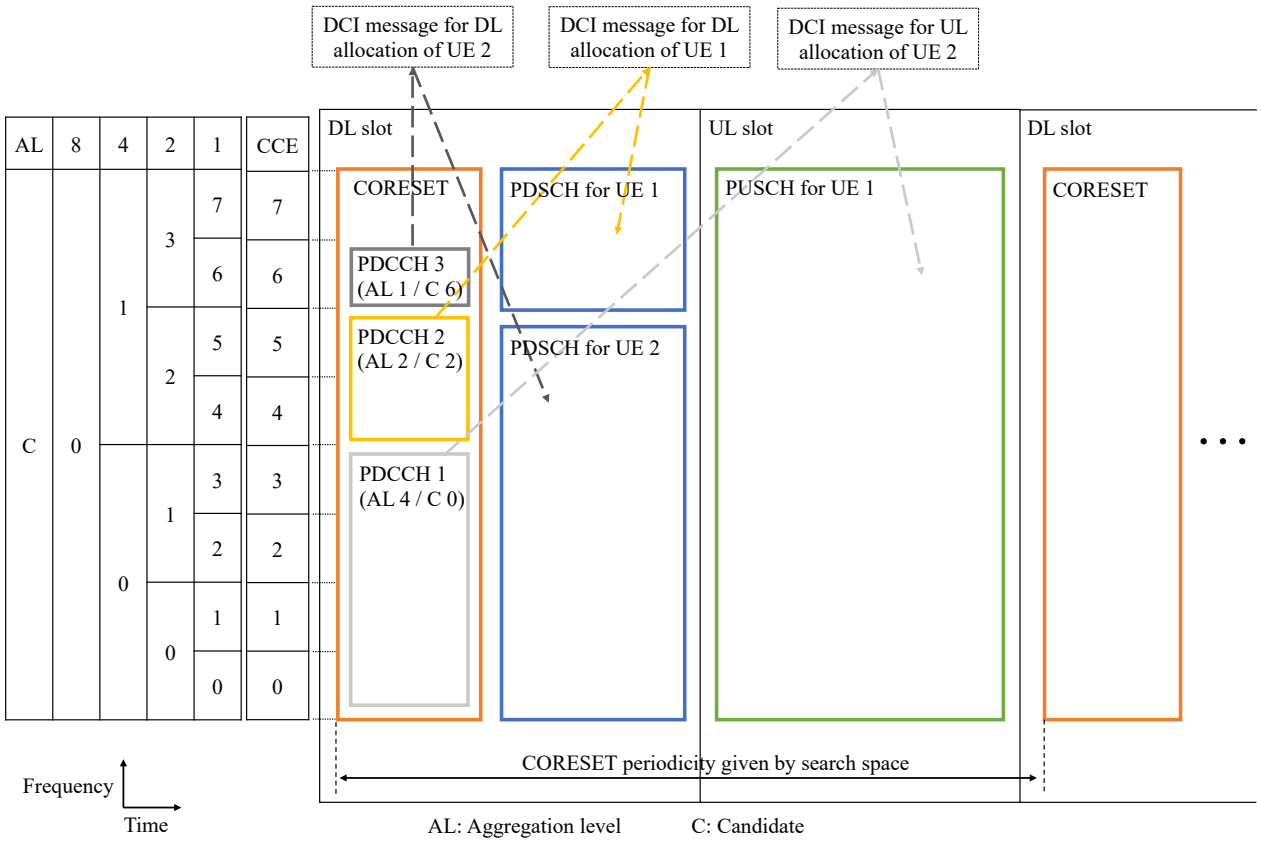


Figure A.8: Transmission de données avec un CORESET de 8 CCEs

- Adaptation de taux.
- Embrouillage.
- Modulation.
- Placement des REs.
- Estimation et égalisation canal.
- Modulation OFDM.

Les algorithmes où résident les plus grands défis sont le décodage du canal, la démodulation et l'estimation du canal. Les algorithmes belief propagation et successive cancellation sont utilisés pour le décodage LDPC et polaire. La démodulation est approximée en ne considérant que les deux points de constellation les plus proches. L'estimation canal se fait par la méthode zero-forcing.

Les autres fonctions restent cruciales, mais il existe un plus petit nombre d'algorithmes possibles, et les performances résident principalement dans l'optimisation algorithmique. Ces algorithmes sont implémentés dans *free5GRAN*. A partir de maintenant, toutes les procédures minimales de la couche physique et les algorithmes associés ont été détaillés.

Architecture logicielle

Ce chapitre présente la structure du projet et l'architecture logicielle qui permet la mise en œuvre de la couche PHY. En effet, une couche PHY est un système qui comprend différentes procédures et algorithmes. Lors de la construction d'un système, comprendre comment lire les normes et mettre en œuvre chaque composant est la moitié du problème, qui a fait l'objet d'une attention particulière dans les chapitres précédents. Comprendre comment créer une structure de code opérationnelle qui peut

être incrémentée composants après composants est l'autre moitié et constitue le cœur de ce chapitre. Différentes architectures peuvent être utilisées. Nous exposons la structure du projet *free5GRAN* et la conception du logiciel pour donner un exemple concret, mais surtout pour montrer les problèmes auxquels la conception doit répondre.

Deux défis principaux doivent être résolus. Le premier est que les systèmes de télécommunication évoluent très rapidement, et que la mise en œuvre n'est pas une tâche en une seule étape mais plutôt un processus itératif où chaque composant peut être mis à jour indépendamment. Par conséquent, le projet doit être structuré de manière à pouvoir être facilement incrémenté. Le deuxième défi principal est que la couche PHY est fortement contrainte. En effet, il s'agit d'une application quasi temps réel avec des exigences de performance élevées, mais certaines fonctions ont une grande complexité algorithmique. L'architecture logicielle doit gérer ces deux contraintes.

La conception logicielle proposée n'est pas monolithique mais utilise largement le multi-threading pour deux raisons. La première est que la norme 5G introduit différentes architectures de station de base, parmi lesquelles le RAN désagrégé reçoit beaucoup d'attention. Par conséquent, même si la mise en œuvre actuelle de la couche PHY ne prend pas en charge le découpage fonctionnel, l'architecture logicielle a été pensée de telle sorte qu'il est facile de dériver l'architecture actuelle pour mettre en œuvre l'option 7.2x (qui est l'option choisie par l'alliance O-RAN). La deuxième raison est liée à la grande hétérogénéité des services qui peuvent être fournis sur le système 5G. En effet, comme certains services peuvent avoir des exigences fortes en termes de fiabilité et d'isolation, la mise en œuvre d'une architecture modulaire permet de dédier des ressources à tel ou tel service. Ainsi, l'architecture actuelle peut être facilement dérivée pour allouer des threads aux services afin de renforcer la fiabilité et le contrôle du traitement de la couche PHY.

Les Figures A.9 et A.10 représentent la structure de la station de base et de l'UE.

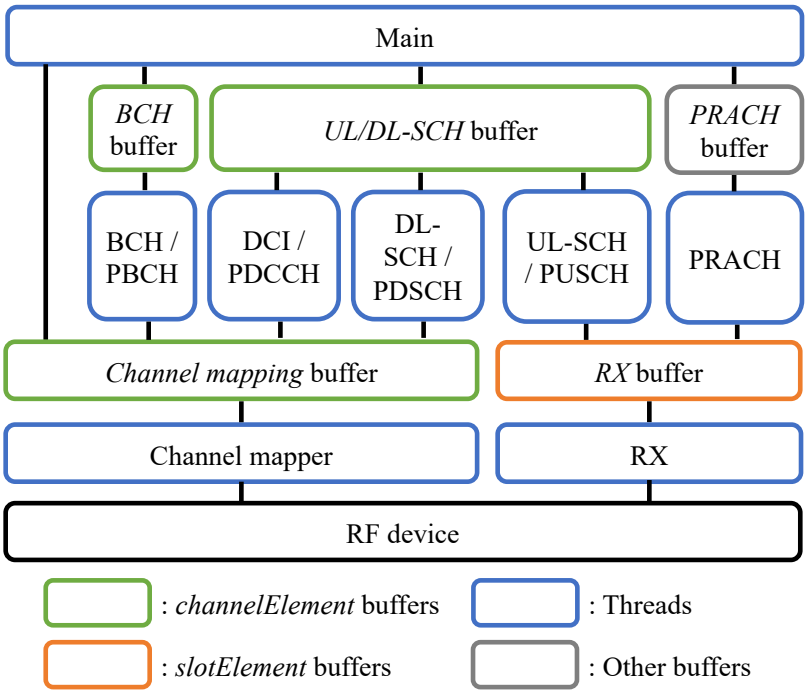


Figure A.9: Structure de la station de base

La structure du projet est construite de manière à ce que les fonctions et les algorithmes puissent être facilement modifiés. Elle permet d'implémenter et de tester de nouveaux algorithmes et également de mettre en œuvre l'évolution future des normes.

L'architecture logicielle est construite de manière modulaire afin que la couche PHY puisse être adaptée aux différents services et slices. De plus, elle s'appuie sur le multithreading, de sorte que l'architecture actuelle peut facilement être modifiée pour mettre en œuvre une station de base prête pour le découpage du RAN.

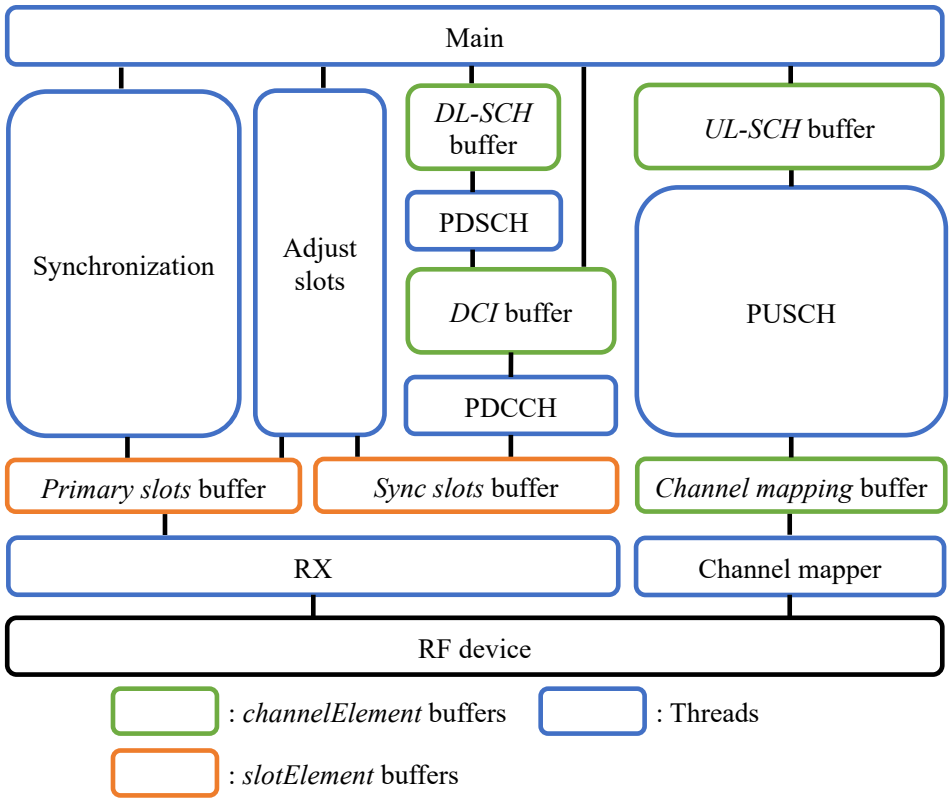


Figure A.10: Structure de l'UE

Architecture RAN pour le slicing du réseau

Le slicing du réseau est l'une des technologies centrales introduites dans la 5G. Elle permet de construire des réseaux logiques au-dessus d'une infrastructure physique unique qui relie les terminaux à un ou plusieurs réseaux de données. Ces réseaux logiques peuvent être personnalisés pour répondre à des exigences de service spécifiques. Les opérateurs utiliseront cette technologie pour adresser de nouveaux marchés. Étant donné que le slicing du réseau est un élément essentiel du système 5G, il a fait l'objet d'une grande attention. Des travaux considérables ont été réalisés pour expliquer comment le slicing peut être utilisé pour aborder de nouveaux verticaux. Cependant, il y a un manque d'informations claires sur la façon dont le slicing peut être mis en œuvre.

Dans la norme, il est défini que les instances de slice sont un ensemble de ressources qui sont associées à un slice. Par conséquent, la mise en œuvre du slicing du réseau réside dans l'allocation des ressources. Cependant, il n'est pas trivial de déterminer les différents types de ressources qui peuvent être utilisés pour la mise en œuvre du slicing. En outre, certaines techniques et technologies doivent être utilisées pour associer les ressources aux slices pour chaque type de ressource.

Ce chapitre se concentre sur la station de base et n'étudie pas le niveau des UE et du CN. Il est considéré qu'un UE n'a pas besoin d'accéder à plusieurs slices simultanément et que le CN est déployé et approvisionné pour prendre en charge les différents réseaux logiques.

La Figure A.11 représente l'architecture RAN proposée pour la mise en place du slicing.

Modélisation du slicing

Comme décrit dans le chapitre précédent, le slicing du réseau repose principalement sur l'association et l'allocation des ressources. De nombreuses solutions ont été proposées dans la littérature pour l'allocation des ressources en fonction des slices. Cependant, les méthodes précédentes ne prennent pas en compte les contraintes spécifiques RAN où la couverture est de la plus haute importance. Comme l'allocation des ressources repose sur la modélisation du système, ce chapitre étudie comment les modèles de réseau peuvent inclure des slices. Au-delà de la capacité et de la densité, qui peuvent

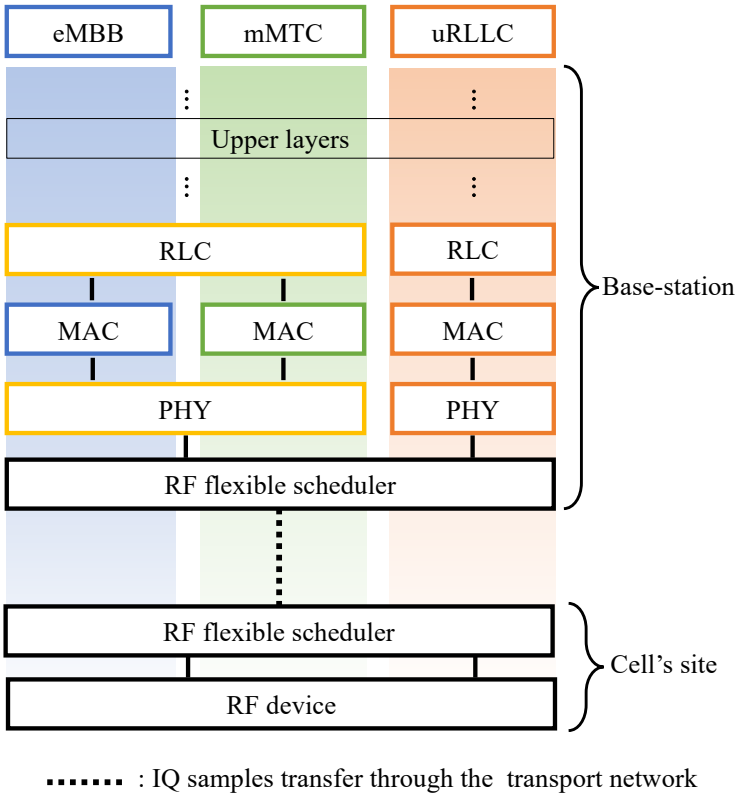


Figure A.11: Architecture RAN proposée

être facilement modélisées, la latence et la fiabilité sont des paramètres critiques pour de tels modèles et ne peuvent être étudiés de manière triviale. La fiabilité est modélisée comme une contrainte de connectivité multiple. À cet égard, l'homologie simpliciale est utilisée pour étudier la couverture et la connectivité des réseaux. Elle a déjà été appliquée avec succès pour les méthodes d'économie d'énergie.

De plus, l'optimisation de la puissance de transmission est une préoccupation majeure. En effet, une puissance de transmission trop faible entraîne des trous de couverture, ce qui diminue la fiabilité du réseau, et une puissance de transmission trop élevée pourrait augmenter les interférences et diminuer la capacité du réseau. Étant donné que les contraintes de couverture (impliquées par les exigences de fiabilité des slices) ne sont pas linéaires par nature, les méthodes d'optimisation classiques ne peuvent être utilisées pour résoudre le problème d'optimisation de la puissance. Des heuristiques spécifiques doivent être développées. Ce travail fournit une heuristique quasi-optimale pour les systèmes 5G, basée sur le recuit simulé. Elle peut être utilisée pour déterminer le budget de puissance requis pour un ensemble de slices déployés sur un réseau physique donné.

Les résultats obtenus par simulation peuvent être analysés pour comprendre le comportement du système 5G dans l'un ou l'autre schéma de déploiement. Cependant, les résultats pour les différents slices ne peuvent pas être strictement comparés car les paramètres de simulation ne sont pas les mêmes. Il est donc impossible de comparer directement l'impact des exigences des slices sur le budget de puissance. Néanmoins, les simulations permettent de comparer quatre cas. Dans les trois premiers cas, le réseau est dédié à un slice, et les paramètres associés sont choisis pour répondre au mieux aux exigences des slices. Dans le dernier cas, toutes les slices fonctionnent sur un seul réseau dont les paramètres assurent un équilibre entre les exigences des trois slices. On observe que la contrainte de couverture a un impacte considérable sur le budget de puissance requis.

Implémentation d'une sonde 5G

La diversité des terminaux a deux impacts sur l'infrastructure du réseau que le slicing doit gérer. Tout d'abord, étant donné que les différents terminaux ne se connectent pas au même réseau de

données, l'infrastructure doit mettre en œuvre des tunnels qui transportent les communications entre les terminaux et les réseaux de données. De plus, les différents terminaux auront un trafic utilisateur très hétérogène, ce qui implique que le réseau doit supporter de nombreux profils QoS différents.

Parmi ces trafics utilisateurs hétérogènes, certains ont des contraintes fortes en matière de fiabilité et de contrôle du réseau. La contrainte de contrôle du réseau exige que les profils de QoS qui peuvent être offerts à l'UE soient strictement maîtrisés par le réseau à chaque instant. C'est ici qu'un défi majeur est soulevé. En effet, l'objectif des réseaux 5G est de supporter ces divers trafics utilisateurs au dessus d'une infrastructure physique unique. Cependant, le contrôle complet du réseau ne peut être assuré lorsque l'infrastructure est partagée entre des tranches critiques et non critiques. Certaines techniques peuvent être utilisées pour augmenter la fiabilité et le contrôle du réseau lorsque l'infrastructure est partagée en dédiant des ressources aux tranches (étudiées dans le chapitre 6). Néanmoins, même si l'infrastructure peut être optimisée pour fournir le meilleur contrôle possible du réseau, elle ne peut pas atteindre un contrôle et une fiabilité absolu. La seule façon d'avoir un contrôle complet du réseau pour les trafics critiques des utilisateurs est de déployer une infrastructure dédiée aux terminaux associés, mais ce n'est pas la vision du système 5G.

Par conséquent, des outils de supervision sont nécessaires pour assurer le meilleur contrôle possible du réseau sur une infrastructure partagée entre des slices critiques et non critiques. Ces outils sont censés ajouter un deuxième niveau de fiabilité et de contrôle au premier niveau fourni par le réseau lui-même. Pour superviser le réseau, ces systèmes doivent être alimentés par le plus grand volume possible de données afin d'atteindre une grande précision. L'infrastructure peut fournir les données en utilisant les APIs du réseau. Cependant, pour les services hautement critiques et surtout pour les services qui peuvent être exposés à des problèmes de sécurité, il est intéressant que les outils de supervision s'appuient également sur des données provenant de l'extérieur du réseau. En effet, en cas d'attaque de sécurité sur le réseau, les données fournies par les APIs pourraient être corrompues, et les outils de supervision pourraient ne pas être en mesure de détecter l'attaque. D'autre part, l'utilisation de données provenant de l'extérieur du réseau garantit que les données ne sont pas corrompues même en cas d'attaque de sécurité.

Dans ce chapitre, le travail effectué pour la couche PHY est dérivé pour implémenter une sonde 5G autonome qui peut extraire toutes les communications descendantes et montantes de tous les UEs d'une cellule. L'objectif de cette sonde est de fournir aux outils de supervision des données provenant de l'extérieur du réseau afin d'augmenter la fiabilité et le contrôle du réseau. L'architecture de la sonde est représentée sur le Figure A.12.

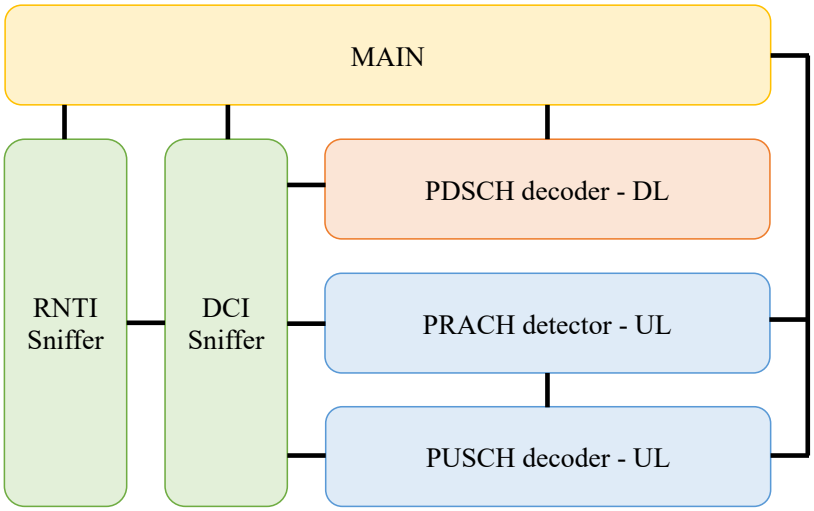


Figure A.12: Architecture de la sonde

Les données extraites de la sonde peuvent être utilisées pour répondre à différents cas d'utilisation. Tout d'abord, l'allocation des ressources et les MAC LCIDs peuvent être utilisés pour déduire le type de trafic transmis. Ces informations peuvent être utilisées pour estimer la charge du réseau par type de trafic mais aussi pour optimiser le déploiement du réseau. De plus, il y a beaucoup d'autres KPI

qui peuvent être extraits avec la sonde, comme la qualité du canal (dérivée du taux de codage et du schéma de modulation) ou l'état de la mémoire tampon des utilisateurs (dérivée du BSR MAC CE). Les outils de supervision peuvent utiliser toutes ces informations pour surveiller le réseau et valider que l'état du réseau est suffisamment bon pour respecter les différentes exigences des slices.

Conclusion

Dans cette thèse, nous avons montré que le slicing du réseau peut être implémenté avec les technologies actuelles. Cependant, cela requiert beaucoup d'attention et la prise en compte de tous les niveaux du réseau (comme les fonctions réseau, les ressources matérielles, le réseau de transport ou l'interface radio). De plus, nous avons exposé les principales procédures et algorithmes impliqués dans la couche physique 5G et proposé une implémentation concrète en open-source appelée *free5GRAN*.

Index

Bibliography

[1] N. C. Ericsson AB, Huawei Technologies Co. Ltd and Nokia, "Common Public Radio Interface: eCPRI Interface Specification," Tech. Rep. eCPRI V2.0, 05 2019.

[2] 3GPP, "Technical Specification Group Radio Access Network; NR; Overall description," 3GPP, TS 38.300 V15.8.0, 01 2020.

[3] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE Communications Letters*, vol. 15, no. 12, pp. 1378–1380, 2011.

[4] E. Dahlman, S. Parkvall, and J. Skold, *5G NR: The next generation wireless access technology*. Academic Press, 2020.

[5] 3GPP, "Technical Specification Group Radio Access Network; NR; Physical layer procedures for control," 3GPP, TS 38.213 V15.2.0, 06 2018.

[6] ——, "Technical Specification Group Radio Access Network; NR; System architecture for the 5G System," 3GPP, TS 23.501 V16.6.0, 10 2020.

[7] T. P. G. laboratory Institut Polytechnique de Paris. free5gran: an open-source 5g ran stack. [Online]. Available: https://github.com/free5G/free5GRAN

[8] 3GPP, "Technical Specification Group Radio Access Network; NR; Base Station (BS) radio transmission and reception ," 3GPP, TS 38.104 V16.4.0, 07 2020.

[9] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[10] P. A. Thangaraj. Introduction to polar codes: Polar transform. [Online]. Available: https://www.youtube.com/watch?v=rB0rhQKyV34

[11] 3GPP, "Technical Specification Group Radio Access Network; NR; Multiplexing and channel coding," 3GPP, TS 38.212 V15.2.0, 06 2018.

[12] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[13] P. A. Thangaraj. Low density parity check codes: definition, properties and introduction to protograph construction. [Online]. Available: https://www.youtube.com/watch?v=bAZnP4kdb44

[14] 3GPP, "Technical Specification Group Radio Access Network; Study on New Radio Access Technology; Radio Access Architecture and Interfaces ," 3GPP, TR 38.801 V2.0.0, 03 2017.

[15] "5g fundamentals: Functional split overview." [Online]. Available: https://www.hubersuhner.com/en/documents-repository/technologies/pdf/fiber-optics-documents/5g-fundamentals-functional-split-overview

[16] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "Srslte: An open-source platform for lte evolution and experimentation," in *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, ser. WiNTECH '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 25–32.

[17] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "Openairinterface: A flexible platform for 5g research," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, p. 33–38, Oct. 2014.

[18] Linux From Scratch Website. [Online]. Available: http://www.linuxfromscratch.org

[19] 3GPP, "Technical Specification Group Radio Access Network; NR; Radio Resource Control (RRC); Protocol specification," 3GPP, TS 38.331 V15.2.0, 06 2018.

[20] "5g/nr - rrc overview." [Online]. Available: https://www.sharetechnote.com/html/5G/5G_RRC_Overview.html

[21] 3GPP, "Technical Specification Group Radio Access Network; NR; Physical channels and modulation," 3GPP, TS 38.211 V15.2.0, 06 2018.

[22] M. Hua, M. Wang, K. W. Yang, and K. J. Zou, "Analysis of the frequency offset effect on zadoff–chu sequence timing performance," *IEEE Transactions on Communications*, vol. 62, no. 11, pp. 4024–4039, 2014.

[23] Y. Bhargava and K. Giridhar, "Efficient frequency offset estimation and tracking for reuse-1 ofdma systems," in *TENCON 2006 - 2006 IEEE Region 10 Conference*, 2006, pp. 1–4.

[24] 3GPP, "Technical Specification Group Radio Access Network; NR; Physical layer procedures for data," 3GPP, TS 38.214 V15.2.0, 06 2018.

[25] C. Johnson, *5G New Radio in Bullets*. Independently Published, 2019. [Online]. Available: https://books.google.fr/books?id=NoRjyAEACAAJ

[26] 3GPP, "Technical Specification Group Radio Access Network; NR; Medium Access Control (MAC) protocol specification," 3GPP, TS 38.321 V15.3.0, 09 2018.

[27] F. Linsalata, M. Magarini, and R. Ferrari, "On the extension of lte and lte-a prach receiver design to 5g new radio," in *International Balkan Conference on Communications and Networking*, 2019, pp. 1–5.

[28] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "Llr-based successive cancellation list decoding of polar codes," *IEEE Transactions on Signal Processing*, vol. 63, no. 19, pp. 5165–5179, 2015.

[29] T. T. B. Nguyen, T. Nguyen Tan, and H. Lee, "Efficient qc-ldpc encoder for 5g new radio," *Electronics*, vol. 8, no. 6, 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/6/668

[30] J. S. Yedidia, W. T. Freeman, Y. Weiss *et al.*, "Understanding belief propagation and its generalizations," *Exploring artificial intelligence in the new millennium*, vol. 8, no. 236-239, pp. 0018–9448, 2003.

[31] X. Cao, Y. Liu, and D. Hu, "Simplified llr algorithm for m-qam demodulation," *The Journal of Engineering*, vol. 2019, 07 2019.

[32] R. W. Lucky, "Automatic equalization for digital communication," *Bell System Technical Journal*, vol. 44, no. 4, pp. 547–588, 1965.

[33] L. Walkin, "Open source asn1c."

[34] USRP UHD Library Website. [Online]. Available: https://files.ettus.com/manual/index.html

[35] G. Garcia-Aviles, M. Gramaglia, P. Serrano, and A. Banchs, "Posens: A practical open source solution for end-to-end network slicing," *IEEE Wireless Communications*, vol. 25, no. 5, pp. 30–37, 2018.

[36] T. Girici, C. Zhu, J. R. Agre, and A. Ephremides, "Proportional fair scheduling algorithm in ofdma-based wireless systems with qos constraints," *Journal of communications and networks*, vol. 12, no. 1, pp. 30–42, 2010.

[37] D. Faggioli, M. Trimarchi, F. Checconi, M. Bertogna, and A. Mancina, "An implementation of the earliest deadline first algorithm in linux," in *Proceedings of the 2009 ACM Symposium on Applied Computing*, ser. SAC '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 1984–1989. [Online]. Available: https://doi.org/10.1145/1529282.1529723

[38] J. Bouron, S. Chevalley, B. Lepers, W. Zwaenepoel, R. Gouicem, J. Lawall, G. Muller, and J. Sopena, "The battle of the schedulers: FreeBSD ULE vs. linux CFS," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, Jul. 2018, pp. 85–96. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/bouron

[39] O. Sefraoui, M. Aissaoui, M. Eleuldj *et al.*, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, 2012.

[40] S. Xi, C. Li, C. Lu, C. D. Gill, M. Xu, L. T. Phan, I. Lee, and O. Sokolsky, "Rt-open stack: Cpu resource management for real-time cloud computing," in *2015 IEEE 8th International Conference on Cloud Computing*, 2015, pp. 179–186.

[41] S. Wu, L. Zhou, H. Sun, H. Jin, and X. Shi, "Poris: A scheduler for parallel soft real-time applications in virtualized environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 3, pp. 841–854, 2016.

[42] V. Yodaiken *et al.*, "The rtlinux manifesto," in *Proc. of the 5th Linux Expo*, 1999.

[43] Linux. cpuset(7) — linux manual page. [Online]. Available: https://man7.org/linux/man-pages/man7/cpuset.7.html

[44] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, "hwloc: A generic framework for managing hardware affinities in hpc applications," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2010, pp. 180–186.

[45] L. Abeni, A. Balsini, and T. Cucinotta, "Container-based real-time scheduling in the linux kernel," *SIGBED Rev.*, vol. 16, no. 3, p. 33–38, nov 2019. [Online]. Available: https://doi.org/10.1145/3373400.3373405

[46] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[47] Z. Li, M. Kihl, Q. Lu, and J. A. Andersson, "Performance overhead comparison between hypervisor and container based virtualization," in *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, 2017, pp. 955–962.

[48] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: A performance comparison," in *2015 IEEE International Conference on Cloud Engineering*, 2015, pp. 386–393.

[49] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *SIGMOD Rec.*, vol. 34, no. 4, p. 42–47, dec 2005. [Online]. Available: https://doi.org/10.1145/1107499.1107504

[50] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[51] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Transactions on Communications*, vol. 47, no. 5, pp. 673–680, 1999.

[52] Linux. Lat_ctx(8) - lmbench man page. [Online]. Available: http://www.bitmover.com/lmbench/lat_ctx.8.html

[53] L. Shiff. (2020) Real time vs batch processing vs stream processing. [Online]. Available: https://www.bmc.com/blogs/batch-processing-stream-processing-real-time/

[54] D. Flair. Batch processing vs real time processing – comparison. [Online]. Available: https://data-flair.training/blogs/batch-processing-vs-real-time-processing/

[55] J. Farkas, L. L. Bello, and C. Gunther, "Time-sensitive networking standards," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 20–21, 2018.

[56] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The segment routing architecture," in *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–6.

[57] IETF. Segment routing architecture. [Online]. Available: https://www.rfc-editor.org/rfc/pdfrfc/rfc8402.txt.pdf

[58] T. L. Kernel. Symmetric multi-processing. [Online]. Available: https://linux-kernel-labs.github.io/refs/heads/master/lectures/smp.html

[59] T. L. F. Projects. Data plane development kit. [Online]. Available: https://www.dpdk.org/

[60] IETF. A border gateway protocol. [Online]. Available: https://www.rfc-editor.org/info/rfc4271

[61] GSMA. (2020) 5g tdd synchronisation guidelines and recommendations for the coexistence of tdd networks in the 3.5 ghz range. [Online]. Available: https://www.gsma.com/spectrum/wp-content/uploads/2020/04/3.5-GHz-5G-TDD-Synchronisation.pdf

[62] D. G. Cattrysse and L. N. Van Wassenhove, "A survey of algorithms for the generalized assignment problem," Tech. Rep., 1990.

[63] M. Leconte, G. S. Paschos, P. Mertikopoulos, and U. C. Kozat, "A resource allocation framework for network slicing," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 2177–2185.

[64] N. Le, P. Martins, L. Decreusefond, and A. Vergne, "Simplicial homology based energy saving algorithms for wireless networks," in *IEEE International Conference on Communication Workshop*, 2015, pp. 166–172.

[65] V. Erceg, L. Greenstein, S. Tjandra, S. Parkoff, A. Gupta, B. Kulic, A. Julius, and R. Bianchi, "An empirically based path loss model for wireless channels in suburban environments," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 7, pp. 1205–1211, 1999.

[66] F. Yan, P. Martins, and L. Decreusefond, "Accuracy of homology based coverage hole detection for wireless sensor networks on sphere," *IEEE Transactions on Wireless Communications*, vol. 13, no. 7, pp. 3583–3595, 2014.

[67] M. Lanza, A. L. Gutiérrez, J. R. Pérez, J. Morgade, M. Domingo, L. Valle, P. Angueira, and J. Basterrechea, "Coverage optimization and power reduction in sfn using simulated annealing," *IEEE Transactions on Broadcasting*, vol. 60, no. 3, pp. 474–485, 2014.

[68] H. D. Trinh, A. Fernández Gambin, L. Giupponi, M. Rossi, and P. Dini, "Mobile traffic classification through physical control channel fingerprinting: A deep learning approach," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1946–1961, 2021.

[69] InfluxData. Influx db. [Online]. Available: https://www.influxdata.com/time-series-platform/

[70] C. N. C. Fundation. grpc, a high performance, open source universal rpc framework. [Online]. Available: https://grpc.io/

[71] Amarisoft. Amarisoft - test & measurements. [Online]. Available: https://www.amarisoft.com/products/test-measurements/

[72] "5g - waveform candidate - 5g." [Online]. Available: https://www.sharetechnote.com/html/5G/5G_RACH.html

[73] F. Hamidi-Sepehr, A. Nimbalker, and G. Ermolaev, "Analysis of 5g ldpc codes rate-matching design," in *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, 2018, pp. 1–5.

ECOLE DOCTORALE

Titre : 5G RAN: implémentation de la couche physique et découpage du réseau.

Mots clés : 5G, couche physique, slicing du réseau

Résumé : Une des évolutions de la 4G à la 5G est l'hétérogénéité des terminaux qui accèdent au réseau. Ces terminaux vont des smartphones aux véhicules connectés en passant par les capteurs pour l'agriculture. Étant donné que les contraintes et les exigences associées aux différents types de terminaux sont hétérogènes, il n'est pas facile de multiplexer les services qui leur sont associés sur une seule infrastructure physique. Le slicing est la technologie qui permet à l'infrastructure physique de fournir plusieurs réseaux logiques (appelés slices) pour servir les différents terminaux et services associés : cette thèse étudie le slicing et sa mise en œuvre au niveau RAN.
Une des principales questions soulevées par le slicing est l'allocation des ressources. En effet, de nombreux modèles existent pour l'allocation des ressources du RAN mais il manque des modèles qui prennent en compte les nouvelles contraintes impliquées par le slicing. La première contribution de cette thèse est de définir un nouveau modèle pour le slicing au niveau RAN. Ce modèle prend en compte différentes contraintes de slicing telles que la capacité, la densité des UEs, la latence et la fiabilité. L'homologie simpliciale est utilisée pour valider le respect des contraintes des slices. De plus, ce modèle est appliqué à l'optimisation de la puissance, qui est un aspect critique du déploiement du réseau.
Le deuxième défi abordé dans ce travail est la supervision et le contrôle du réseau. En effet, certains verticaux ont des exigences de contrôle très élevées, et le réseau lui-même pourrait ne pas être en mesure de satisfaire pleinement ces contraintes. Par conséquent, nous introduisons une sonde qui peut extraire des données du réseau pour alimenter des outils de supervision pour le contrôle et le suivi du réseau. Cette sonde est conçue pour être résiliente aux cyber-attaques et est donc indépendante du réseau.
La dernière contribution principale de cette thèse est l'introduction d'une couche physique 5G open-source appelée free5GRAN. La couche physique fournit toutes les procédures et algorithmes minimaux pour les communications entre le gNodeB et les UEs. La structure du projet est construite de manière à pouvoir facilement la modifier et mettre en place de nouvelles fonctionnalités. De plus, l'architecture logicielle est conçue de manière à ce que la couche physique soit modulaire et puisse être dérivée pour mettre en œuvre le split 7.2 de l'open-RAN.

Title : 5G RAN: physical layer implementation and network slicing.

Keywords : 5G, physical layer, network slicing

Abstract : A critical evolution from 4G to 5G is the heterogeneity of the terminals that connect the network. Those terminals range from smartphones to connected vehicles and sensors for agriculture. Given that the constraints and requirements associated with the different kinds of terminals are heterogeneous, it is not trivial to multiplex the services associated with them on top of a single physical infrastructure. Network slicing is the technology that enables the physical infrastructure to provide multiple logical networks (called network slices) to serve the various devices and associated services: this thesis studies network slicing and its implementation at the RAN level.
One main issue raised by network slicing is resource allocation. Indeed, many models exist for resource allocation of the RAN but we are missing models which take into account new constraints implied by network slicing. The first contribution of this thesis is to define a new model for network slicing at the RAN level. This model takes into account diverse slices constraints such as capacity, UEs density, latency, and reliability. Simplicial homology is used to validate slices constraints fulfillment. Furthermore, this model is applied to power optimization, which is a critical aspect of network deployment.
The second challenge addressed in this work is the network's supervision and control. Indeed, some verticals have ultra-high control requirements, and the network itself might not be able to satisfy this constraint fully. Therefore, we introduce a probe that can extract data from the network to feed supervision tools for the network's monitoring and control. This probe is designed to be resilient to cyber-attacks and is thus independent of the network.
The last main contribution of this thesis is the introduction of an open-source 5G physical layer called free5GRAN. The physical layer provides all the minimal procedures and algorithms for communications between the gNodeB and UEs. The project's structure is built so one can easily modify it and implement new features. Furthermore, the software architecture is designed so that the physical layer is modular and can be derived to implement the open-RAN split 7.2.

Institut Polytechnique de Paris
91120 Palaiseau, France